

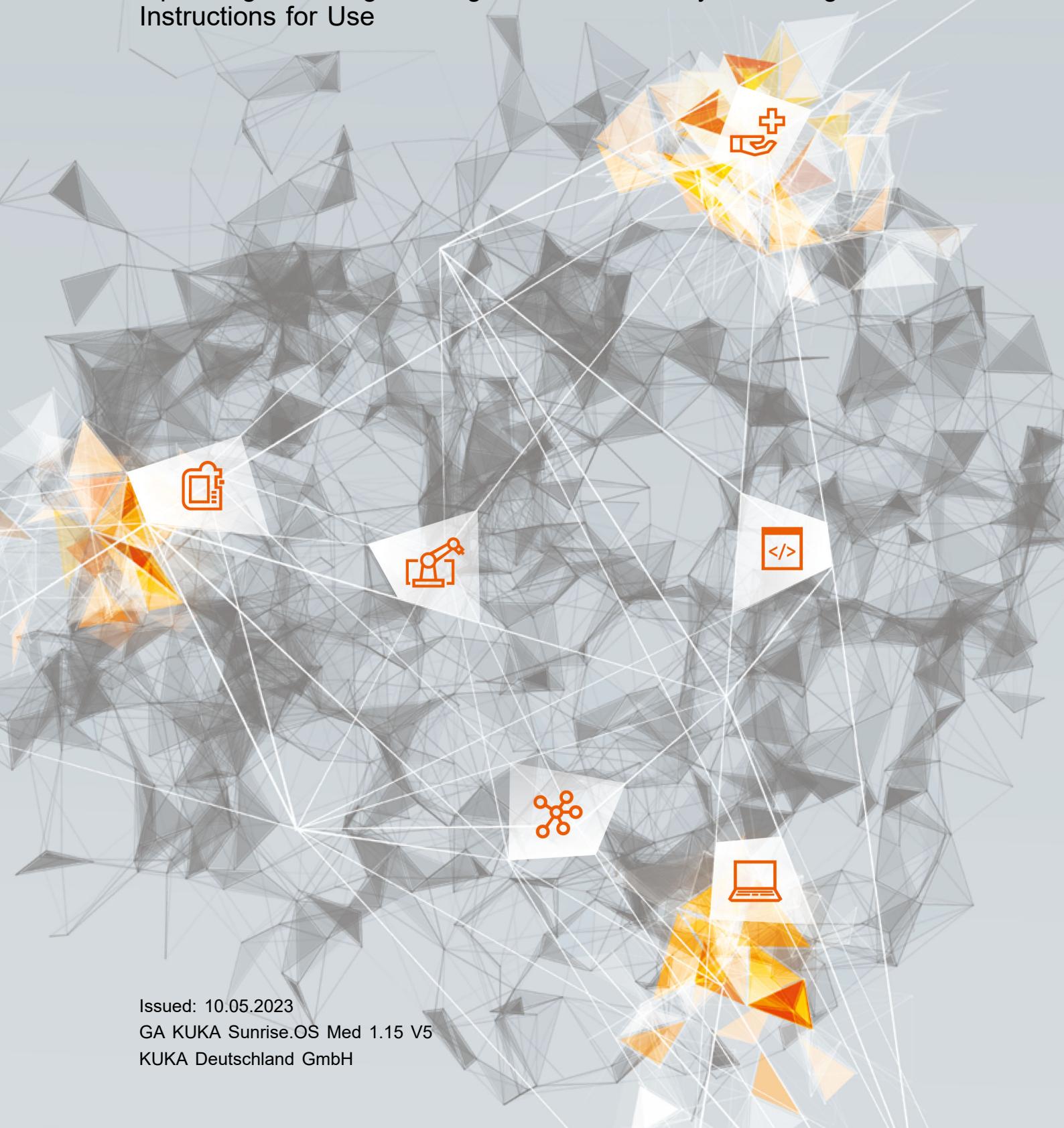
KUKA



System Software

KUKA Sunrise.OS Med 1.15, KUKA Sunrise.Workbench Med 1.15

Operating and Programming Instructions for System Integrators
Instructions for Use



Issued: 10.05.2023

GA KUKA Sunrise.OS Med 1.15 V5

KUKA Deutschland GmbH

© Copyright 2023

KUKA Deutschland GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

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

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

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

Subject to technical alterations without an effect on the function.

KIM-PS5-DOC

Translation of the original documentation

Publication: Pub GA KUKA Sunrise.OS Med 1.15 (PDF) en
PB10480

Book structure: GA KUKA Sunrise.OS Med 1.15 V5.2
BS9686

Version: GA KUKA Sunrise.OS Med 1.15 V5

Contents

1	Introduction.....	19
1.1	Target group.....	19
1.2	LBR Med documentation.....	19
1.3	Representation of warnings and notes.....	19
1.4	Trademarks.....	20
1.5	Terms used.....	20
1.6	Licenses.....	23
2	Product description.....	25
2.1	Overview of the robot system.....	25
2.2	Overview of the software components.....	25
2.3	Overview of KUKA Sunrise.OS.....	26
2.4	Overview of KUKA Sunrise.Workbench.....	27
2.5	Intended use of the System Software.....	28
3	Safety.....	29
3.1	General.....	29
3.1.1	Disclaimer.....	29
3.1.1.1	CB Test Certificate and CB Test Report.....	30
3.1.1.2	EC declaration of conformity and declaration of incorporation.....	30
3.1.2	Terms in the "Safety" chapter.....	31
3.2	Groups of persons.....	33
3.3	Workspace, safety zone and danger zone.....	34
3.4	Triggers for safety-oriented stop reactions.....	35
3.5	Safety functions.....	36
3.5.1	Safety-oriented functions.....	37
3.5.1.1	EMERGENCY STOP device.....	38
3.5.1.2	Enabling device.....	38
3.5.1.3	"Operator safety" signal.....	39
3.5.1.4	External EMERGENCY STOP device.....	40
3.5.1.5	External safety stop 1 (path-maintaining)	40
3.5.1.6	External enabling device.....	40
3.5.1.7	External safe operational stop.....	41
3.5.2	Non-safety-oriented functions.....	41
3.5.2.1	Mode selection.....	41
3.5.2.2	Velocity monitoring in T1.....	42
3.5.2.3	Software limit switches.....	42
3.6	Service phases of the robot system.....	43
3.7	Additional protective equipment.....	43
3.7.1	Jog mode.....	43
3.7.2	Labeling on the LBR Med.....	43
3.7.3	External safeguards.....	44
3.8	Safety measures.....	45
3.8.1	General safety measures.....	45
3.8.2	IT security.....	46
3.8.3	Transportation.....	47
3.8.4	Start-up and recommissioning.....	47

3.8.5	Manual mode.....	50
3.8.6	Automatic mode.....	51
3.8.7	Maintenance and repair.....	51
3.8.8	Decommissioning, storage and disposal.....	52
3.8.9	Safety measures for “single point of control”.....	53
4	Installing KUKA Sunrise.Workbench.....	55
4.1	PC system requirements.....	55
4.2	Installing Sunrise.Workbench.....	55
4.3	Uninstalling Sunrise.Workbench.....	55
5	Operation of KUKA Sunrise.Workbench.....	57
5.1	Starting Sunrise.Workbench.....	57
5.2	Overview of the user interface of Sunrise.Workbench.....	57
5.2.1	Repositioning the views.....	59
5.2.2	Closing views and files.....	59
5.2.3	Displaying perspectives.....	59
5.2.4	Toolbar of the Programming perspective.....	60
5.3	Creating a Sunrise project with a template.....	61
5.4	Sunrise applications.....	64
5.4.1	Application development.....	64
5.4.2	Creating a new Java package.....	64
5.4.3	Creating a robot application with a package.....	65
5.4.4	Creating a robot application for an existing package.....	65
5.4.5	Creating a new background application.....	65
5.4.5.1	Creating a background application with a package.....	66
5.4.5.2	Creating a background application for an existing package.....	66
5.4.6	Setting the robot application as the default application.....	67
5.5	Workspace.....	67
5.5.1	Creating a new workspace.....	67
5.5.2	Switching to an existing workspace.....	68
5.5.3	Switching between the most recently opened workspaces.....	68
5.5.4	Archiving projects.....	68
5.5.5	Loading projects from archive to the workspace.....	68
5.5.6	Loading projects from the directory to the workspace.....	69
5.6	Sunrise projects with referenced Java projects.....	69
5.6.1	Creating a new Java project.....	70
5.6.1.1	Inserting robot-specific class libraries in a Java project.....	70
5.6.2	Referencing Java projects.....	71
5.6.3	Canceling the reference to Java projects.....	71
5.7	Renaming an element in the Package Explorer.....	71
5.7.1	Renaming a project or Java package.....	72
5.7.2	Renaming a Java file.....	72
5.8	Removing an element from Package Explorer.....	72
5.8.1	Deleting an element from a project.....	72
5.8.2	Removing a project from Package Explorer.....	72
5.8.3	Deleting a project from the workspace.....	73
5.9	Activating the automatic change recognition.....	73
5.10	Displaying release notes.....	73

6	Operating the KUKA smartPAD.....	75
6.1	smartPAD.....	75
6.1.1	Front view.....	75
6.1.2	Rear of smartPAD.....	78
6.2	smartPAD-2.....	79
6.2.1	Front of smartPAD-2.....	79
6.2.2	Rear of smartPAD-2.....	81
6.3	Disconnecting and connecting the smartPAD.....	82
6.3.1	Disconnecting the smartPAD.....	82
6.3.2	Connecting the smartPAD.....	83
6.4	Update of the smartPAD software.....	84
6.5	KUKA smartHMI user interface.....	85
6.5.1	Navigation bar.....	86
6.5.2	Status display.....	87
6.5.3	Keypad.....	88
6.5.4	Station level.....	88
6.5.5	Robot level.....	90
6.6	Calling the main menu.....	92
6.7	Setting the user interface language.....	94
6.8	User groups.....	95
6.8.1	Changing user group.....	96
6.9	CRR mode – controlled robot retraction.....	96
6.10	Changing the operating mode.....	97
6.11	Activating the user keys.....	97
6.12	Resuming the safety controller.....	98
6.13	Coordinate systems.....	99
6.14	“Override” window.....	100
6.15	“Jogging type” window.....	101
6.16	Jogging the robot.....	103
6.16.1	“Jogging options” window.....	103
6.16.2	Setting the jog override.....	105
6.16.3	Axis-specific jogging with the jog keys.....	106
6.16.4	Cartesian jogging with the jog keys.....	107
6.16.4.1	Null space motion.....	108
6.17	Manually guiding the robot.....	109
6.18	Frame management.....	109
6.18.1	“Frames” view.....	109
6.18.2	Creating a frame.....	111
6.18.3	Reteaching frames.....	112
6.18.4	Teaching a frame with the hand guiding device.....	114
6.18.5	Manually addressing frames.....	115
6.19	Program execution.....	115
6.19.1	Selecting a robot application.....	115
6.19.2	Setting the program run mode.....	117
6.19.2.1	Program run modes.....	118
6.19.3	Setting the manual override.....	118
6.19.4	Starting a robot application forwards (manually).....	119
6.19.5	Starting a robot application forwards (automatically).....	119

6.19.6	Resetting a robot application.....	120
6.19.7	Repositioning the robot after leaving the path.....	120
6.19.8	Starting/stopping a background application manually.....	121
6.19.8.1	Stopping a background application manually.....	121
6.19.8.2	Starting a background application manually.....	122
6.20	Display functions.....	122
6.20.1	Displaying the end frame of the motion currently being executed.....	122
6.20.2	Displaying the axis-specific actual position.....	123
6.20.3	Displaying the Cartesian actual position.....	124
6.20.4	Displaying axis-specific torques.....	125
6.20.5	Displaying an I/O group and changing the value of an output.....	125
6.20.6	Displaying information about the robot and robot controller.....	128
6.21	Backup Manager.....	128
6.21.1	“Backup Manager” view.....	129
6.21.2	Backing up data manually.....	132
6.21.3	Restoring data manually.....	132
6.21.4	Configuring the network path for restoration.....	132
7	Start-up and recommissioning.....	135
7.1	Switching the robot controller on/off.....	135
7.1.1	Switching on the robot controller.....	135
7.1.2	Switching the robot controller off.....	135
7.2	Update of the smartPAD software.....	135
7.3	Performing a PDS firmware update.....	136
7.4	Position mastering.....	137
7.4.1	Performing position mastering.....	137
7.4.2	Manually unmastering axes.....	138
7.5	Calibration.....	138
7.5.1	Tool calibration.....	138
7.5.1.1	TCP calibration: XYZ 4-point method.....	140
7.5.1.2	Defining the orientation: ABC 2-point method.....	142
7.5.1.3	Defining the orientation: ABC world method.....	144
7.5.2	Base calibration: 3-point method	145
7.6	Determining tool load data.....	147
8	Brake test.....	151
8.1	Special behavior of the brakes of the LBR Med.....	151
8.2	Cyclical brake test.....	151
8.2.1	Functionality of the cyclical brake test.....	153
8.2.2	Interfaces of the cyclical brake test.....	153
8.2.3	Interface IBrakeTestMonitorListener.....	158
8.2.4	Checking the LOG files.....	159
9	Project management.....	161
9.1	Overview of a Sunrise project.....	161
9.2	Frame management.....	161
9.2.1	Creating a frame for an application.....	162
9.2.2	Designating a frame as a base.....	163
9.2.3	Moving a frame.....	164
9.2.4	Deleting a frame.....	164

9.2.5	Displaying/editing frame properties.....	165
9.2.5.1	Frame properties – General tab.....	166
9.2.5.2	Frame properties – Transformation tab.....	166
9.2.5.3	Frame properties – Redundancy tab.....	166
9.2.5.4	Frame properties – Teach information tab.....	166
9.2.5.5	Frame properties – Measurement tab.....	167
9.2.6	Inserting a frame in a motion instruction.....	167
9.3	Object management.....	168
9.3.1	Geometric structure of tools.....	168
9.3.2	Geometric structure of workpieces.....	169
9.3.3	Creating a tool or workpiece.....	169
9.3.4	Entering load data.....	170
9.3.5	Displaying/editing object properties.....	171
9.3.5.1	Object properties – General tab.....	171
9.3.5.2	Object properties – Load data tab.....	171
9.3.6	Creating a frame for a tool or workpiece.....	172
9.3.7	Displaying/editing frame properties.....	173
9.3.7.1	Frame properties – General tab.....	173
9.3.7.2	Frame properties – Transformation tab.....	173
9.3.7.3	Frame properties – Measurement tab.....	174
9.3.7.4	Frame properties – Safety tab.....	174
9.3.8	Defining a default motion frame.....	175
9.3.9	Copying an object template.....	176
9.3.10	Configuring a safety-oriented tool.....	176
9.3.10.1	Tool properties – Load data tab.....	179
9.3.10.2	Frame properties – Transformation tab.....	179
9.3.10.3	Frame properties – Safety tab.....	180
9.3.10.4	Tool properties – Safety tab.....	180
9.3.11	Safety-oriented use of workpieces.....	181
9.3.11.1	Entering workpiece load data.....	183
9.3.11.2	Workpiece properties – Load data tab.....	183
9.3.11.3	Configuring the mass of the heaviest workpiece.....	184
9.4	User management.....	184
9.4.1	Overview of Sunrise.RolesRights.....	185
9.4.2	Changing and activating the password.....	186
9.5	Project synchronization.....	187
9.5.1	Transferring the project to the robot controller.....	188
9.5.2	Updating the project.....	189
9.6	Loading the project from the robot controller.....	190
10	Station configuration and installation.....	193
10.1	Station configuration overview.....	193
10.2	Adapting the network settings.....	193
10.3	“Software” tab.....	194
10.3.1	Eliminating errors in the software catalog.....	194
10.4	“Configuration” tab.....	196
10.4.1	IP address range for KUKA Line Interface (KLI).....	196
10.4.2	Handguiding Support.....	196
10.4.3	General safety settings.....	197

10.4.4	Configuration parameters for calibration.....	198
10.4.5	Configuration parameters for Backup Manager.....	198
10.5	“Installation” tab.....	200
10.5.1	Installing System Software on the robot controller.....	200
10.6	Loading an old project and converting the safety configuration.....	202
10.7	Option packages.....	203
10.7.1	Installing the option package.....	204
10.7.2	Installing or updating the virus scanner.....	205
10.7.3	Installing a language package.....	206
10.7.4	Uninstalling the option package.....	206
10.7.4.1	Instructions for uninstallation of safety options.....	207
10.7.4.2	Removing an option package from the robot controller.....	207
11	Bus configuration.....	209
11.1	Overview: Configuration and I/O mapping in WorkVisual.....	209
11.2	Overview of field buses.....	209
11.3	Creating a new I/O configuration.....	210
11.4	Opening an existing I/O configuration.....	210
11.5	Creating Sunrise I/Os.....	211
11.5.1	“Create I/O signals” window.....	212
11.5.2	Creating an I/O group and inputs/outputs within the group.....	214
11.5.3	Editing an I/O group.....	214
11.5.4	Deleting an I/O group.....	215
11.5.5	Changing an input/output of a group.....	215
11.5.6	Deleting an input/output of a group.....	215
11.5.7	Exporting an I/O group as a template.....	216
11.5.8	Importing an I/O group from a template.....	216
11.6	Mapping the bus I/Os.....	217
11.6.1	I/O Mapping window.....	217
11.6.2	Buttons in the “I/O Mapping” window.....	218
11.6.3	Mapping Sunrise I/Os.....	219
11.7	Exporting the I/O configuration to the Sunrise project.....	219
12	External control.....	221
12.1	Overview of external controller.....	221
12.2	Configuring the external controller via the I/O system.....	221
12.3	Configuring the external controller via the UDP interface.....	222
12.4	External controller input signals.....	222
12.5	External controller output signals.....	223
12.6	Signal diagrams.....	224
12.7	Configuring the external controller in the project settings.....	225
12.7.1	Input/output parameters of the I/O interface.....	227
12.7.2	Input/output parameters of the UDP interface.....	227
12.8	Formatting of the UDP data packets.....	227
12.8.1	Status messages of the robot controller.....	228
12.8.2	Controller messages of the external client.....	230
12.9	External control via UDP – Start-up example.....	231
12.9.1	Starting up the external controller.....	231
12.9.2	Programming the external controller.....	232

12.10	Configuring the signal outputs for a project that is not externally controlled.....	234
12.10.1	Output parameters of the I/O interface.....	235
12.10.2	Output parameters of the UDP interface.....	235
13	Safety configuration.....	237
13.1	Overview of safety configuration.....	237
13.2	Software classification.....	238
13.2.1	Requirements on SOUP components.....	239
13.2.1.1	Development process.....	239
13.2.1.2	Software requirements.....	240
13.2.1.3	Retrofitting SOUP components.....	240
13.2.1.4	SOUP components.....	240
13.2.2	Effects of software classification.....	241
13.2.2.1	Example: Tool changing.....	241
13.2.2.2	Example: Holding the position.....	241
13.3	Single fault safety.....	242
13.4	Safety concept.....	243
13.5	Safety-oriented reactions.....	244
13.5.1	Time response of safety-oriented outputs.....	245
13.6	Safety interfaces.....	245
13.7	Permanent Safety Monitoring.....	246
13.8	Event-driven Safety Monitoring.....	247
13.9	Atomic Monitoring Functions.....	248
13.9.1	Standard AMFs.....	249
13.9.2	Parameterizable AMFs.....	250
13.9.3	Extended AMFs.....	252
13.9.4	Availability of the AMFs depending on the kinematic system.....	253
13.10	Worst-case reaction times of the safety functions in the case of a single fault.	254
13.10.1	Worst-case reaction times of the LBR Med monitoring functions.....	255
13.11	Deactivation of safety functions via an input.....	258
13.12	Safety configuration (SafetyConfiguration.sconf file).....	260
13.12.1	Overview of safety configuration and start-up.....	260
13.12.2	Opening the safety configuration.....	261
13.12.2.1	Evaluating the safety configuration.....	262
13.12.2.2	Overview of the graphical user interface for the safety configuration.....	262
13.12.3	Configuring the safety functions of the PSM mechanism.....	264
13.12.3.1	Opening the Customer PSM table.....	264
13.12.3.2	Creating safety functions for the PSM mechanism.....	265
13.12.3.3	Deleting safety functions of the PSM mechanism.....	266
13.12.3.4	Editing existing safety functions of the PSM mechanism.....	266
13.12.4	Configuring the safe states of the ESM mechanism.....	267
13.12.4.1	Adding a new ESM state.....	267
13.12.4.2	Opening a table for an ESM state.....	267
13.12.4.3	Deleting an ESM state.....	269
13.12.4.4	Creating a safety function for the ESM state.....	269
13.12.4.5	Deleting a safety function of an ESM state.....	269
13.12.4.6	Editing an existing safety function of an ESM state.....	270
13.12.4.7	Deactivating the ESM mechanism.....	270
13.12.4.8	Switching between ESM states.....	270

13.12.5	Mapping safety-oriented tools.....	271
13.13	Activating the safety configuration.....	273
13.13.1	Activating the safety configuration.....	274
13.13.2	Restoring the safety configuration.....	274
13.13.3	Deactivating the safety configuration.....	274
13.14	Using and parameterizing the AMFs.....	275
13.14.1	Evaluating the safety equipment on the KUKA smartPAD.....	275
13.14.2	Evaluating the operating mode.....	275
13.14.3	Evaluating the motion enable.....	276
13.14.4	Monitoring safe inputs.....	276
13.14.5	Manual guidance with enabling device and velocity monitoring.....	277
13.14.5.1	Monitoring of enabling switches on hand guiding devices.....	277
13.14.5.2	Monitoring functions during manual guidance.....	279
13.14.5.3	Velocity monitoring during manual guidance.....	280
13.14.6	Evaluating the position referencing.....	280
13.14.7	Evaluating the torque referencing.....	281
13.14.8	Velocity monitoring functions.....	282
13.14.8.1	Defining axis-specific velocity monitoring.....	282
13.14.8.2	Defining Cartesian velocity monitoring.....	283
13.14.8.3	Direction-specific monitoring of Cartesian velocity.....	285
13.14.9	Monitoring spaces.....	288
13.14.9.1	Defining Cartesian workspaces.....	290
13.14.9.2	Defining Cartesian protected spaces.....	292
13.14.9.3	Defining axis-specific monitoring spaces.....	295
13.14.10	Monitoring the tool orientation.....	296
13.14.11	Standstill monitoring (safe operational stop).....	299
13.14.12	Activation delay for safety function.....	300
13.14.13	Monitoring of forces and torques.....	300
13.14.13.1	Axis torque monitoring.....	300
13.14.13.2	Collision detection.....	301
13.14.13.3	TCP force monitoring.....	303
13.14.13.4	Direction-specific monitoring of the external force on the TCP.....	304
13.15	Example of a safety configuration.....	308
13.15.1	Task.....	308
13.15.2	Requirements.....	309
13.15.3	Suggested solution for the task.....	309
13.16	Position and torque referencing.....	312
13.16.1	Position referencing.....	312
13.16.2	Torque referencing.....	313
13.16.3	Creating an application for position and torque referencing.....	315
13.16.4	External position referencing.....	316
13.16.4.1	Configuring the input for external position referencing.....	316
13.17	Safety acceptance overview.....	317
13.17.1	Checklist – System safety functions.....	317
13.17.2	Tool selection table checklist.....	321
13.17.3	Checklists for safety-oriented tools.....	323
13.17.3.1	Pickup frame of fixed tools.....	323
13.17.3.2	Pickup frames of activatable tools.....	324
13.17.3.3	Tool orientation.....	325

13.17.3.4	Points and orientation for tool-related velocity component.....	326
13.17.3.5	Geometry data of the tool.....	326
13.17.3.6	Load data of the tool.....	327
13.17.4	Checklist for rows used in table Customer PSM.....	328
13.17.5	Checklists for ESM states.....	329
13.17.5.1	Used ESM states.....	329
13.17.5.2	Non-used ESM states.....	330
13.17.6	Checklists for AMFs used.....	330
13.17.6.1	AMF smartPAD Emergency Stop.....	331
13.17.6.2	AMF smartPAD enabling switch inactive.....	331
13.17.6.3	AMF smartPAD enabling switch panic active.....	331
13.17.6.4	AMF Hand guiding device enabling inactive.....	331
13.17.6.5	AMF Hand guiding device enabling active.....	332
13.17.6.6	AMF Test mode.....	332
13.17.6.7	AMF Automatic mode.....	332
13.17.6.8	AMF Reduced-velocity mode.....	332
13.17.6.9	AMF High-velocity mode.....	332
13.17.6.10	AMF Motion enable.....	333
13.17.6.11	AMF Input signal.....	333
13.17.6.12	AMF Standstill monitoring of all axes.....	333
13.17.6.13	AMF Axis torque monitoring.....	333
13.17.6.14	AMF Axis velocity monitoring.....	334
13.17.6.15	AMF Position referencing.....	334
13.17.6.16	AMF Torque referencing.....	334
13.17.6.17	AMF Axis range monitoring.....	334
13.17.6.18	AMF Cartesian velocity monitoring.....	335
13.17.6.19	AMF Cartesian workspace monitoring / Cartesian protected space monitoring.	335
13.17.6.20	AMF Collision detection.....	336
13.17.6.21	AMF TCP force monitoring.....	337
13.17.6.22	AMF Base-related TCP force component.....	337
13.17.6.23	AMF Time delay.....	338
13.17.6.24	AMF Tool orientation.....	338
13.17.6.25	AMF Tool-related velocity component.....	340
13.17.7	Checklists – safety-oriented project settings.....	341
13.17.7.1	smartPAD unplugging allowed.....	341
13.17.7.2	Allow muting via input.....	341
13.17.7.3	Allow external position referencing.....	342
13.17.7.4	Mass of the heaviest workpiece.....	342
13.17.8	Creating a safety configuration report.....	343
14	Basic principles of motion programming.....	345
14.1	Overview of motion types.....	345
14.2	PTP motion type.....	345
14.3	LIN motion type	346
14.4	CIRC motion type.....	346
14.5	SPL motion type.....	347
14.6	Spline motion type.....	347
14.6.1	Velocity profile for spline motions.....	348
14.6.2	Modifications to spline blocks.....	350

14.6.3	LIN-SPL-LIN transition.....	352
14.7	Manual guidance motion type.....	353
14.8	Approximate positioning.....	354
14.9	Orientation control with LIN, CIRC, SPL.....	356
14.9.1	Orientation control reference system for CIRC.....	358
14.9.2	Combination of reference system and orientation type for CIRC.....	359
14.10	Redundancy information.....	361
14.10.1	Redundancy angle.....	362
14.10.2	Status.....	362
14.10.3	Turn.....	363
14.11	Singularities.....	363
14.11.1	Kinematic singularities.....	363
14.11.2	System-dependent singularities.....	365
15	Programming.....	367
15.1	Java Editor.....	367
15.1.1	Opening a robot application in the Java Editor.....	367
15.1.2	Structure of a robot application.....	367
15.1.3	Edit functions.....	368
15.1.3.1	Renaming variables.....	368
15.1.3.2	Auto-complete.....	369
15.1.3.3	Templates – Fast entry of Java statements.....	369
15.1.3.4	Creating user-specific templates.....	370
15.1.3.5	Extracting methods.....	370
15.1.4	Displaying Javadoc information.....	371
15.1.4.1	Configuration of the Javadoc browser.....	373
15.2	Symbols and fonts.....	376
15.3	Data types.....	376
15.3.1	Declaration.....	377
15.3.2	Initialization.....	378
15.3.2.1	Primitive data types.....	378
15.3.2.2	Complex data types.....	378
15.3.3	Dependency injection.....	379
15.3.3.1	Dependency injection for Sunrise types.....	380
15.3.3.2	Dependency injection for dedicated types.....	382
15.4	Requesting individual values of a vector.....	385
15.5	Network communication via UDP and TCP/IP.....	386
15.6	Motion programming: PTP, LIN, CIRC.....	386
15.6.1	Synchronous and asynchronous motion execution.....	386
15.6.2	PTP.....	387
15.6.3	LIN.....	388
15.6.4	CIRC.....	388
15.6.5	LIN REL.....	389
15.6.6	MotionBatch.....	390
15.7	Motion programming: spline.....	391
15.7.1	Programming tips for spline motions.....	391
15.7.2	Creating a CP spline block.....	392
15.7.3	Creating a JP spline block.....	393
15.7.4	Using spline in a motion instruction.....	394

15.8	Overview of motion parameters (PTP, LIN, CIRC, SPL, Spline).....	394
15.8.1	Programming axis-specific motion parameters.....	397
15.9	Programming manual guidance.....	398
15.9.1	Overview of motion parameters (manual guidance).....	399
15.9.2	Axis limitation for manual guidance.....	400
15.9.3	Velocity limitation for manual guidance.....	402
15.10	Using tools and workpieces in the program.....	403
15.10.1	Integrating tools and workpieces.....	403
15.10.2	Attaching tools and workpieces to the robot.....	405
15.10.2.1	Attaching a tool to the robot flange.....	405
15.10.2.2	Attaching a workpiece to other objects.....	406
15.10.2.3	Detaching objects.....	408
15.10.3	Moving tools and workpieces.....	408
15.10.4	Integrating dedicated object classes with dependency injection.....	409
15.10.5	Transferring workpiece load data to the safety controller.....	412
15.11	Using inputs/outputs in the program.....	415
15.11.1	Integrating an I/O group.....	416
15.11.2	Reading inputs/outputs.....	417
15.11.3	Setting outputs.....	418
15.12	Requesting axis torques.....	419
15.13	Reading Cartesian forces and torques.....	420
15.13.1	Requesting external Cartesian forces and torques.....	421
15.13.2	Requesting forces and torques individually.....	422
15.13.3	Checking the reliability of the calculated values.....	422
15.14	Requesting the robot position.....	424
15.14.1	Requesting the axis-specific robot position.....	425
15.14.2	Requesting the Cartesian actual or setpoint position.....	425
15.14.3	Requesting the Cartesian setpoint/actual value difference.....	427
15.15	HOME position.....	428
15.15.1	Changing the HOME position.....	428
15.16	Requesting system states.....	429
15.16.1	Requesting the HOME position.....	429
15.16.2	Requesting the mastering state.....	430
15.16.3	Checking “ready for motion”.....	431
15.16.3.1	Reacting to changes in the “ready for motion” signal.....	431
15.16.4	Checking the robot activity.....	432
15.16.5	Requesting the state of safety signals.....	433
15.16.5.1	Requesting the referencing state.....	435
15.16.5.2	Reacting to a change in state of safety signals.....	436
15.17	Changing and requesting the program run mode.....	437
15.18	Changing and requesting the override.....	438
15.18.1	Reacting to an override change.....	439
15.19	Overview of conditions.....	440
15.19.1	Complex conditions.....	442
15.19.2	Axis torque condition.....	443
15.19.3	Force condition.....	444
15.19.3.1	Condition for Cartesian force from all directions.....	446
15.19.3.2	Condition for normal force.....	447
15.19.3.3	Condition for shear force.....	448

15.19.4	Force component condition.....	450
15.19.5	Condition for Cartesian torque.....	452
15.19.5.1	Condition for Cartesian torque from all directions.....	454
15.19.5.2	Condition for torque.....	455
15.19.5.3	Condition for tilting torque.....	456
15.19.6	Torque component condition.....	457
15.19.7	Path-related condition.....	458
15.19.8	Distance condition.....	461
15.19.8.1	Distance component condition.....	462
15.19.9	Condition for Boolean signals.....	463
15.19.10	Condition for the range of values of a signal.....	463
15.20	Break conditions for motion commands.....	464
15.20.1	Defining break conditions.....	464
15.20.2	Evaluating the break conditions.....	466
15.20.2.1	Requesting a break condition.....	466
15.20.2.2	Requesting the robot position at the time of termination.....	467
15.20.2.3	Requesting a terminated motion (spline block, MotionBatch).....	468
15.21	Path-related switching actions (trigger).....	469
15.21.1	Programming triggers.....	469
15.21.2	Programming a path-related switching action.....	470
15.21.3	Evaluating trigger information.....	471
15.22	Monitoring of processes.....	473
15.22.1	Listener for monitoring conditions.....	473
15.22.2	Creating a listener object to monitor the condition.....	474
15.22.3	Registering a listener for notification of change in state.....	475
15.22.4	Activating or deactivating the notification service for listeners.....	477
15.22.5	Programming example for monitoring.....	477
15.23	Blocking wait for condition.....	478
15.24	Recording and evaluating data.....	479
15.24.1	Creating an object for data recording.....	480
15.24.2	Specifying data to be recorded.....	481
15.24.3	Starting data recording.....	483
15.24.4	Ending data recording.....	485
15.24.5	Requesting states from the DataRecorder object.....	485
15.24.6	Example program for data recording.....	486
15.25	Defining user keys.....	487
15.25.1	Creating a user key bar.....	488
15.25.2	Adding user keys to the bar.....	489
15.25.3	Defining the function of a user key.....	491
15.25.4	Labeling and graphical assignment of the user key bar.....	493
15.25.4.1	Assigning a text element.....	494
15.25.4.2	Assigning an LED icon.....	495
15.25.5	Identifying safety-critical user keys.....	496
15.25.6	Publishing a user key bar.....	497
15.26	Message programming.....	498
15.26.1	Programming user messages.....	498
15.26.2	Programming user dialogs.....	499
15.27	Program execution control.....	501
15.27.1	Pausing an application.....	501

15.27.2	Pausing motion execution.....	502
15.27.3	Canceling a motion command.....	502
15.27.4	FOR loop.....	503
15.27.5	WHILE loop.....	504
15.27.6	DO WHILE loop.....	505
15.27.7	IF ELSE branch.....	506
15.27.8	SWITCH branch.....	508
15.27.9	Examples of nested loops.....	511
15.28	Continuing a paused application in Automatic mode (recovery).....	512
15.29	Error treatment.....	514
15.29.1	Handling of failed motion commands.....	514
15.29.2	Handling of failed synchronous motion commands.....	514
15.29.3	Handling of failed asynchronous motion commands.....	516
15.30	Overview of additional Med functions.....	519
15.30.1	Creating a Sunrise Med project using a template.....	519
15.30.2	Robot status overview.....	520
15.30.2.1	Polling the robot state.....	520
15.30.2.2	Robot status log.....	521
15.30.2.3	Status indicator on the smartHMI.....	523
15.30.3	Mastering axes.....	524
15.30.4	Jog mode.....	526
16	Background tasks.....	533
16.1	Using background tasks.....	533
16.2	Cyclic background task.....	535
16.3	Non-cyclic background task.....	538
16.4	Data exchange between tasks.....	538
16.4.1	Declaring task functions.....	540
16.4.2	Implementing task functions.....	540
16.4.3	Creating the providing task.....	542
16.4.4	Using task functions.....	544
17	KUKA Sunrise.EnhancedVelocityControl.....	549
17.1	Overview of KUKA Sunrise.EnhancedVelocityController.....	549
17.2	“Brake” safety reaction.....	550
17.3	Cartesian velocity limitation via application.....	552
17.3.1	Setting and deactivating velocity limitation.....	552
17.3.2	Requesting information about velocity limitation functions.....	553
18	KUKA Sunrise.StatusController.....	557
18.1	Overview of KUKA Sunrise.StatusController.....	557
18.1.1	Predefined status groups.....	558
18.1.2	Creating status and status groups.....	559
18.1.3	Using the IStatusController interface.....	560
18.1.4	Setting and deleting the status via the status monitor.....	561
18.1.5	Implementing a status listener.....	562
19	Programming with a compliant robot.....	567
19.1	Sensors and control.....	567
19.2	Overview of servo controllers.....	567

19.3	Using controllers in robot applications.....	568
19.3.1	Creating a controller object.....	568
19.3.2	Defining controller parameters.....	568
19.3.3	Transferring the controller object as a motion parameter.....	569
19.4	Position controller.....	569
19.5	Cartesian impedance controller.....	569
19.5.1	Calculation of the forces on the basis of Hooke's law.....	570
19.5.2	Parameterization of the Cartesian impedance controller.....	572
19.5.2.1	Representation of Cartesian degrees of freedom.....	573
19.5.2.2	Defining controller parameters for individual degrees of freedom.....	574
19.5.2.3	Controller parameters specific to the degrees of freedom.....	575
19.5.2.4	Controller parameters independent of the degrees of freedom.....	576
19.6	Cartesian impedance controller with overlaid force oscillation.....	579
19.6.1	Overlaying a simple force oscillation.....	580
19.6.2	Overlaying superposed force oscillations (Lissajous curves).....	580
19.6.3	Parameterization of the impedance controller with overlaid force oscillation.....	581
19.6.3.1	Controller parameters specific to the degrees of freedom.....	582
19.6.3.2	Controller parameters independent of the degrees of freedom.....	585
19.7	Static methods for impedance controller with superposed force oscillation.....	587
19.7.1	Overlaying a constant force.....	588
19.7.2	Overlaying a simple force oscillation.....	588
19.7.3	Overlaying a Lissajous oscillation.....	589
19.7.4	Overlaying a spiral-shaped force oscillation.....	591
19.8	Axis-specific impedance controller.....	592
19.8.1	Parameterization of the axis-specific impedance controller.....	593
19.8.2	Methods of the axis-specific impedance controller.....	593
19.9	Holding the position under servo control.....	595
20	Diagnosis.....	597
20.1	Field bus diagnosis.....	597
20.1.1	Displaying general field bus errors.....	597
20.1.2	Displaying the error state of I/Os and I/O groups.....	597
20.2	Displaying a log.....	597
20.2.1	"Log" view.....	598
20.2.2	Filtering log entries.....	600
20.3	Displaying error messages.....	601
20.4	Displaying messages of the virus scanner.....	603
20.5	Collecting diagnostic information for error analysis at KUKA.....	604
20.5.1	Creating a diagnosis package with the smartHMI.....	604
20.5.2	Creating a diagnosis package with the smartPAD.....	604
20.5.3	Creating a diagnosis package with Sunrise.Workbench.....	605
20.5.4	Loading existing diagnosis packages from the robot controller.....	605
21	Remote debugging.....	607
21.1	Debugging session sequence.....	607
21.1.1	Remote debugging of tasks.....	608
21.1.2	Starting the debugging session.....	609
21.1.3	Ending the debugging session.....	610
21.2	Debugging tasks.....	610

21.2.1	Remote debugging of a robot application.....	611
21.2.2	Remote debugging of a background task.....	612
21.3	Fundamentals of remote debugging.....	613
21.3.1	Overview of user interface – “Debugging” perspective.....	613
21.3.2	Break points.....	614
21.3.2.1	Creating and deleting break points.....	615
21.3.2.2	Deactivating and activating break points.....	616
21.3.2.3	Editing the properties of the break points.....	616
21.3.2.4	“Break points” view.....	617
21.3.2.5	Conditional break point.....	618
21.3.2.6	Suspend thread property.....	620
21.3.3	Command pointer.....	620
21.3.4	“Debugging” view.....	621
21.3.5	Overview of the toolbar in the “Debugging” view.....	623
21.3.5.1	Continuing execution (Resume).....	624
21.3.5.2	Jump into the method (Step in).....	624
21.3.5.3	Executing a method completely (Step over).....	625
21.3.5.4	Terminating the executed method (Step back).....	626
21.3.5.5	Executing code sections again (Back to frame).....	627
21.3.5.6	Defining the code section to be executed (Execution to line).....	628
21.3.5.7	Debugging: pausing threads (Pause).....	629
21.3.6	Variables view.....	629
21.3.6.1	Displaying and modifying variables.....	631
21.3.6.2	Advanced context help for variables.....	632
21.3.7	Monitoring processes.....	633
21.3.7.1	Adding new monitoring expressions.....	634
21.3.7.2	Deleting monitoring expressions.....	635
21.3.7.3	Evaluating monitoring expressions.....	635
21.3.8	Modifying source code.....	636
21.3.8.1	Impermissible modification of the source code.....	637
21.3.8.2	Permissible modification of the source code.....	637
22	Appendix.....	639
22.1	Risk management.....	639
23	KUKA Service.....	643
23.1	Requesting support.....	643
23.2	KUKA Customer Support.....	643
	Index	645

1 Introduction

1.1 Target group

This documentation is aimed at medical product manufacturers. The medical product manufacturer must ensure that the LBR Med is only handled by appropriately trained skilled personnel with the following knowledge and skills:

- Advanced knowledge of mechanical engineering
- Advanced knowledge of electrical engineering
- Knowledge of the robot controller system



This documentation may not be forwarded to the medical product manufacturer's customer as Instructions for Use.



For optimal use of KUKA products, we recommend the training courses offered by KUKA College. Information about the training program can be found at www.kuka.com or can be obtained directly from our subsidiaries.

1.2 LBR Med documentation

The documentation consists of the following parts:

- Instructions for Use of LBR Med:
Instructions for Use of LBR Med 7 R800 and LBR Med 14 R820
- Instructions for Use of KUKA Sunrise Cabinet Med:
Information about operating the robot controller
- Documentation for the smartPAD-2 (if used)
- Instructions for Use of KUKA Sunrise.OS Med:
Operating and Programming Instructions for System Integrators
- Instructions for options and accessories
- Spare parts in KUKA.Xpert

Each set of instructions is a separate document.



For integration of the LBR Med into a medical product, additionally supplied documents are available to the medical product manufacturer, which must be taken into consideration.

(>>> [22 "Appendix" Page 639](#))

1.3 Representation of warnings and notes

Safety

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



DANGER

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



WARNING

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

**CAUTION**

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

**NOTICE**

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

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

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

SAFETY INSTRUCTION

The following procedure must be followed exactly!

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

Notices

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



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

1.4 Trademarks

Java is a trademark of Sun Microsystems (Oracle Corporation).

Windows is a trademark of Microsoft Corporation.



EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

1.5 Terms used

Term	Description
AMF	Atomic Monitoring Function Smallest unit of a monitoring function
API	Application Programming Interface Interface for programming applications
CSV	Comma Separated Values Text format used for importing and exporting long texts and tables in databases.
ESM	Event-Driven Safety Monitoring Safety monitoring functions which are activated using defined events
EtherCAT	Ethernet for Control Automation Technology Ethernet-based field bus that is suitable for real-time requirements (Ethernet interface).

EVC	<p>Enhanced Velocity Control Option for limitation of the Cartesian robot velocity</p> <p>EVC automatically adapts the robot velocity so that safety-oriented and application-specific Cartesian velocity limits are adhered to.</p>
Exception	<p>Exception or exceptional situation</p> <p>An exception describes a procedure for forwarding information about certain program statuses, mainly error states, to other program levels for further processing.</p>
Frame	<p>3-dimensional coordinate system that is described by its position and orientation relative to a reference system</p> <p>Points in space can be easily defined using frames. Frames are often arranged hierarchically in a tree structure.</p>
FSoE	<p>FailSafe over EtherCAT Protocol for transferring safety-relevant data via EtherCAT. An FSoE master and an FSoE slave are used for this.</p>
GMS	<p>German acronym for joint torque sensor</p> <p>Sensitive robots of type LBR have a joint torque sensor in each axis. The torques on the output side in each axis are measured using these sensors.</p> <p>Note: This sensor is referred to as the axis torque sensor in the documentation.</p>
Javadoc	Javadoc is a documentation generated from specific Java comments.
JRE	<p>Java Runtime Environment Runtime environment of the Java programming language</p>
KLI	<p>KUKA Line Interface</p> <p>Ethernet interface of the robot controller for external communication (not real-time-capable).</p>
KUKA RoboticsAPI	<p>Java programming interface for KUKA robots</p> <p>KUKA RoboticsAPI is an object-oriented Java interface for controlling robots and peripheral devices.</p>
KUKA smartHMI	see “smartHMI”
KUKA smartPAD	see “smartPAD”
KUKA smartPAD-2	see “smartPAD”
KUKA Sunrise Cabinet	Control hardware for operating robots. Hereinafter called “robot controller” in these Instructions for Use.
KUKA Sunrise.OS	<p>KUKA Sunrise.Operating System</p> <p>System Software for robots which are operated with the robot controller KUKA Sunrise Cabinet</p>

LBR	<p>Lightweight robot</p> <p>Robot as part of a robotic component</p> <p>There are 2 robot variants available:</p> <ul style="list-style-type: none"> • LBR Med 7 R800 • LBR Med 14 R820 <p>Hereinafter called “robot” for both variants in these Instructions for Use.</p>
LBR Med	<p>A robotic component for integration into a medical device by a medical device manufacturer. The LBR Med includes all assemblies, such as the robot (robot arm and the associated electrical installations), robot controller and connecting cables.</p> <p>Hereinafter also called “robot system” in these Instructions for Use.</p>
Med	Identifier for KUKA products that are supplied to medical device manufacturers.
HRC	Human-robot collaboration
PEMS	Programmable electrical medical systems
PROFINET	Ethernet-based field bus (Ethernet interface)
PROFIsafe	PROFINET-based safety interface for connecting a safety PLC to the robot controller (PLC = master, robot controller = slave)
PSM	Permanent Safety Monitoring Safety monitoring functions which are permanently active
smartHMI	smart Human-Machine Interface User interface on the smartPAD
smartPAD	<p>Teach pendant</p> <p>The smartPAD is the teach pendant for the robot. It provides all the operator control and display functions required by the distributor and the system integrator (in the medical environment) for operating the robot during start-up, maintenance and diagnosis.</p> <p>2 models exist:</p> <ul style="list-style-type: none"> • smartPAD • smartPAD-2 <p>In turn, for each model there are variants, e.g. with different lengths of connecting cables.</p> <p>The designation “KUKA smartPAD” or “smartPAD” refers to both models unless an explicit distinction is made.</p>
PLC	Programmable Logic Controller

Sunrise project	A Sunrise project is a specialization of a Java project. All Sunrise-specific functions are only possible in Sunrise projects and not in conventional Java projects. These functions include, for example:
	<ul style="list-style-type: none"> • Creation of a safety configuration • Creation of an I/O configuration • Creation of robot and background applications • Configuration of the Automatic External interface
TCP	<p>Tool Center Point Working point of a tool and origin of the corresponding tool coordinate system</p>
Medical device manufacturer (distributor, customer)	The customer is the manufacturer of the medical device. In their capacity as medical device manufacturer/system integrator, they integrate the robotic component into their medical device and market the overall system as a medical device.
System integrator (plant integrator)	System integrators are people who safely integrate the LBR Med into a medical device and commission it.
End user	Customer of the medical device manufacturer (user, e.g. medical personnel)
VxWorks	Real-time operating system

1.6 Licenses

KUKA Sunrise.OS uses open-source software. The license terms can be found in the **licenses** folder in the installation directory of KUKA Sunrise.Workbench.



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

2 Product description

2.1 Overview of the robot system

The robot system (>>> [Fig. 2-1](#)) includes all assemblies, such as the robot (robot arm and the associated electrical installations), robot controller and connecting cables.

The robot system consists of the following components:

- Robot (with media flange Inside electrical Med)
- KUKA Sunrise Cabinet Med robot controller
- KUKA smartPAD teach pendant (optional)
- Connecting cables
- KUKA Sunrise.OS Med software
- Options, accessories



Detailed information about the components of the robot system can be found in the supplied documentation.

(>>> [22 "Appendix" Page 639](#))

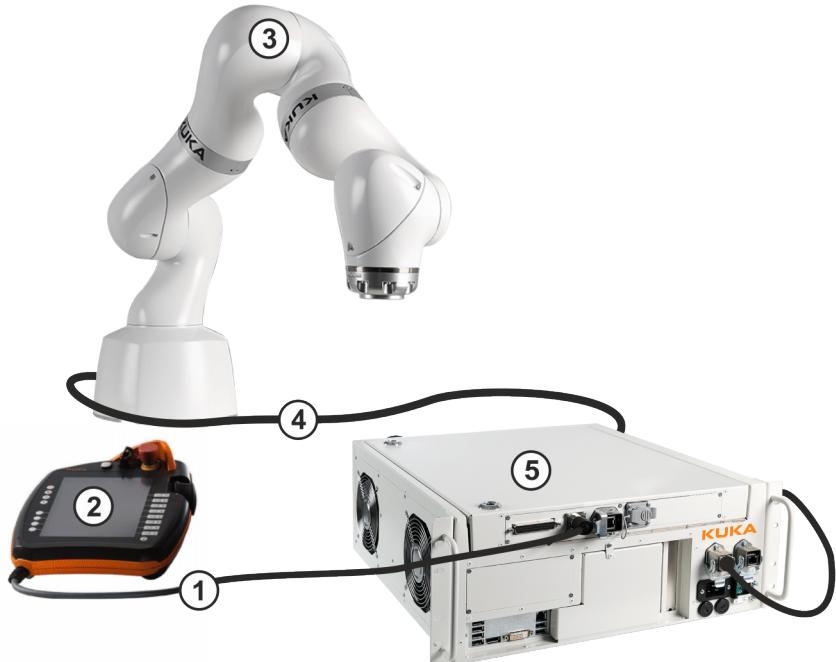


Fig. 2-1: Overview of robot system (LBR Med)

- 1 Connecting cable to KUKA smartPAD
- 2 KUKA smartPAD teach pendant
- 3 Robot
- 4 Connecting cable to KUKA Sunrise Cabinet Med robot controller
- 5 KUKA Sunrise Cabinet Med robot controller

2.2 Overview of the software components

The following software components are used:

- KUKA Sunrise.OS Med 1.15
 - The software consists of the following components:
 - KUKA Sunrise.OS 1.15
 - KUKA Sunrise.Framework Med 1.15
 - KUKA Sunrise.Workbench 1.15
 - WorkVisual 5.0

2.3 Overview of KUKA Sunrise.OS

Description

KUKA Sunrise.OS Med is a System Software package for robots in which programming and operator control tasks are strictly separated from one another.

- Robot applications are programmed with KUKA Sunrise.Workbench.
- The robot is operated by the medical device manufacturer using the KUKA smartPAD teach pendant.
- A station consists of a robot controller, a robot and other devices.
- A station may carry out multiple applications (tasks).

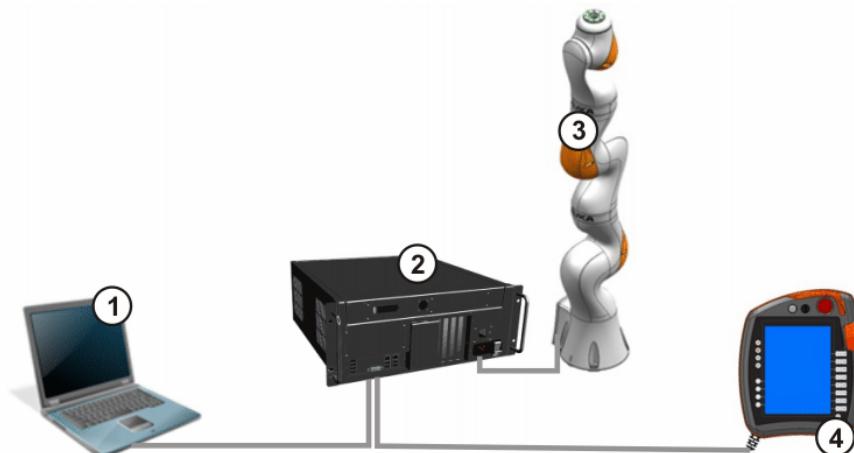


Fig. 2-2: Separation of operator control and programming

- 1 Development computer with KUKA Sunrise.Workbench (connection via the KLI of the robot controller)
- 2 KUKA Sunrise Cabinet Med robot controller
- 3 Robot
- 4 KUKA smartPAD teach pendant



The development computer is not included in the scope of supply of the robot.

Division of tasks

KUKA Sunrise.Workbench is the tool for start-up and for the development of robot applications. WorkVisual is used for bus configuration and bus mapping.

The smartPAD is only required by the system integrator for tasks during start-up and recommissioning (e.g. after maintenance) that cannot be performed with KUKA Sunrise.Workbench for practical or safety reasons. The smartPAD is used, for example, for calibrating tools and teaching points.

After start-up and application development, the operator can carry out simple maintenance work and operating tasks using the smartPAD. The operator cannot change the station and safety configuration or the programming.

Overview

Task	WorkVisual	Workbench	smartPAD
Station configuration		✓	
Software installation		✓	
Bus configuration/diagnosis	✓		
Bus mapping	✓		
Configuring safety settings		✓	
Activating the safety configuration			✓
Programming		✓	
Remote debugging		✓	
Managing/editing process data		✓	
Creating frames		✓	✓
Teaching frames			✓
Operating mode selection			✓
Jogging			✓
Mastering			✓
Calibration			✓
Load data determination			✓
Setting outputs		✓	✓
Polling inputs/outputs			✓
Starting/stopping robot applications			✓
Starting/stopping background applications			✓
Creating a diagnosis package		✓	✓

2.4 Overview of KUKA Sunrise.Workbench

KUKA Sunrise.Workbench is the development environment for the LBR Med/medical product. It offers the following functionalities for start-up and application development:

Start-up

- Installing the System Software
- Configuring the robot
- Editing the safety configuration
- Creating the I/O configuration

- Transferring the project to the robot controller
- Implementation of user-specific applications of the medical product manufacturer

Application development

- Programming robot applications in Java
- Managing projects and programs
- Editing and managing runtime data
- Project synchronization
- Remote debugging (fault location and elimination)
 - Setting break points
 - Program execution in single-step operation (stop after each program line)
 - Displaying and modifying application variables during program execution
 - Modifying a program during execution

2.5 Intended use of the System Software

Use

The System Software is intended exclusively for operation of an LBR Med in conjunction with KUKA Sunrise Cabinet Med.

Each version of the System Software must 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:

- Operating axes that are not KUKA axes
- Operation of the System Software not in accordance with the specified system requirements
- Use of any debugger other than that provided by Sunrise.Workbench
- Use for industrial applications for which specific product requirements/standards exist (e.g. industrial systems)
- Direct use of the underlying operating systems and their interfaces (e.g. login or use of the Windows desktop).

3 Safety

3.1 General

3.1.1 Disclaimer

The robot system described in this document is a robotic component for integration into a medical device (hereinafter called "LBR Med"):

The LBR Med includes:

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

The LBR Med is integrated into a stationary, permanently installed or mobile medical product by a medical product manufacturer.

The LBR Med is intended to carry out the following tasks within the medical device:

- Guidance of tools including navigation
- Repetition of motions

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

The LBR Med may only be used in perfect technical condition in accordance with its intended use and only by safety-conscious persons who are fully aware of the risks involved in its operation. Use is subject to compliance with this document and with the documentation provided with the robot upon delivery (Important information on LBR Med). Any functional disorders, especially those affecting safety, must be rectified immediately.

The LBR Med meets the requirements of the standards named in the CB Test Certificate. All requirements deviating from the CB Test Certificate are not met.

All applications deviating from the CB Test Certificate have not been considered, tested and checked. The medical product manufacturer must evaluate these applications in its risk management process and take the necessary measures for the medical product / overall system.

Examples of such applications include:

- Integration into a portable or hand-held medical device
- Integration of the LBR Med into medical devices which are intended for sale to the general public
- Operation in domestic environments
- Operation in the vicinity of emergency response operations
- Operation in the vicinity of strong magnetic fields
- Operation in combination with the use of defibrillators (the robot is not defibrillator-safe)
- Unsupervised operation of the LBR Med or the medical device (e.g. by the medical device manufacturer or end user)
- Operation of the medical device by non-medical personnel, patients or third parties.

- Operation of the robot as an applied part
- Connection of type BF or CF applied parts according to IEC 60601-1 directly to the robot without suitable intermediate insulation
- Power supply to an applied part via the data cables of the integrated energy supply system (e.g. Power over Ethernet)
- Operation by persons without the relevant training

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 LBR Med will not cause personal injuries or material damage.

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

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

3.1.1.1 CB Test Certificate and CB Test Report

The LBR Med is tested according to the CB Scheme and delivered with a CB Test Certificate.

Through the CB Test Certificate, it is declared that the LBR Med has been produced in accordance with the standards named in the CB Test Certificate.

Only the software components named in the CB Test Report in the specified versions are compatible with one another and may be jointly installed. If software components deviating from the CB Test Report are installed, the CB Test Report is no longer valid.

3.1.1.2 EC declaration of conformity and declaration of incorporation

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

- The LBR Med is integrated into a medical device.
- The medical device complies with Medical Device Directive/Regulation. 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 in accordance with the EU Directive/Regulation for the medical device. The EC declaration of conformity forms the basis for the CE mark for the medical device. The LBR Med must always be operated in accordance with the applicable national laws, regulations and standards.

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

Declaration of incorporation

The partly completed machinery is supplied with a declaration of incorporation in accordance with Annex II B of the Machinery Directive 2006/42/EC. The assembly instructions and a list of essential require-

ments 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 a medical device, or has been assembled with other parts to form a medical device, and this medical device complies with the terms of the Medical Device Directive/Regulation, and the EC declaration of conformity is present.

3.1.2 Terms in the “Safety” chapter

Term	Description
Axis range	Range within which the axis may move. The axis range must be defined for each axis.
Stopping distance	Stopping distance = reaction distance + braking distance The stopping distance is part of the danger zone.
Workspace	Area within which the robot may move. The workspace is derived from the individual axis ranges.
AUT	Automatic mode Operating mode for program execution. The robot moves at the programmed velocity.
User	The user of the medical device can be the management, employer or delegated person responsible for use of the medical device.
ESD	Electrostatic sensitive devices
Service life	The service life of a safety-relevant component begins at the time of delivery of the component to the customer. The service life is not affected by whether the safety-relevant component is used in a robot controller or elsewhere or not, as safety-relevant components are also subject to aging during storage.
Danger zone	The danger zone consists of the workspace and the stopping distances.
CRR	Controlled robot retraction CRR is an operating mode to which the system can be switched when the robot is stopped by the safety controller for one of the following reasons: <ul style="list-style-type: none"> • Robot violates an axis-specific or Cartesian monitoring space. • Orientation of a safety-oriented tool is outside the monitored range. • Robot violates a force or torque monitoring function. • A position sensor is not mastered or referenced. • A joint torque sensor is not referenced. Once the operating mode has been switched to CRR, the robot can be moved again. Note: This operating mode can only be selected in conjunction with the smartPAD.

KUKA smartPAD	see "smartPAD"
KUKA smartPAD-2	see "smartPAD"
Safety zone	The safety zone is situated outside the danger zone.
Safety stop	<p>The safety stop is triggered by the safety controller, interrupts the work procedure and causes all robot motions to come to a standstill. The program data are retained in the case of a safety stop and the program can be resumed from the point of interruption.</p> <p>The safety stop can be executed as a Stop category 0, Stop category 1 or Stop category 1 (path-maintaining).</p> <p>Note: In this document, a safety stop of Stop category 0 is referred to as safety stop 0, a safety stop of Stop category 1 as safety stop 1 and a safety stop of Stop category 1 (path-maintaining) as safety stop 1 (path-maintaining).</p>
smartPAD	<p>Teach pendant</p> <p>The smartPAD is the teach pendant for the robot. It provides all the operator control and display functions required by the distributor and the system integrator (in the medical environment) for operating the robot during start-up, maintenance and diagnosis.</p> <p>2 models exist:</p> <ul style="list-style-type: none"> • smartPAD • smartPAD-2 <p>In turn, for each model there are variants, e.g. with different lengths of connecting cables.</p> <p>The designation "KUKA smartPAD" or "smartPAD" refers to both models unless an explicit distinction is made.</p>
Stop category 0	The energy supply to the safety-oriented drives is immediately disconnected and the brakes are applied.
Stop category 1	<p>The robot is braked and deviates from the programmed path. The robot is brought to a standstill with the drives. As soon as an axis is at a standstill, the drive is switched off and the brake is applied.</p> <p>The internal electronic drive system of the robot performs safety-oriented monitoring of the braking process. Stop category 0 is executed in the event of a fault.</p> <p>Note: Stop category 1 is currently only supported by the LBR Med. For other manipulators, Stop category 0 is executed.</p>
Stop category 1 (path-maintaining)	<p>The robot is braked and stays on the programmed path. At standstill, the drives are deactivated and the brakes are applied.</p> <p>If Stop category 1 (path-maintaining) is triggered by the safety controller, the safety controller monitors the braking process. The brakes are applied and the drives are switched off after 1 s at the latest. Stop category 1 is executed in the event of a fault.</p>
System integrator (plant integrator)	System integrators are people who safely integrate the LBR Med into a medical device and commission it.

T1	Test mode, Manual Reduced Velocity (<= 250 mm/s)
	Note: With manual guidance in T1, the velocity is not reduced, but rather limited through a safety-oriented velocity monitoring in accordance with the safety configuration.
	Note: This operating mode can only be selected in conjunction with the smartPAD.

T2	Test mode, Manual High Velocity (> 250 mm/s permissible)
	Note: This operating mode can only be selected in conjunction with the smartPAD.

3.2 Groups of persons



The operating and programming instructions for system integrators contain information for the medical device manufacturer and are not intended for the end user (e.g. medical personnel).

The following persons or groups of persons are defined for the LBR Med:

- System integrator / medical device manufacturer
- User
- Personnel



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

System integrator

The LBR Med must be safely integrated into the medical product by the system integrator.

The following list gives an indication as to the tasks for which the system integrator is responsible:

- Installing the LBR Med
- Connecting the LBR Med
- Conducting risk management for the medical product
- Use of the required safety equipment and safeguards
- Issuing the EC declaration of conformity
- Attaching the CE mark
- Creating the Instructions for Use for the medical product and the corresponding components for the overall system for the user and end user

User

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

- The user must comply with his monitoring obligations.
- The user must carry out briefing at defined intervals.
- The user must comply with the regulations relating to personal protective equipment (PPE).



The user must ensure that the robot controller is inaccessible to unauthorized personnel. The user is responsible for the storage and issuing of keys for the system (e.g. robot controller, smartPAD).
The user must issue keys in accordance with the applicable laws, regulations and standards.

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 Instructions for Use for the relevant component of the LBR Med and only by personnel specially trained for this purpose.

Operators

The operator must meet the following preconditions:

- The operator must be trained for the work to be carried out.
- Work on the LBR Med 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 robot may only be carried out by KUKA Deutschland GmbH.

3.3

Workspace, safety zone and danger zone

Workspaces are to be restricted to the necessary minimum size in order to prevent danger to persons or the risk of material damage. Safe axis range limitations required for personnel protection are configurable.



Further information about the configuration of safely monitored axis limits is contained in the "Safety configuration" chapter of the operating and programming instructions for system integrators. (>>> [13 "Safety configuration" Page 237](#))

The danger zone consists of the workspace and the stopping distances of the robot. In the event of a stop, the robot is braked and comes to a stop within the danger zone. The safety zone is the area outside the danger zone.

The danger zone must be protected by means of physical safeguards, e.g. by light barriers, light curtains or safety fences. If there are no physical safeguards present, the requirements for collaborative operation must be examined and met where applicable in accordance with EN ISO 10218.

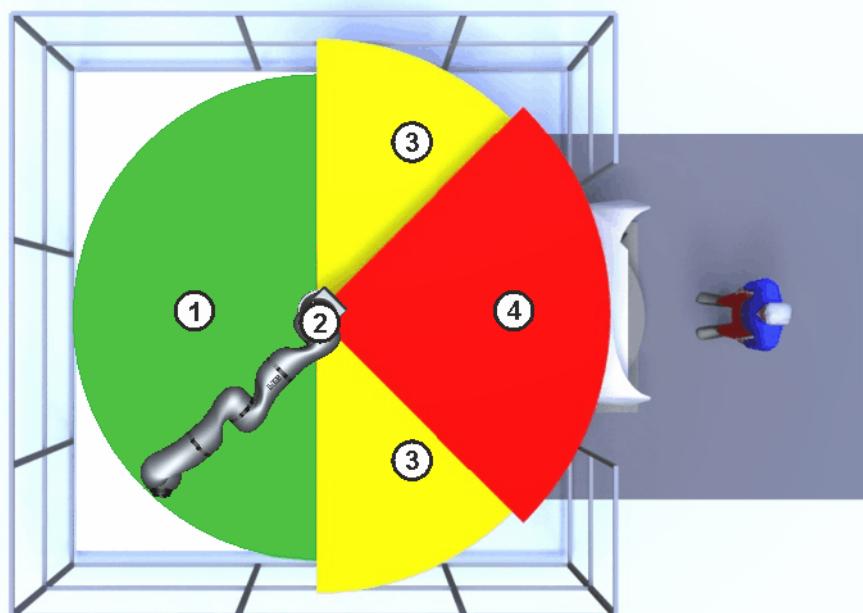


Fig. 3-1: Example: axis range A1

- | | |
|------------------|------------------------|
| 1 Workspace | 3 Stopping distance |
| 2 Manipulator | 4 Safety zone |

3.4 Triggers for safety-oriented stop reactions

Stop reactions are triggered in response to operator actions or as a reaction to monitoring functions and errors. The following tables show the different stop reactions according to the operating mode that has been set.

Overview

In Sunrise.OS a distinction is made between the following triggers:

- Permanently defined triggers
Permanently defined triggers for stop reactions and the associated stop category are preset by the system and cannot be changed. However, it is possible for the implemented stop reaction to be stepped up in the user-specific safety configuration.
- User-specific triggers
In addition to the permanently defined triggers, the user can also configure other triggers for stop reactions including the associated stop category.



Further information about configuring the safety functions is contained in the "Safety configuration" chapter of the operating and programming instructions of the system software for system integrators. (>>> [13 "Safety configuration" Page 237](#))

Permanently defined triggers

The following triggers for stop reactions are permanently defined:

Trigger	T1, T2, CRR	AUT
Operating mode changed during operation	Safety stop 1 (path-maintaining)	
Enabling switch released	Safety stop 1 (path-maintaining)	-
Enabling switch pressed fully down (panic position)	Safety stop 1 (path-maintaining)	-
Local E-STOP pressed	Safety stop 1 (path-maintaining)	
Error in safety controller	Safety stop 1	

User-specific triggers

When creating a new Sunrise project, the system automatically generates a project-specific safety configuration. This contains the following user-specific stop reaction triggers preconfigured by KUKA (in addition to the permanently defined triggers):

Trigger	T1, CRR	T2, AUT
Safety gate opened (operator safety)	-	Safety stop 1 (path-maintaining)
External E-STOP pressed	Safety stop 1 (path-maintaining)	
External safety stop	Safety stop 1 (path-maintaining)	



This default safety configuration is valid for the system software without additionally installed option packages or catalog elements. If additional option packages or catalog elements have been installed, the default safety configuration may be modified.

Triggers for manual guidance

If an enabling device is configured for manual guidance, the following additional triggers for stop reactions are permanently defined:

Trigger	T1, CRR	T2, AUT
Manual guidance enabling switch released	Safety stop 1 (path-maintaining)	-
Manual guidance enabling switch pressed fully down (panic position)	Safety stop 1 (path-maintaining)	-
Maximum permissible velocity exceeded while manual guidance enabling signal is set	Safety stop 1 (path-maintaining)	

A maximum permissible velocity of 250 mm/s is preconfigured for manual guidance. The maximum permissible velocity can be adapted.

The medical product manufacturer must evaluate the maximum permissible velocity in its risk management process and take the appropriate measures.

3.5 Safety functions

Safety functions are distinguished according to the safety requirements that they fulfill:

- Safety-oriented functions for the protection of personnel

The safety-oriented functions of the LBR Med meet the following safety requirements:

- **Category 3 and Performance Level d** in accordance with EN ISO 13849-1
- **SIL 2** according to EN 62061

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

- All safety-relevant mechanical and electromechanical components of the LBR Med are tested for correct functioning during start-up and at least once every 12 months, unless otherwise determined in accordance with a workplace risk assessment. These include:

- Local EMERGENCY STOP device on the teach pendant
- Enabling device on the teach pendant
- Enabling device on the hand guiding device (if present)
- External enabling devices (if present)
- Mode selector switch on the smartPAD (if used as teach pendant)
- Safety-oriented outputs of the discrete safety interface



Details about safety parameters (e.g. PFH, SIL, Performance Level) are also available as a SISTEMA library. The library can be downloaded from the KUKA website.

- Non-safety-oriented functions

The non-safety-oriented functions of the LBR Med do not meet specific safety requirements.



DANGER

Risk of fatal injury due to non-operational safety functions or external safeguards

In the absence of operational safety functions or safeguards, the robot can cause death, severe injuries or damage to property.

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



Integrate robot into safety system of medical device

During the integration of the LBR Med, the safety functions of the medical device must also be planned and designed. Death, severe injuries or damage to property may otherwise result.

- The LBR Med must be integrated into the safety system of the medical device.

3.5.1 Safety-oriented functions

The following safety-oriented functions are present and permanently defined in the LBR Med:

- EMERGENCY STOP device
- Enabling device

The following safety-oriented functions are preconfigured and can be integrated via the safety interface of the robot controller:

- Operator safety (= connection for the monitoring of physical safeguards)

- External EMERGENCY STOP device
- External safety stop 1 (path-maintaining)

Other safety-oriented functions may be configured, e.g.:

- External enabling device
- External safe operational stop
- Axis-specific workspace monitoring
- Cartesian workspace monitoring
- Cartesian protected space monitoring
- Velocity monitoring
- Standstill monitoring
- Axis torque monitoring
- Collision detection



Further information about configuring the safety functions is contained in the "Safety configuration" chapter of the operating and programming instructions of the system software for system integrators. (>>> [13 "Safety configuration" Page 237](#))

The preconfigured safety functions are described in the following sections on safety.

3.5.1.1 EMERGENCY STOP device

As standard, the EMERGENCY STOP device for the manipulator is the EMERGENCY STOP device on the smartPAD. The device must be pressed in the event of a hazardous situation or emergency.

Reaction of the LBR Med if the EMERGENCY STOP device is pressed:

- The robot stops with a safety stop 1 (path-maintaining).

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



WARNING

Danger to life and limb due to tools and equipment without EMERGENCY STOP

If tools and other equipment connected to the robot are not integrated into the EMERGENCY STOP circuit, this can result in death, severe injuries or damage to property.

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

If a holder is used for the teach pendant and conceals the EMERGENCY STOP device on the teach pendant, an external EMERGENCY STOP device must be installed that is accessible at all times.

If the smartPAD is unplugged, an external EMERGENCY STOP device must be installed that is accessible at all times.

(>>> [3.5.1.4 "External EMERGENCY STOP device" Page 40](#))

3.5.1.2 Enabling device

The enabling devices of the robot are the enabling switches on the smartPAD teach pendant as standard.

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

The enabling switches have 3 positions:

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

In operating modes T1, T2 and CRR, the robot can only be moved if one of the enabling switches is held in the center position.

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

In operating modes T1, T2 and CRR, the manipulator can be stopped in the following ways:

- Press at least one enabling switch down fully.
Fully pressing an enabling switch triggers a safety stop 1 (path-maintaining).
- Or release all enabling switches.

Releasing all (!) enabling switches held in the center position triggers a safety stop 1 (path-maintaining).



WARNING

Danger to life and limb due to lack of reaction when an enabling switch is released

Releasing one of multiple enabling switches held in the center position does not trigger a stop reaction.

If multiple switches are held in the center position, the robot controller cannot distinguish whether one of them was intentionally released or if it was unintentionally released as the result of an accident.

- Create awareness for the hazard.

If an enabling switch malfunctions (e.g. jams in the center position), the LBR Med can be stopped using the following methods:

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



WARNING

Danger to life and limb due to manipulation of enabling switches

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

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

3.5.1.3 “Operator safety” signal

The “operator safety” signal is used for monitoring physical safeguards, e.g. safety gates. In the default configuration, T2 and automatic operation are not possible without this signal. Alternatively, the requirements for collaborative operation must be examined and met where applicable in accordance with EN ISO 10218.

Reaction of the robot in the event of a loss of signal during T2 or automatic operation (default configuration):

- The robot stops with a safety stop 1 (path-maintaining).

By default, operator safety is not active in the modes T1 (Manual Reduced Velocity) and CRR, i.e. the signal is not evaluated.



WARNING

Following a loss of signal, automatic operation must not be resumed merely by closing the safeguard; the signal for operator safety must first be set by an additional device, e.g. by an acknowledge button. It is the responsibility of the system integrator to ensure this. This 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.

- This additional device must be designed in such a way that an actual check of the danger zone can be carried out first. Devices that do not allow this (e.g. because they are automatically triggered by closure of the safeguard) are not permitted.
- Failure to observe this may result in death to persons, severe injuries or considerable damage to property.

3.5.1.4 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.

Reaction of the LBR Med if the external EMERGENCY STOP device is pressed (default configuration):

- The robot stops with a safety stop 1 (path-maintaining).

External EMERGENCY STOP devices are connected via the safety interface of the robot controller. External EMERGENCY STOP devices are not included in the scope of supply of the LBR Med.

3.5.1.5 External safety stop 1 (path-maintaining)

The external safety stop 1 (path-maintaining) can be triggered via an input on the safety interface (default configuration). The state is maintained as long as the external signal is FALSE. If the external signal is TRUE, the robot can be moved again. No acknowledgement is required.

3.5.1.6 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 LBR Med.

Multiple external enabling devices can be connected via the safety interface of the robot controller. External enabling devices are not included in the scope of supply of the LBR Med.

An external enabling device can be used for manual guidance of the robot. When enabling is active, the robot may only be moved at reduced velocity.

For manual guidance, safety-oriented velocity monitoring with a maximum permissible velocity of 250 mm/s is preconfigured. The maximum permissible velocity can be adapted.

The value for the maximum permissible velocity must be determined as part of a risk management process.

3.5.1.7 External safe operational stop

The safe operational stop is a standstill monitoring function. It does not stop the robot motion, but monitors whether the robot axes are stationary.

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

3.5.2 Non-safety-oriented functions

3.5.2.1 Mode selection

The LBR Med can be operated in the following modes:

- Manual Reduced Velocity (T1)
- Manual High Velocity (T2)
- Automatic (AUT)
- Controlled robot retraction (CRR)

Operating mode	Use	Velocities
T1	Programming, teaching and testing of programs.	<ul style="list-style-type: none"> • Program verification: reduced programmed velocity, maximum 250 mm/s • Jog mode: jog velocity, maximum 250 mm/s • Manual guidance: no limitation of the velocity, but safety-oriented velocity monitoring in accordance with the safety configuration
T2	Testing of programs	<ul style="list-style-type: none"> • Program verification: programmed velocity • Jog mode: not possible
AUT	Automatic execution of programs For robots with and without a higher-level controller	<ul style="list-style-type: none"> • Program mode: programmed velocity • Jog mode: not possible

Operating mode	Use	Velocities
CRR	<p>CRR is an operating mode to which the system can be switched when the robot is stopped by the safety controller for one of the following reasons:</p> <ul style="list-style-type: none"> • Robot violates an axis-specific or Cartesian monitoring space. • Orientation of a safety-oriented tool is outside the monitored range. • Robot violates a force or torque monitoring function. • A position sensor is not mastered or referenced. • An axis torque sensor is not referenced. <p>Once the operating mode has been switched to CRR, the robot can be moved again.</p>	<ul style="list-style-type: none"> • Program verification: reduced programmed velocity, maximum 250 mm/s • Jog mode: jog velocity, maximum 250 mm/s • Manual guidance: no limitation of the velocity, but safety-oriented velocity monitoring in accordance with the safety configuration

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 for LBR Med:

- With key

It is only possible to change operating mode if the key is inserted.

3.5.2.2 Velocity monitoring in T1

The reduced velocity in T1 does not constitute a safety-rated reduced speed in the standard safety configuration, i.e. the maximum permissible velocity of 250 mm/s in T1 is not subjected to safety-oriented monitoring.

If the application requires safety-oriented velocity monitoring in T1, this can be added in the safety configuration. The safety option KUKA Sunrise.SafeOperation provides the monitoring function *Cartesian velocity monitoring* for this purpose.



Further information about configuring safety-oriented velocity monitoring for T1 is contained in the “Safety configuration” chapter of the operating and programming instructions of the system software for system integrators. (>>> [13 “Safety configuration” Page 237](#))

3.5.2.3 Software limit switches

The axis ranges of all axes on the robot are limited by means of non-safety-oriented software limit switches. These software limit switches only serve as protection for the robot and are preset in such a way that the robot is stopped under servo control if the axis limit is exceeded, thereby preventing damage to the mechanical system.

3.6 Service phases of the robot system

The following service phases have been defined for the robot system:

- **Transportation**
- **Start-up and recommissioning**
- **Normal operation**
- **Malfunction**
- **Cleaning**
- **Maintenance**
- **Repair**
- **Decommissioning**
- **Storage**
- **Disposal**

Full responsibility for the service phases indicated lies with the medical product manufacturer.

The medical product manufacturer must provide the end customer the necessary information for operation of the medical product (overall system) during the service phases of normal operation, malfunction, cleaning and maintenance.

3.7 Additional protective equipment

3.7.1 Jog mode

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

- Releasing the enabling switch on the smartPAD triggers a safety stop 1 (path-maintaining).
- Pressing the enabling switch on the smartPAD fully down triggers a safety stop 1 (path-maintaining).
- Releasing the Start key triggers a stop of Stop category 1 (path-maintaining).

3.7.2 Labeling on the LBR Med

All plates, labels, symbols and marks constitute safety-relevant parts of the LBR Med.



CAUTION

If the plates and labels are removed for integration into a medical product, the medical product manufacturer must evaluate this in its risk management process and take the necessary measures.

Labeling on the LBR Med consists of:

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

- Rating plates



Further information is contained in the technical data of the Instructions for Use for the components of the LBR Med.

3.7.3 External safeguards

The medical product manufacturer must evaluate the selection of external safeguards in its risk management process and take the necessary measures for the medical product / overall system.

Access of persons to the danger zone of the LBR Med must be prevented by means of safeguards. Alternatively, the requirements for collaborative operation must be examined and met where applicable in accordance with EN ISO 10218. It is the responsibility of the system integrator to ensure this.

Physical safeguards must meet the following requirements:

- They must be examined according to requirements and meet EN ISO 14120 where applicable.
- 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.
- The prescribed minimum clearance from the danger zone is maintained.

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 configured operator safety inputs of the robot controller.
- Switching devices, switches and the type of switching conform to the requirements of Performance Level d and category 3 according to EN ISO 13849-1.
- Depending on the hazard situation: the safety gate is additionally safeguarded by means of a locking mechanism that only allows the gate to be opened if the robot is safely at a standstill.
- The device for setting the signal for safety, e.g. 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.

3.8 Safety measures

3.8.1 General safety measures

The LBR Med 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 robot even after the robot controller has been switched off and locked. Incorrect installation (e.g. overload) or mechanical defects (e.g. brake defect) can cause the robot to sag. If work is to be carried out on a switched-off LBR Med, the robot must first be moved into a position in which it is unable to move on its own, whether the payload is mounted or not. If this is not possible, the robot must be secured by appropriate means.



DANGER

Risk of fatal injury due to non-operational safety functions or external safeguards

In the absence of operational safety functions or safeguards, the robot can cause death, severe injuries or damage to property.

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



DANGER

Danger to life and limb of persons under the robot arm

Standing underneath the robot arm can lead to death, serious injuries or damage to property. Especially if objects are being moved that can become detached.

- The system integrator must evaluate the risk from standing under the robot arm in his risk management process and implement the appropriate measures.

HRC

In the case of collaborative operation (HRC), the system must be equipped with a visual display indicating when the robot is in collaborative operation.

Implants



DANGER

Danger to life due to malfunction of implants caused by motors and magnets

Electric motors and magnets generate electric and magnetic fields. The fields can cause malfunctions in implants, e.g. pacemakers.

- The integrator / medical device manufacturer must evaluate this through its risk management and take the appropriate measures.

smartPAD

The medical device manufacturer must ensure that the LBR Med is only operated with the smartPAD by authorized persons.



The smartPAD may only be used as intended by the system integrator for start-up and recommissioning (e.g. after maintenance).

The smartPAD is not intended for operation of the medical device by the end user (e.g. medical personnel).

If more than one smartPAD is used in a medical device, it must be ensured that each smartPAD is unambiguously assigned to the corresponding LBR Med. It must be ensured that 2 smartPADs are not interchanged.

The medical device manufacturer must configure the smartPAD as unpluggable.



WARNING

Risk of fatal injury due to non-operational EMERGENCY STOP device

If the smartPAD is disconnected, the medical device can no longer be switched off by means of the EMERGENCY STOP device on the smartPAD.

Death, injuries or damage to property may otherwise result.

- If the smartPAD is configured as unpluggable, at least one external EMERGENCY STOP device must be installed that is accessible at all times.

Modifications

After modifications to the LBR Med, 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 LBR Med, existing programs must always be tested first in Manual Reduced Velocity mode (T1). This applies to all components of the LBR Med and includes modifications to the software and configuration settings.

The robot may not be connected and disconnected when the robot controller is running.

Faults

The following procedure must be followed in the case of faults on the LBR Med:

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

3.8.2 IT security

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

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



Take measures to ensure IT security

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

- Technology, organization, personnel, infrastructure

KUKA urgently recommends that users implement an information security management system for their products which designs, coordinates and monitors the tasks related to information security.

Sources for information about IT security for companies include:

- Independent consulting firms
- National cybersecurity authorities

National authorities often make their recommendations available on the Internet.

3.8.3 Transportation

Robot

The prescribed transport position of the robot must be observed. Transportation must be carried out in accordance with the Instructions for Use for the LBR Med.

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 Instructions for Use for the robot controller.

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

Contamination

Seemingly or actually contaminated products constitute a health hazard to persons who come into contact with the returned goods due to infectious biomaterials or hazardous substances.



A template for declaration of the hygiene status can be found in the supplied documentation. The declaration must be filled out and attached to the goods being returned.



Further information can be found on the website of BVMed - Bundesverband Medizintechnologie e. V.

3.8.4 Start-up and recommissioning

Before starting up the LBR Med and the medical device for the first time, a check must be carried out to ensure that the LBR Med and the medical device 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.



Prior to start-up, the passwords for the user groups must be modified by the administrator, transferred to the robot controller in an installation procedure and activated. The passwords must only be communicated to authorized personnel. (>>> [9.4.2 "Changing and activating the password" Page 186](#))



Do not impair safety functions

Additional components (e.g. cables and hoses) not supplied by KUKA may be integrated into the LBR Med or the medical device. The medical device manufacturer is responsible for ensuring that these components do not impair or disable safety functions. Non-compliance can result in death, serious injury or damage to property.

- Additional components must not impair or disable safety functions.

NOTICE

Damage to property due to condensation

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

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

Function test

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

General test:

It must be ensured that:

- The LBR Med is correctly installed and fastened in accordance with the specifications in the documentation.
- There are no foreign bodies, or defective or loose parts on the LBR Med.
- All required safety equipment is correctly installed and operational.
- The power supply ratings of the LBR Med correspond to the local supply voltage and mains type.
- The equipotential bonding cable is 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 all the safety-oriented functions to ensure that they are working correctly.

Test of the safety-relevant mechanical and electromechanical components:

The following tests must be performed prior to start-up and at least once every 12 months unless otherwise determined in accordance with a workplace risk assessment:

- Function of all connected EMERGENCY STOP devices

Press the EMERGENCY STOP device. A message must be displayed on the teach pendant indicating that the EMERGENCY STOP has been actuated. At the same time, no error message may be displayed about the EMERGENCY STOP device.

- Function of the enabling switches of all connected enabling devices
Move the robot in Test mode and release the enabling switch. The robot motion must be stopped. At the same time, no error message may be displayed on the teach pendant about the enabling device.
The test must always be carried out for all enabling switches of a connected enabling device.
If the state of the enabling device is configured at an output, the test can also be performed via the output.
- Panic function of the enabling switches of all connected enabling devices
Move the robot in test mode, press the enabling switch down and hold in the panic position for 3 seconds. The robot motion must be stopped. At the same time, no error message may be displayed on the teach pendant about the enabling device.
The test must always be carried out for all enabling switches of a connected enabling device.
If the state of the enabling device is configured at an output, the test can also be performed via the output.
- Function of the mode selector switch on the smartPAD (if used as teach pendant)
Turn the mode selector switch to the right and then back again. There must be no error message displayed on the smartPAD.
- Switch-off capability of the safety-oriented outputs
Switch robot controller off and then on again. After it is switched on, no error message relating to a safety-oriented output may be displayed on the teach pendant.



In the case of incomplete start-up of the medical device, additional substitute measures for minimizing risk must be taken and documented, e.g. installing a safety fence, attaching a warning sign, locking the main switch. Start-up is incomplete, for example, if not all safety functions have yet been implemented, or if a function test of the safety functions has not yet been carried out.

Test of the functional capability of the brakes:

For the LBR Med, a brake test is available which must be used to check whether the brake of each axis applies sufficient braking torque.

The brake test ensures that any impairment of the braking function is detected, e.g. due to wear, overheating, fouling or damage, thereby eliminating avoidable risks.

The brake test cannot be switched off by the system integrator. The system forces regular brake tests and stops the system if these are not carried out.

- The brake test must be carried out for each axis during start-up and recommissioning of the LBR Med.

Inspection before start-up:

- Carry out a visual inspection of the robot and robot controller to check for damage.
- Check the fastening screws with the torque wrench.
- Carry out a visual inspection of the connecting cables and connectors to check for damage. Shake gently by hand. Secure any loose cables and connectors.

3.8.5 Manual mode

General

Manual mode is the mode for setup work. Setup work comprises all the tasks that have to be carried by the system integrator on the LBR Med to enable automatic operation. Setup work includes:

- Jog mode
- Teaching
- 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 robot and its tooling must never touch or project beyond the safety fence.
- Workpieces, tooling and other objects must not become jammed as a result of the 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 LBR Med.
- Eye-contact between all persons must be possible at all times.
- The system integrator must be so positioned that he can see into the danger area and get out of harm's way.
- Unexpected motions of the manipulator cannot be ruled out, e.g. in the event of a fault. For this reason, an appropriate clearance must be maintained between persons and the manipulator (including tool). Guide value: 50 cm.

The minimum clearance may vary depending on local circumstances, the motion program and other factors. In his risk assessment, the medical device manufacturer must assess the minimum clearance that is to apply for the specific application and take appropriate measures.

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 is not permissible in this operating mode.
- Before commencing the test, the system integrator must ensure that the enabling devices are operational.
- The system integrator must be positioned outside the danger zone.

- There must be no-one present inside the safeguarded area. It is the responsibility of the system integrator to ensure this.

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, or the requirements for collaborative operation have been examined and met where applicable in accordance with EN ISO 10218.
- The defined working procedures are adhered to.

If the robot 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



No maintenance and repair work may be carried out during operation.

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 taken when working on the LBR Med:

- 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 LBR Med 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, additional safety measures must be defined to ensure the safe protection of personnel.
- If it is necessary to carry out work with the robot controller switched on, this may only be done in operating mode T1.
- Label the system with a sign indicating that work is in progress. This sign must remain in place, even during temporary interruptions to the work.
- The EMERGENCY STOP devices must remain active. If safety functions or safeguards are deactivated during maintenance or repair work, they must be reactivated immediately after the work is completed.



DANGER

Danger to life and limb due to live parts

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

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

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

Cleaning and preventive maintenance work is to be carried out in accordance with the Instructions for Use.

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 (electrostatic sensitive devices).

Voltages in excess of 60 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 LBR Med in this time.

Various components may have elevated temperatures after the robot controller has been switched off. To prevent burns, wait until the components have cooled off.

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

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

Robot

In connection with maintenance and repair work, KUKA Deutschland GmbH must be informed beforehand as to the application for which the LBR Med was used and under what conditions. Information must also be provided as to the possible types of contamination.

3.8.8 Decommissioning, storage and disposal

The LBR Med 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 are used on the LBR Med, safety measures must be taken to ensure full implementation of the principle of “single point of control” (SPOC).

Components:

- Tools for configuration of bus systems with online functionality



The implementation of additional safety measures may be required. This must be clarified for each specific application and is the responsibility of the medical product manufacturer. The medical product manufacturer must evaluate this in its risk management plan and take the appropriate measures.

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.

T1, T2, CRR

In modes T1, T2 and CRR, a robot motion can only be initiated if an enabling switch is held down.

Tools for configuration of bus systems

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

Such tools include:

- KUKA Sunrise.Workbench
- WorkVisual from KUKA
- Tools from other manufacturers

Safety measure:

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

4 **Installing KUKA Sunrise.Workbench**

4.1 PC system requirements

Hardware

Minimum requirements

- PC with Pentium IV processor, min. 1500 MHz
- 2 GB RAM
- 200 MB free hard disk space
- DirectX-compatible graphics card with a resolution of 1024x768 pixels

Recommended specifications

- PC with Pentium IV processor or higher and 2500 MHz
- 4 GB RAM
- 1 GB free hard disk space
- DirectX-compatible graphics card with a resolution of 1280x1024 pixels

Software

- Windows 7

Both the 32-bit version and the 64-bit version can be used.

The following software is required for bus configuration:

- WorkVisual 5.0

4.2 Installing Sunrise.Workbench

Preparation

If an older version of Sunrise.Workbench is already installed:

- Uninstall the old version first.

Precondition

- Local administrator rights

Procedure

1. Start the program **SunriseWorkbench-[...]-Setup.exe**. A window opens.
2. Select the language for the installation procedure and confirm with **OK**.
The language selection only applies to the installation and not to Sunrise.Workbench itself. The default user interface language for Sunrise.Workbench is German.
3. An installation wizard opens. Follow the instructions in the wizard.

4.3 Uninstalling Sunrise.Workbench

Description

When Sunrise.Workbench is uninstalled, all program files are removed from the computer. User-specific files are retained, e.g. the workspace with the Sunrise projects.

Precondition

- Local administrator rights

Procedure

1. Call the list of installed programs in the Windows Control Panel.
2. In the list, select the program **Sunrise Workbench** and uninstall it.

Alternative procedure

- In the Windows Start menu, open the installation directory of Sunrise.Workbench and click on **Uninstall**.

5 Operation of KUKA Sunrise.Workbench

5.1 Starting Sunrise.Workbench

Procedure

1. Double-click on the **Sunrise.Workbench** icon on the desktop.
Alternative:
In the Windows Start menu, open the installation directory and double-click on **Sunrise Workbench**.
The **Workspace Launcher** window opens.
2. In the **Workspace** box, specify the directory for the workspace in which projects are to be saved.
 - A default directory is suggested. The directory can be changed by clicking on the **Browse...** button.
 - If the workspace should not be queried the next time Sunrise.Workbench is started, activate the option **Use this as the default value[...]** (set check mark).
 Confirm the settings with **OK**.
3. A welcome screen opens the first time Sunrise.Workbench is started. There are different options here.
 - Click on **Workbench** to open the user interface of Sunrise.Workbench.
(>>> [5.2 "Overview of the user interface of Sunrise.Workbench" Page 57](#))
 - Click on **New Sunrise project** to create a new Sunrise project directly. The project creation wizard opens.
(>>> [5.3 "Creating a Sunrise project with a template" Page 61](#))

5.2 Overview of the user interface of Sunrise.Workbench

The user interface of KUKA Sunrise.Workbench consists of several views. The combination of several views is called a perspective. KUKA Sunrise.Workbench offers various preconfigured perspectives.

The **Programming** perspective is opened by default. Additional perspectives can be displayed.

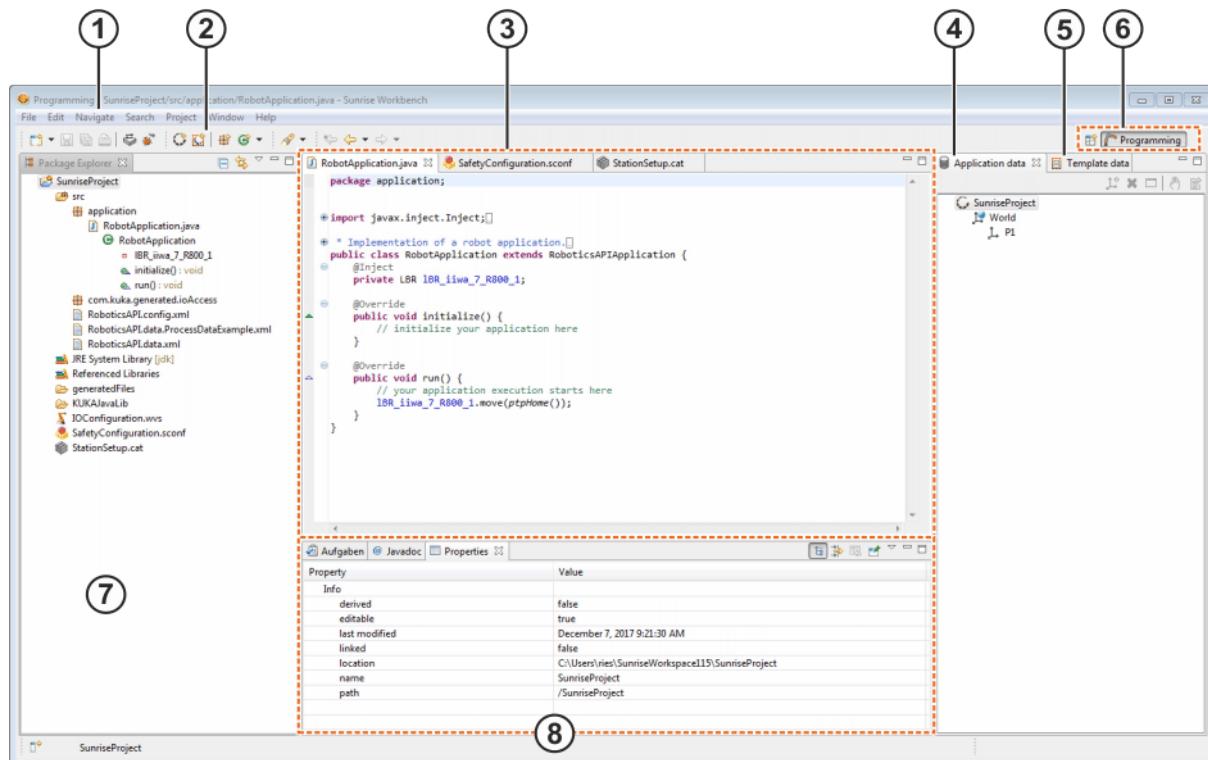


Fig. 5-1: Overview of user interface – “Programming” perspective

Item	Description
1	Menu bar
2	Toolbars (>>> 5.2.4 "Toolbar of the Programming perspective" Page 60)
3	Editor area Opened files, e.g. robot applications, can be displayed and edited in the editor area.
4	Application data view This view displays the frames created for a project in a tree structure.
5	Object templates view This view displays the geometrical objects, tools and workpieces created for a project in a tree structure.
6	Perspective selection It is possible to switch between different perspectives that are already in use by clicking on the name of the desired perspective or selecting it via the Open perspective icon. (>>> 5.2.3 "Displaying perspectives" Page 59)
7	Package Explorer view This view contains the projects created and their corresponding files.

Item	Description
8	Tasks view The tasks that a user has created are displayed in this view.
	Javadoc view The Javadoc comments about the selected elements of a Java application are displayed in this view.
	Properties view The properties of the object, e.g. project, frame or tool, selected in a different view, are displayed in this view.

5.2.1 Repositioning the views

Procedure

1. Grip the view by the title bar while holding down the left mouse button and move it to the desired position on the user interface.
The possible positions for the view are indicated here by a gray frame.
2. Release the mouse button when the desired position for the view is selected.

5.2.2 Closing views and files

Procedure

- Click on the “X” at the top right of the corresponding tab.

5.2.3 Displaying perspectives

Description

The user interface can be displayed in different perspectives. These can be selected via the menu sequence **Window > Open Perspective** or by clicking on the **Open Perspective** icon.

The perspectives are tailored to different types of work:

Perspective	Center of gravity
Programming	This perspective has views suitable for editing Sunrise projects, e.g. for the station configuration, safety configuration and application development.
Debug	This perspective has views suitable for locating faults and eliminating programming faults.

Perspectives can be adapted to the needs of the user. Examples:

- Creating own perspectives
- Showing/hiding views
- Showing/hiding menus
- Showing/hiding menu items

It is possible to save the adapted perspective as a default setting for the perspective or under a separate name of its own.

Procedure

To display views in the current perspective:

- Select the menu sequence **Window > Show View** and the desired view.

Further views can be selected by clicking the menu item **Other....**

To reset the current perspective to the default setting:

- Select the menu sequence **Window > Reset Perspective...** and answer the request for confirmation with **Yes**.

To save user-defined perspectives:

1. Select the menu sequence **Window > Save Perspective As....**
2. In the **Name** box, enter a name for the perspective and confirm it with **OK**.

If an existing perspective is selected and overwritten, the perspective will be opened with these settings in the future.

5.2.4 Toolbar of the Programming perspective

The buttons available as standard on the toolbar depend on the active perspective. The buttons of the **Programming** perspective are described here.

Icon	Name/description
	New Opens the wizard for creating new documents. The arrow can be used to open the menu with the available wizards.
	Save Saves the currently opened and selected file.
	Save All Saves all files and projects that have been edited since the last save.
	Print Opens the menu for printing a file.
	Synchronize project Synchronizes the selected project with the robot controller.
	Debug project Establishes a remote connection to the robot controller in order to debug an application during ongoing operation.
	New Sunrise project Opens the wizard for creating a new Sunrise project.
	New Sunrise application Opens the wizard for creating a new Sunrise application in the selected project. The wizard contains templates for all application types and sample applications suitable for the project.

Icon	Name/description
	New Java package Opens the wizard for creating a new Java package in the selected project.
	New Java class Opens the wizard for creating a new Java class in the selected project. The arrow can be used to open the menu with the available Java classes.
	Search Opens the wizard to search for words or text modules.
	Last Edit Location Switches to the last edit location in the currently opened and selected file.
	Back to ... Switches back to the previous edit steps.
	Forward to ... Switches forward again to the subsequent edit steps.

5.3 Creating a Sunrise project with a template

Procedure

1. Select the menu sequence **File > New > Sunrise project**. The wizard for creating a new Sunrise project is opened.
2. Enter the IP address of the robot controller to be created for the Sunrise project in the **IP address of controller:** box.
It is possible to change the address again during subsequent project configuration.



The following IP address ranges are used by default by the robot controller for internal purposes. IP addresses from these ranges cannot therefore be assigned.

- 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

3. Retain the **Create new project (offline)** setting.
Press **Next >** to switch to the next page.
4. Enter a name for the Sunrise project in the **Project name:** box.
5. The default directory for projects is given in the **Location:** box.
A different directory can be selected: to do so, remove the check mark at **Use default location** and select **Browse....** Then use the **Browse for Folder** dialog to select the desired file path and confirm with **OK**.
Press **Next >** to switch to the next page.
6. Select a template from the **Topology template** list.
The template determines which elements are subsequently preselected in the station configuration on the **Topology** tab. Irrespective of the template that is selected here, all elements are always

available on the **Topology** tab and the preselection can be modified again as required.

Press **Next >** to switch to the next page.

7. If the selected template contains a robot with a media flange, select the appropriate media flange.



The weight and height of the selected media flange are automatically taken into consideration by the system software.

8. As standard, the mounting orientation of a floor-mounted robot is set ($A=0^\circ$, $B=0^\circ$, $C=0^\circ$).

In the case of a ceiling- or wall-mounted robot, enter the direction of installation relative to the floor-mounted robot:

- a. **Rotation about the Z axis in ° (A angle):** Rotation of angle A about the Z axis of the robot base coordinate system ($-180^\circ \leq A \leq 180^\circ$).
- b. **Rotation about the Y axis in ° (B angle):** Rotation of angle B about the Y axis ($-90^\circ \leq B \leq 90^\circ$). The rotation about the Y axis is relative to the rotated coordinate system from step a.
- c. **Rotation about the X axis in ° (C angle):** Rotation of angle C about the X axis ($-180^\circ \leq C \leq 180^\circ$). The rotation about the X axis is relative to the rotated coordinate system from step b.

(>>> [6.13 "Coordinate systems" Page 99](#))



The mounting orientation of the robot must be entered correctly. An incorrectly entered mounting orientation can have the following effects:

- Unexpected robot behavior under impedance control
- Changed position of previously taught frames
- Prevention of motion enable due to collision detection and TCP force monitoring
- Unexpected behavior during jogging in the world or base coordinate system

Press **Next >** to switch to the next page.

9. A summary of information on the Sunrise project is displayed.

Remove the check mark at **Create Sunrise application (starts another wizard)** and click on **Finish**. The Sunrise project is created and added to the **Package Explorer** view.

If the check mark has been set at **Create Sunrise application (starts another wizard)**, the wizard for creation of a new Sunrise application opens. A first Sunrise application can be created directly for the newly-created Sunrise project.

Description

The figure shows the structure of a newly created Sunrise project, for which no Sunrise applications have yet been created or other changes made. The robot configured for the Sunrise project has a media flange.

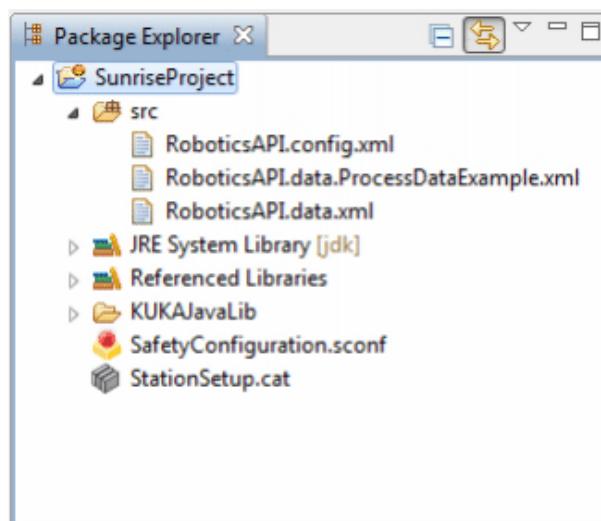


Fig. 5-2: Overview of the project structure

Element	Description
src	<p>Source folder of the project</p> <p>The created robot applications and Java classes are stored in the source folder.</p> <p>The source folder also contains various XML files in which, in addition to the configuration data, the runtime data are saved, e.g. the frames and tools created by the system integrator.</p> <p>The XML files can be displayed but not edited.</p>
JRE System Library	<p>System library for Java Runtime Environment</p> <p>The system library contains the Java class libraries which can be used for standard Java programming.</p>
Referenced libraries	<p>Referenced libraries</p> <p>The referenced libraries can be used in the project. As standard, the robot-specific Java class libraries are automatically added when a Sunrise project is created. It is possible to add further libraries.</p>
KUKAJavaLib	Folder with special libraries required for robot programming
SafetyConfiguration.sconf	<p>Safety configuration</p> <p>The file contains the safety functions preconfigured by KUKA. The configuration can be displayed and edited.</p> <p>(>>> 13 "Safety configuration" Page 237)</p>
StationSetup.cat	<p>Station configuration</p> <p>The file contains the station configuration for the station (controller) selected when the project was created. The configuration can be displayed and edited.</p> <p>The System Software can be installed on the robot controller via the station configuration.</p> <p>(>>> 10 "Station configuration and installation" Page 193)</p>

5.4 Sunrise applications

Sunrise applications are Java programs. They define tasks that are to be executed in a station. They are transferred to the robot controller with the Sunrise project and can be selected and executed using the smartPAD.

There are 2 kinds of Sunrise applications:

- Robot applications

Only one robot application can be executed on the robot controller at any given time.

- Background applications

Multiple background applications can run simultaneously and independently of the running robot application.

Sunrise applications are grouped into Java packages. This makes programming more transparent and makes it easier to use a Java package later in other projects.

5.4.1 Application development

KUKA Sunrise.Workbench is available for application development. In order to implement user-specific system software of the medical device manufacturer, the following interfaces can be used:

- RoboticsAPIApplication
- RoboticsAPIBackgroundTask
- RoboticsAPICyclicBackgroundTasks

Additional libraries or software components of the medical device manufacturer can be integrated via the Sunrise project of Workbench and transferred to the controller.

The medical device manufacturer may only execute software on the robot controller that has been developed using Sunrise.Workbench and synchronized on the robot controller.

Interfaces that can be used:

- ([">>>> 11 "Bus configuration" Page 209](#))
- ([">>>> 12 "External control" Page 221](#))

During application development, the medical device manufacturer must take into consideration the requirements on the software according to IEC 62304 and perform a risk assessment.

([">>>> 13.2 "Software classification" Page 238](#))

5.4.2 Creating a new Java package

Description

A Java package can be created independently of a Sunrise application. The created package does not yet contain any files in this case. An empty package is indicated by a white package icon. As soon as a package contains files, the icon turns brown.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Select the menu sequence **File > New > Package**. The wizard for creating a new Java package is opened.
3. Enter a name for the package in the **Name:** box.

4. Click on **Finish**. The package is created and inserted into the source folder for the project.

5.4.3 Creating a robot application with a package

Description

A robot application can be created together with the Java package into which the application is to be inserted.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Select the menu sequence **File > New > Sunrise application**. The wizard for creating a new Sunrise application is opened.
3. Select the template **RoboticsAPI Application** and click on **Finish**. The wizard for creating a new robot application is opened.
4. In the **Package:** box, enter the name of the package in which the application should be created.
5. Enter a name for the package in the **Name:** box.
6. Click on **Finish**. The application and package are created and inserted into the source folder of the project. The *Name.java* application is opened in the editor area.

5.4.4 Creating a robot application for an existing package

Description

If the Java package into which a robotic application is to be inserted already exists, the application can be created for the existing package.

Procedure

1. In the **Package Explorer** view, select the desired package in the project.
2. Select the menu sequence **File > New > Sunrise application**. The wizard for creating a new Sunrise application is opened.
3. Select the template **RoboticsAPI Application** and click on **Finish**. The wizard for creating a new robot application is opened.
4. Enter a name for the package in the **Name:** box.
5. Click on **Finish**. The application is created and inserted into the package. The *Name.java* application is opened in the editor area.

5.4.5 Creating a new background application

Background applications are Java programs that are executed on the robot controller parallel to the robot application. For example, they can perform control tasks for peripheral devices.

The use and programming of background applications are described here:
(>>> [16 "Background tasks" Page 533](#))

The following properties are defined when the application is created:

- Start type:
 - **Automatic**

The background application is automatically started after the robot controller has booted (default).

- **Manual**

The background application must be started manually via the smartPAD. (This function is not yet supported.)

- Execution type:

- Task template **Cyclic background task**

Template for background applications that are to be executed cyclically (default)

- Task template **Non-cyclic background task**

Template for background applications that are to be executed once

5.4.5.1 Creating a background application with a package

Description

A background application can be created together with the Java package into which the application is to be inserted.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Select the menu sequence **File > New > Sunrise application**. The wizard for creating a new Sunrise application is opened.
3. Select the template **Background task** and click on **Finish**. The wizard for creating a new background application is opened.
4. In the **Package:** box, enter the name of the package in which the application should be created.
5. Enter a name for the package in the **Name:** box.
6. Click on **Next >** and select the desired start type.
7. Click on **Next >** and select the desired execution type (task template).
8. Click on **Finish**. The application and package are created and inserted into the source folder of the project. The *Name.java* application is opened in the editor area.

5.4.5.2 Creating a background application for an existing package

Description

If the Java package into which a background application is to be inserted already exists, the application can be created for the existing package.

Procedure

1. In the **Package Explorer** view, select the desired package in the project.
2. Select the menu sequence **File > New > Sunrise application**. The wizard for creating a new Sunrise application is opened.
3. Select the template **Background task** and click on **Finish**. The wizard for creating a new background application is opened.
4. Enter a name for the package in the **Name:** box.
5. Click on **Next >** and select the desired start type.
6. Click on **Next >** and select the desired execution type (task template).
7. Click on **Finish**. The application is created and inserted into the package. The *Name.java* application is opened in the editor area.

5.4.6 Setting the robot application as the default application

Description

A default application can be defined for every Sunrise project; it is automatically selected after a reboot of the robot controller or synchronization of the project.

In the case of an externally controlled project, it is essential to define a default application. This is automatically selected when the operating mode is switched to Automatic.

Procedure

- Right-click on the desired robot application in the **Package Explorer** view and select **Sunrise > Set as default application** from the context menu.

The robot application is indicated as the default application in the **Package Explorer** view and automatically set as the default application in the project settings.

Example



Fig. 5-3: Default application MainApp.java

5.5 Workspace

The directory in which the created projects and user-defined settings for KUKA Sunrise.Workbench are saved is called the workspace. The directory for the workspace must be defined by the user when Sunrise.Workbench is started for the first time. It is possible to create additional workspaces in KUKA Sunrise.Workbench and to switch between them.

5.5.1 Creating a new workspace

Procedure

- Select the menu sequence **File > Switch Workspace > Other...**. The **Workspace Launcher** window opens.
- In the **Workspace** box, manually enter the path to the new project directory.
Alternative:
 - Click on **Browse...** to navigate to the directory where the new workspace should be created.
 - Create the new project directory by clicking on **Create new folder**. Confirm with **OK**.
 The path to the new project directory is inserted in the **Workspace** box.
- Click on **OK** to confirm the new workspace. Sunrise.Workbench restarts and the welcome screen opens.

5.5.2 Switching to an existing workspace

Precondition

- Other workspaces are available.

Procedure

1. Select the menu sequence **File > Switch Workspace > Other...**. The **Workspace Launcher** window opens.
2. Navigate to the desired workspace using **Browse...** and select it.
3. Confirm with **OK**. The path to the new project directory is applied in the **Workspace Launcher** window.
4. Confirm the selected workspace with **OK**. Sunrise.Workbench restarts and opens the selected workspace.

5.5.3 Switching between the most recently opened workspaces

Precondition

- Other workspaces are available.

Procedure

1. Select the menu sequence **File > Switch Workspace**. The most recently used workspaces are displayed in a list (max. 4).
2. Select the desired workspace from the list. KUKA Sunrise.Workbench restarts and opens the selected workspace.

5.5.4 Archiving projects

Procedure

1. Select the menu sequence **File > Export....** The file export wizard opens.
2. In the **General** folder, select the **Archive File** option and click on **Next >**.
3. All the projects in the workspace are displayed in a list in the top left-hand area of the screen. Select the projects to be archived (set check mark).
4. Click on **Browse...** to navigate to the desired file location, enter the file name for the archive and click on **Save**.
5. Click on **Finish**. The archive file is being created.

5.5.5 Loading projects from archive to the workspace

Precondition

- An archive file (e.g. a ZIP file) with the projects to be loaded is available.
- The workspace does not contain any project with the name of the project to be loaded.

Procedure

1. Select the menu sequence **File > Import....** The file import wizard opens.

2. In the **General** folder, select the **Existing Projects into Workspace** option and click on **Next >**.
3. Activate the **Select archive** file radio button, click on **Browse...** to navigate to the desired archive file and select it.
4. Click on **Open**. All the projects in the archive are displayed in a list under **Projects**.
5. Select projects to be loaded to the workspace.

The following options specify the selection (set the check mark as required):

- **Search for embedded projects**
The projects from the archive are searched for embedded projects.
- **Copying projects to the workspace**
The project is copied into the workspace of Sunrise.Workbench (and thus duplicated). The original project remains unchanged.
Without this option (no check mark), the project remains in the original directory and is modified by Sunrise.Workbench.
- **Hiding projects already in the workspace**
If there are already projects of the same name in the workspace, these projects are hidden.

6. Click on **Finish**. The selected projects are loaded.

5.5.6 Loading projects from the directory to the workspace

Precondition

- One or more projects are available in any directory.
- The workspace does not contain any project with the name of the project to be loaded.

Procedure

1. Select the menu sequence **File > Import....** The file import wizard opens.
2. In the **General** folder, select the **Existing Projects into Workspace** option and click on **Next >**.
3. Activate the **Select root directory** radio button, click on **Browse...** to navigate to the desired directory and select it.
4. Click on **OK**. All the projects in the selected directory are displayed in a list under **Projects**.
5. Select projects to be loaded to the workspace (check mark must be set).
6. Click on **Finish**. The selected projects are loaded.

5.6 Sunrise projects with referenced Java projects

One or more Java projects can be referenced within a Sunrise project. The referencing of Java projects allows them to be used in any number of Sunrise projects and thus on different robot controllers.

The referenced Java projects can in turn reference further Java projects. Only one Sunrise project may exist among all the cross-referenced projects.



When Sunrise projects are synchronized, referenced Java projects are also transferred onto the robot controller. If a further Sunrise project is referenced within a Sunrise project, synchronization is aborted with an error message.

5.6.1 Creating a new Java project

Procedure

1. Select the menu sequence **File > New > Project....** The project creation wizard opens.
2. In the **Java** folder, select the **Java Project** option and click on **Next >**.
3. Enter the name of the Java project in the **Project name** box.
4. In the **JRE** area, select the JRE version that corresponds to the JRE version of the Sunrise project. This is generally JavaSE-1.6.
5. Click on **Next >** and then on **Finish**.
6. The first time a Java project is created in the workspace – or if the user's preference has not yet been specified in previous Java projects – a query is displayed asking whether the **Java** perspective should be opened.
 - Select **Yes** or **No** as appropriate.
 - If the query should not be displayed when the next Java project is created in the workspace, activate the **Remember my decision** option (set check mark).



In the Java projects, all classes which should be referenced externally must be stored in a defined Java package. If referenced classes are created in the standard package, they cannot be found in the Sunrise project.

5.6.1.1 Inserting robot-specific class libraries in a Java project

Description

If a Java project is used for robot programming, the specific KUKA libraries required for this purpose must be inserted into the project. As standard, these libraries are not contained in a Java project.

The KUKA libraries must be copied from a compatible Sunrise project. Ideally, this should be a Sunrise project in which the Java project is referenced or will be referenced. The precondition for compatibility of referenced projects is that the RoboticsAPI versions match.

Precondition

- At least one compatible Sunrise project is available in the workspace.

Procedure

1. Copy the **KUKAJavaLib** folder of a compatible Sunrise project: Right-click on the folder in the **Package Explorer** and select **Copy** from the context menu.
2. Insert the **KUKAJavaLib** folder into the Java project: Right-click on the desired Java project in the **Package Explorer** and select **Insert** from the context menu.
3. Right-click again on the Java project and select **Build Path > Configure Build Path...** from the context menu. The **Properties for Project** window opens.

4. Select the **Libraries** tab in the **Java Build Path** and click on the **Add JARs...** button. The **JAR Selection** window opens.
5. All the projects in the workspace are displayed in a list. Expand the Java project where the referenced libraries are to be inserted.
6. Expand the **KUKAJavaLib** folder and select the existing JAR files.
7. Confirm the selection with **OK**. The JAR files are inserted on the **Libraries** tab of the build path.
8. Close the window by clicking on **OK**. The referenced libraries are inserted into the Java project.

5.6.2 Referencing Java projects

Description

- The referenced classes are saved in a defined Java package (not in the standard package).
- For Java projects which use referenced KUKA libraries: In the referenced projects, the RoboticsAPI versions must match.

Procedure

1. In the **Package Explorer**, right-click on the project which is to be referenced for the Java project.
2. Select **Build Path > Configure Build Path...** from the context menu. The **Properties for Project** window opens.
3. Select the **Projects** tab in the **Java Build Path** and click on the **Add ...** button. The **Required Project Selection** window opens.
4. All the projects in the workspace are displayed in a list. Select the Java projects to be referenced (set check mark).
5. Confirm your selection with **OK**. The selected projects are inserted on the **Projects** tab of the build path.
6. Close the window by clicking on **OK**.

5.6.3 Canceling the reference to Java projects

Description

References to inadvertently added projects or projects that are not required (any longer) can be removed.

Procedure

1. In the **Package Explorer**, right-click on the project from which referenced projects should be removed.
2. Select **Properties** from the context menu. The **Properties for Project** window opens.
3. Select the **Projects** tab in the **Java Build Path**.
4. Select the projects that are not required and click on **Remove**.
5. Close the window by clicking on **OK**.

5.7 Renaming an element in the Package Explorer

In the **Package Explorer** view, the names of inserted elements can be changed, e.g. the names of projects, Java packages and Java files.

5.7.1 Renaming a project or Java package

Procedure

1. Right-click on the desired project or Java package. Select **Refactoring > Rename** in the context menu. The **Rename Java Project** or **Renamne Java Package** window opens.
2. In the **New name** box, enter the desired name. Confirm with **OK**.

5.7.2 Renaming a Java file

Procedure

1. Right-click on the desired Java file. Select **Refactoring > Rename** in the context menu. The **Rename Compilation Unit** window opens.
2. In the **New name** box, enter the desired name. Click on **Finish**.
3. Possible conflicts are indicated before the renaming is completed. After acknowledging and checking these, click on **Finish** once more.

5.8 Removing an element from Package Explorer

In the **Package Explorer** view, inserted elements can be removed again, e.g. entire projects or individual Java packages and Java files of a project.

5.8.1 Deleting an element from a project

Description

Elements created for a project can be deleted again. The elements are permanently deleted from the workspace and cannot be restored.

It is also possible to remove some – but not all – of the default elements of a project.

Procedure

1. Right-click on the element. Select **Delete** in the context menu.
2. Answer the request for confirmation with **OK**. The element is deleted.

5.8.2 Removing a project from Package Explorer

Description

With this procedure, a project is only removed from the **Package Explorer** and is retained in the directory for the workspace on the data storage medium.

If required, the project can be reloaded from the directory into the workspace. The project is then available again in the **Package Explorer**.

(>>> *5.5.6 "Loading projects from the directory to the workspace"*
Page 69)

Procedure

1. Right-click on the desired project. Select **Delete** in the context menu. A request for confirmation is displayed, asking if the project is really to be deleted.

2. The check box next to **Delete project content on disk (cannot be undone)** is activated by default. Leave it like this.
3. Confirm the request for confirmation with **OK**.

5.8.3 Deleting a project from the workspace

Description

With this procedure, a project is removed from the **Package Explorer** and permanently deleted from the directory for the workspace on the data storage medium. The project cannot be restored.

Procedure

1. Right-click on the desired project. Select **Delete** in the context menu. A request for confirmation is displayed, asking if the project is really to be deleted.
2. Activate the check box next to **Delete project content on disk (cannot be undone)**.
3. Confirm the request for confirmation with **OK**.

5.9 Activating the automatic change recognition

Description

The automatic change recognition is activated as standard in Sunrise.Workbench. If it has been deactivated, this could mean, for example, that the Java classes and files required for use of the signals may not be created in when exporting an I/O configuration from WorkVisual.

Procedure

1. Select the menu sequence **Window > User definitions**. The **User definitions** window is opened.
2. Select **General > Workspace** in the directory in the left area of the window.
3. Activate the check box **Update via native hooks or polling** to activate the automatic change recognition.

5.10 Displaying release notes

Description

The release notes contain information about the versions of the System Software, e.g. new functions or system requirements. They can be displayed in the editor.

Procedure

- Sunrise.OS Release Notes:
Select the menu sequence **Help > Sunrise.OS Release Notes**.
- Sunrise.OS Med Release Notes:
In the Windows Start menu, open the installation directory and double-click on the **readme** folder.

6 Operating the KUKA smartPAD

Description

The smartPAD is the teach pendant for the robot.

The smartPAD has all the operator control and display functions required for operation.

2 models exist:

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

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



The USB interfaces can be used for integrating the controller into the medical product. The medical product manufacturer must ensure that the functionality of the medical product cannot be adversely affected, e.g. by a test for malware or by restriction of the USB devices that may be connected.

Use

In accordance with the intended use, the smartPAD may only be used for start-up and recommissioning (e.g. after maintenance), for maintenance and for troubleshooting by the system integrator or medical device manufacturer. The smartPAD is not intended for operation of the medical device by the end user (e.g. medical personnel).

Misuse

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

6.1 smartPAD

6.1.1 Front view

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

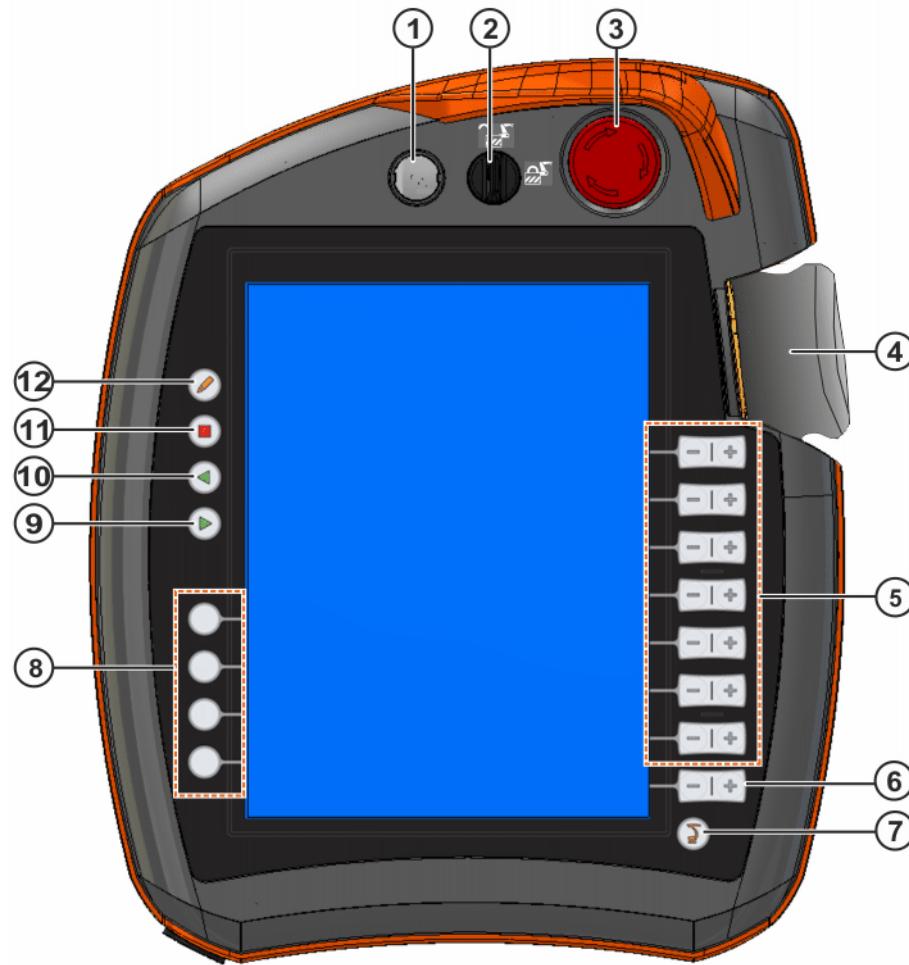


Fig. 6-1: Front of smartPAD

Item	Description
1	Button for disconnecting the smartPAD
2	Keyswitch The connection manager is called by means of the keyswitch. The switch can only be turned if the key is inserted. The connection manager is used to change the operating mode.
3	EMERGENCY STOP device The robot can be stopped in hazardous situations using the EMERGENCY STOP device. The EMERGENCY STOP device locks itself in place when it is pressed.
4	Space Mouse No function
5	Jog keys The jog keys can be used to move the robot manually. (>> 6.16.4 "Cartesian jogging with the jog keys" Page 107)

Item	Description
6	Key for setting the jog override (>>> 6.16.2 "Setting the jog override" Page 105) (>>> 6.19.3 "Setting the manual override" Page 118)
7	Main menu key The main menu key shows and hides the main menu on the smartHMI. (>>> 6.6 "Calling the main menu" Page 92)
8	User keys The function of the user keys is freely programmable. Possible uses of the user keys include controlling peripheral devices or triggering application-specific actions. (>>> 6.11 "Activating the user keys" Page 97)
9	Start key The Start key is used to start a program. The Start key is also used to manually address frames and to move the robot back onto the path. (>>> 6.19 "Program execution" Page 115)
10	Start backwards key No function
11	STOP key The STOP key is used to stop a program that is running.
12	Keyboard key No function



The following applies to the jog keys, the user keys and the Start, Start backwards and STOP keys:

- The current function is displayed next to the key on the smartHMI.
- If there is no display, the key is currently without function.

6.1.2 Rear of smartPAD



Fig. 6-2: Rear of smartPAD

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

Item	Description
1	<p>Enabling switch</p> <p>The enabling switches have 3 positions:</p> <ul style="list-style-type: none"> Not pressed Center position Fully pressed (panic position) <p>In operating modes T1, T2 and CRR, the manipulator can only be moved if one of the enabling switches is held in the center position.</p> <p>In the Automatic operating mode, the enabling switches have no function.</p>
2	<p>Start key (green): the Start key is used to start a program.</p> <p>The Start key is also used to manually address frames and to move the robot back onto the path.</p>
3	Enabling switch
4	USB connection: for FAT32-formatted USB sticks

Item	Description
5	Enabling switch
6	Identification plate

6.2 smartPAD-2

KUKA smartPAD-2

- i Further information about the smartPAD-2 is contained in the operating instructions of the smartPAD-2.
- i 6D mouse on the smartPAD-2 has no function.
- i The firmware update for operation with the smartPAD-2 may only be carried out by KUKA Service.
This update is only possible from KUKA Sunrise.OS Med 1.15.3 onwards.

6.2.1 Front of smartPAD-2

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

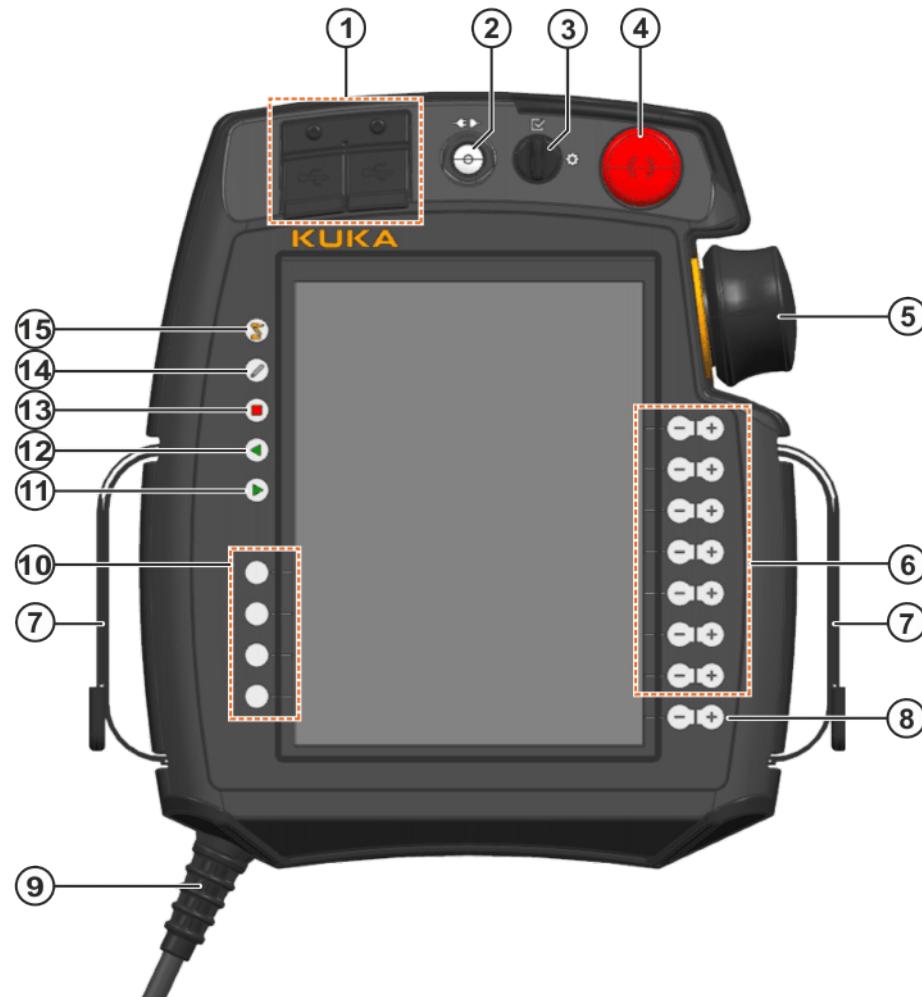


Fig. 6-3: Front of smartPAD-2

Item	Description
1	2 USB 2.0 interfaces with cover: for NTFS and FAT32-formatted USB sticks
2	Button for disconnecting the smartPAD (>>> 6.3 "Disconnecting and connecting the smartPAD" Page 82)
3	Mode selector switch. The switch may be one of the following variants: <ul style="list-style-type: none"> • With key It is only possible to change operating mode if the key is inserted. • Without key The mode selector switch is used to call the connection manager. The connection manager is used to change the operating mode.
4	EMERGENCY STOP device: stops the robot in hazardous situations. The EMERGENCY STOP device locks itself in place when it is pressed.
5	6D mouse: no function
6	Jog keys: for jogging the robot
7	Hand straps with Velcro fastener: when the hand straps are not in use, they can be pulled in completely.
8	Key for setting the override
9	Connecting cable
10	User keys: the function of the user keys is freely programmable. Uses of the user keys include controlling peripheral devices or triggering application-specific actions. (>>> 6.11 "Activating the user keys" Page 97)
11	Start key: the Start key is used to start a program. The Start key is also used to manually address frames and to move the robot back onto the path. (>>> 6.19 "Program execution" Page 115)
12	Start backwards key: no function
13	STOP key: the STOP key is used to stop a program that is running.
14	Keyboard key: no function
15	Main menu key: the main menu key shows and hides the main menu on the smartHMI.



The following applies to the jog keys, the user keys and the Start, Start backwards and STOP keys:

- The current function is displayed next to the key on the smartHMI.
- If there is no display, the key is currently without function.

6.2.2 Rear of smartPAD-2

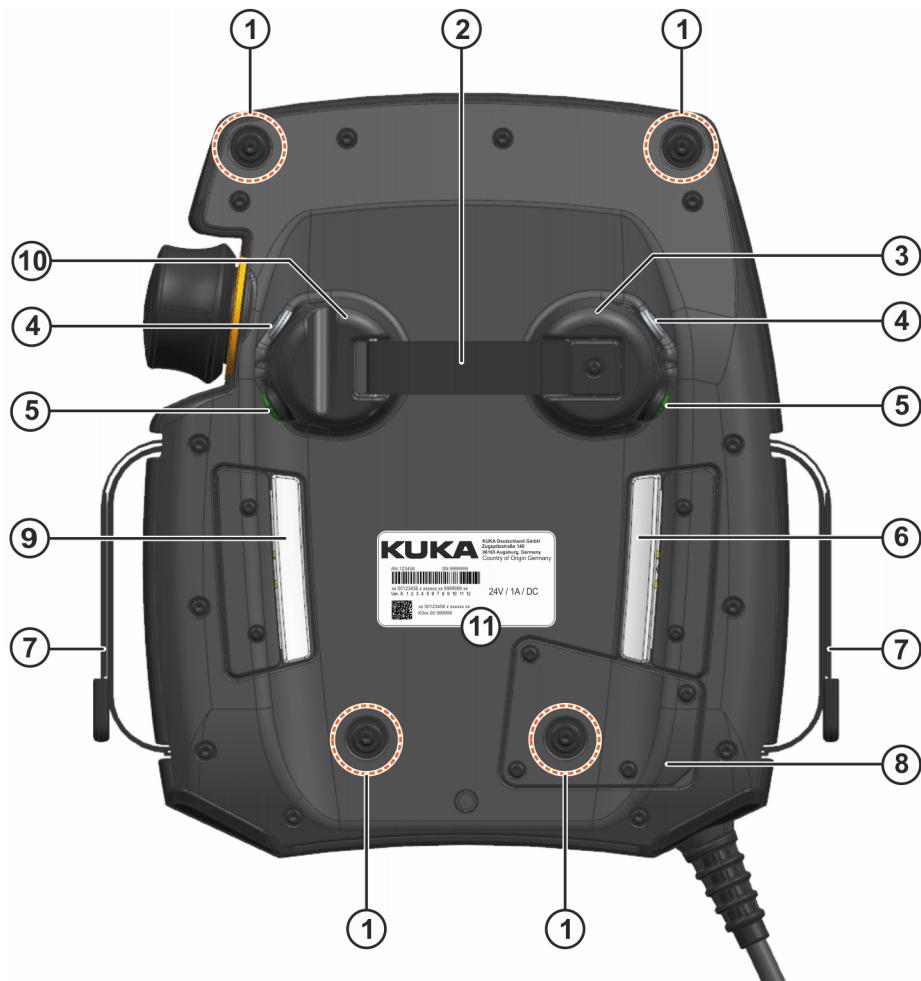


Fig. 6-4: Rear of smartPAD-2

Item	Description
1	Press studs for fastening the (optional) carrying strap
2	Strap, dome
3	Left-hand dome: holding the smartPAD with the right hand
4	Enabling switches The enabling switches have 3 positions: <ul style="list-style-type: none"> • Not pressed • Center position • Fully pressed (panic position) In operating modes T1, T2 and CRR, the manipulator can only be moved if one of the enabling switches is held in the central position. In the Automatic operating mode, the enabling switches have no function.
5	Start key (green): the Start key is used to start a program. The Start key is also used to manually address frames and to move the robot back onto the path.
6	Enabling switches
7	Hand straps with Velcro fastener: when the hand straps are not in use, they can be pulled in completely.

Item	Description
8	Cover (connection cable cover)
9	Enabling switches
10	Right-hand dome: holding the smartPAD with the left hand
11	Identification plate

6.3 Disconnecting and connecting the smartPAD

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



From KUKA Sunrise.OS Med 1.15.3 onwards, the “smartPAD” and “smartPAD-2” models are compatible with one another, i.e. if one model has been disconnected, the other model can then be plugged in.

6.3.1 Disconnecting the smartPAD

Description

If disconnection of the smartPAD is configured as allowed in the station configuration of the project that is active on the robot controller, the smartPAD can be disconnected while the robot controller is running.



WARNING

Risk of fatal injury due to non-operational EMERGENCY STOP device

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

- If unplugging of the smartPAD is to be allowed, connect at least one external EMERGENCY STOP device that is accessible at all times to the robot controller.
- Immediately remove the unplugged smartPAD from the system and store it out of sight and reach.
- Do not disconnect the smartPAD with the EMERGENCY STOP pressed, as the EMERGENCY STOP will remain active in this case until the robot controller is rebooted:
 - Do not use disconnection of the smartPAD to prevent the EMERGENCY STOP device on the smartPAD from being released.
 - If an EMERGENCY STOP is to be active with the smartPAD disconnected, always trigger this EMERGENCY STOP via an external EMERGENCY STOP device.

Precondition

- Disconnection of the smartPAD is allowed.
- The EMERGENCY STOP device on the smartPAD has been released.

Procedure

1. Press the disconnect button on the smartPAD.



Fig. 6-5: Disconnecting the smartPAD button

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

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

2. Disconnect the smartPAD from the robot controller.



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

6.3.2 Connecting the smartPAD

Description

A smartPAD can be connected at any time. The connected smartPAD assumes the current operating mode of the robot controller. The smartHMI is automatically displayed again.



WARNING

The system integrator connecting a smartPAD to the robot controller must subsequently check whether the smartPAD is operational once again.

The smartPAD is not operational in the following cases:

- smartHMI is not displayed again.
It may take more than 30 seconds before the smartHMI is displayed again.
- An error message is displayed in the **Safety** tile, indicating that there is a connection error to the smartPAD.



WARNING

Risk of fatal injury due to non-operational EMERGENCY STOP device

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

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

Procedure

1. Connect the smartPAD to the robot controller.
 - The EMERGENCY STOP and enabling switches are operational again 30 s after connection.
 - The smartHMI is automatically displayed again. (This may take longer than 30 s.)
 - The connected smartPAD assumes the current operating mode of the robot controller.
2. Check the functions. The following checks must be performed:
 - Function test of EMERGENCY STOP
 - Function test for the enabling switches
 - Check whether the smartHMI is displayed again. (This may take longer than 30 s.)
3. If the smartPAD is no longer operational, disconnect it and remove it from the system.

6.4 Update of the smartPAD software



This update mechanism is only applicable for the smartPAD model with 3 enabling switches, not for the smartPAD-2. The update of the smartPAD-2 software is described in the smartPAD-2 operating instructions.

Description

The smartPAD software version is checked automatically in the following cases:

- Reboot of robot controller
- Connection of smartPAD to a running robot controller

If the version check reveals a conflict between the smartPAD software and the system software on the robot controller, the update of the smartPAD software is started automatically.



If the smartPAD is connected to the robot controller with the EMERGENCY STOP pressed, the running application is paused in AUT mode. To resume the application:

1. Release EMERGENCY STOP device.
2. Press the Start key.

NOTICE

Material damage due to interruption of the software update

If the software update is interrupted, this may result in damage to the smartPAD.

- Do not disconnect the smartPAD from the robot controller during the update.
- Do not disconnect the robot controller from the power supply during the update.

Procedure

1. No user input may be entered during the smartPAD update.
2. In the case of a successful update, the smartPAD is automatically rebooted.

6.5 KUKA smartHMI user interface

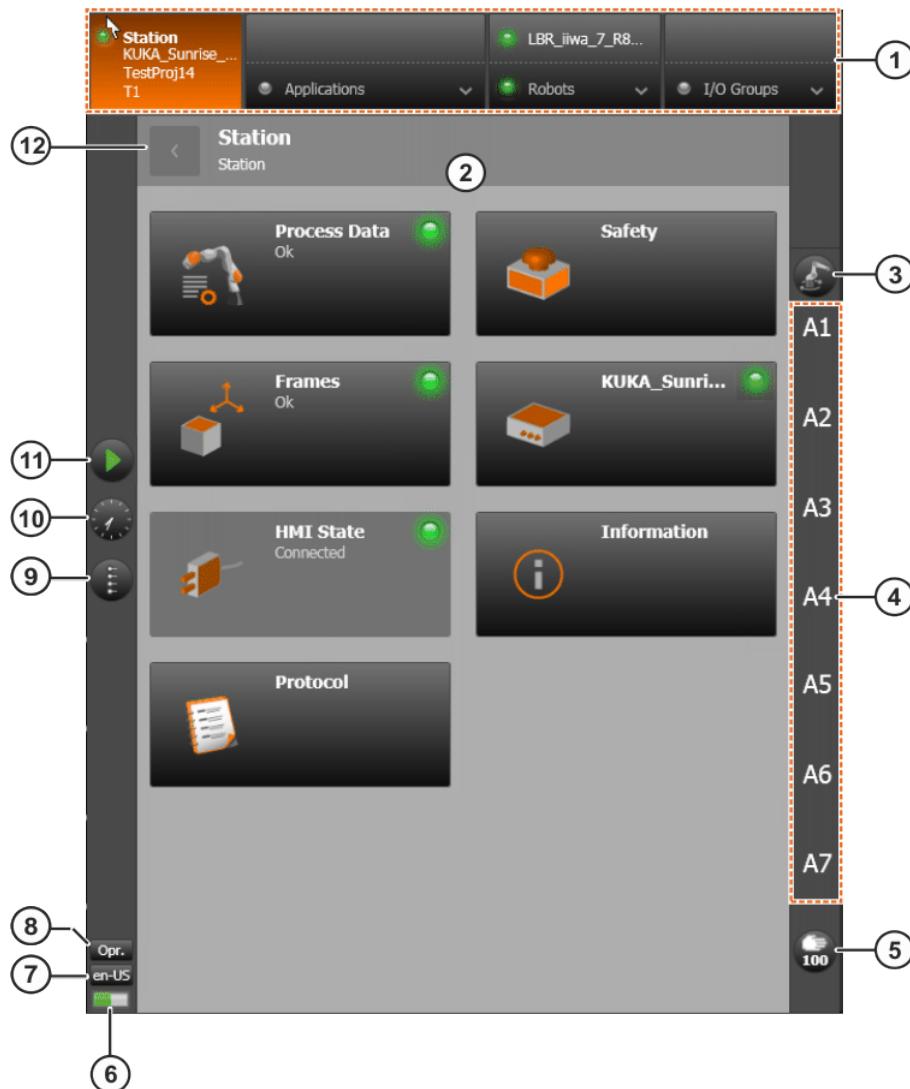


Fig. 6-6: KUKA smartHMI user interface

Item	Description
1	Navigation bar: main menu and status display (>>> 6.5.1 "Navigation bar" Page 86)
2	Display area Display of the level selected in the navigation bar, here the Station level
3	Jogging options button Displays the current coordinate system for jogging with the jog keys. Touching the button opens the Jogging options window, in which the reference coordinate system and further parameters for jogging can be set. (>>> 6.16.1 ""Jogging options" window" Page 103)

Item	Description
4	<p>Jog keys display</p> <p>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). In the case of an LBR, the elbow angle (R) for executing a null space motion is additionally displayed.</p> <p>(>>> 6.16 "Jogging the robot" Page 103)</p>
5	<p>Override button</p> <p>Indicates the current override. Touching the button opens the Override window, in which the override can be set.</p> <p>(>>> 6.14 "Override" window" Page 100)</p>
6	<p>Life sign display</p> <p>A steadily flashing life sign indicates that the smartHMI is active.</p>
7	<p>Language selection button</p> <p>Indicates the currently set language. Touching the button opens the Language selection menu, in which the language of the user interface can be changed.</p>
8	<p>User group button</p> <p>Indicates the currently logged-on user group. Touching the button opens the Login window, in which the user group can be changed.</p> <p>(>>> 6.8.1 "Changing user group" Page 96)</p>
9	<p>User key selection button</p> <p>Touching the button opens the User key selection window, in which the currently available user key bars can be selected.</p> <p>(>>> 6.11 "Activating the user keys" Page 97)</p>
10	<p>Clock button</p> <p>The clock displays the system time. Touching the button displays the system time in digital format, together with the current date.</p>
11	<p>Jogging type button</p> <p>Displays the currently set mode of the Start key. Touching the button opens the Jogging type window, in which the mode can be changed.</p> <p>(>>> 6.15 "Jogging type" window" Page 101)</p>
12	<p>Back button</p> <p>Return to the previous view by touching this button.</p>

6.5.1 Navigation bar

The navigation bar is the main menu of the user interface and is divided into 4 levels. It is used for navigating between the different levels.

Some of the levels are divided into two parts:

- Lower selection list: opens a list for selecting an application, a robot or an I/O group, depending on the level.

- Upper button: if a selection has been made in the list, this button can be used to display the selected application, robot or I/O group.

Alternatively, the main menu can be called using the main menu key on the smartPAD. The main menu contains further menus which cannot be accessed from the navigation bar.

(>>> [6.6 "Calling the main menu" Page 92](#))

Overview



Fig. 6-7: KUKA smartHMI navigation bar

Item	Description
1	Station level Displays the controller name and the selected operating mode (>>> 6.5.4 "Station level" Page 88)
2	Applications level Displays the selected robot application (>>> 6.19.1 "Selecting a robot application" Page 115) All robot and background applications are listed under Applications .
3	Robot level Displays the selected robot (>>> 6.5.5 "Robot level" Page 90)
4	I/O groups level Displays the selected I/O group (>>> 6.20.5 "Displaying an I/O group and changing the value of an output" Page 125)

6.5.2 Status display

The status of the system components is indicated by colored circles on the smartHMI.

The “collective status” is displayed in the lower part of the navigation bar. The status of each of the selected components is displayed in the upper part. For example, it is possible for one application to be executed while another application is in the error state.

Status	Description
	Serious error The system component cannot be used. The reason for this may be an operator error or an error in the system component.
	Warning There is a warning for the system component. The operability of the component may be restricted. It is therefore advisable to remedy the problem. For applications, the yellow status indicator means that the application is paused.
	Status OK There are no warnings or faults for the system component.
	Status unknown The status of the system component cannot be determined.

6.5.3 Keypad

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 appropriate keypad.



Fig. 6-8: Example of keypad



SYM must be pressed to activate the secondary characters assigned to the keys, e.g. the “=” character on the “S” key. The key remains activated for one keystroke. In other words, it does not need to be held down.

6.5.4 Station level

The Station level provides access to information and functionalities which affect the entire station.

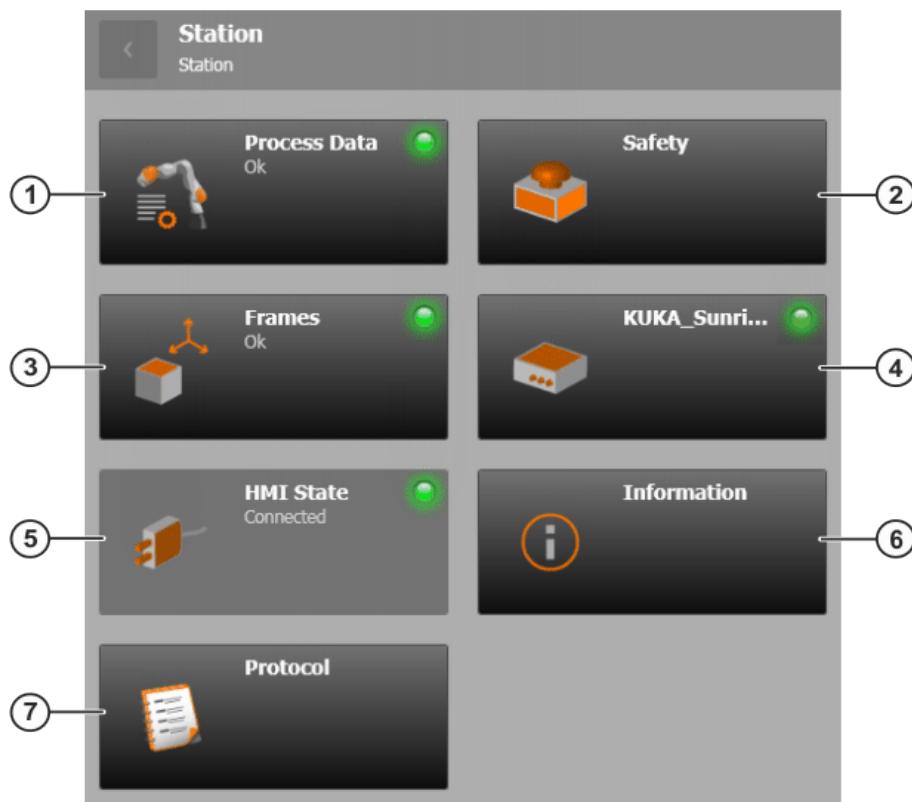


Fig. 6-9: Station level

Item	Description
1	Process data tile Opens the Process data view.
2	Safety tile Indicates the safety status of the station and opens the Safety sublevel. The sublevel contains the following tiles: <ul style="list-style-type: none">• Activation Opens the Activation view for activating and deactivating the safety configuration. A precondition for activation/deactivation is the user group “Safety maintenance”.• State Opens the State view and displays error messages relating to the safety controller.
3	Frames tile Opens the Frames view. The view contains the frames created for the station. (>>> 6.18.1 ““Frames” view” Page 109)

Item	Description
4	<p>Robot controller tile</p> <p>Indicates the status of the robot controller and opens a sub-level. The sublevel contains the following tiles:</p> <ul style="list-style-type: none"> • Boot state Indicates the boot status of the robot controller. • Field buses Indicates the status of the field buses. The tile is only displayed if I/O groups have been created and corresponding signals have been mapped with WorkVisual. • Backup Manager Opens the Backup Manager view. The tile is only displayed if the Backup Manager has been installed. (>>> 6.21 "Backup Manager" Page 128) • Virus scanner Opens the Virus scanner view. The tile is only displayed if the virus scanner has been installed. (>>> 20.4 "Displaying messages of the virus scanner" Page 603)
5	<p>HMI state tile</p> <p>Displays the connection status between the smartHMI and the robot controller.</p>
6	<p>Information tile</p> <p>Opens the Information view and displays system information, e.g. the IP address of the robot controller. (>>> 6.20.6 "Displaying information about the robot and robot controller" Page 128)</p>
7	<p>Log tile</p> <p>Opens the Log view and displays the logged events and changes in state of the system. The display can be filtered based on various criteria. (>>> 20.2 "Displaying a log" Page 597)</p>

6.5.5 Robot level

The Robot level gives access to information and functionalities which affect the selected robot.



Fig. 6-10: Robot level

Item	Description
1	Axis position tile Opens the Axis position view. The axis-specific actual position of the robot is displayed. (>>> 6.20.2 "Displaying the axis-specific actual position" Page 123)
2	Cartesian position tile Opens the Cartesian position view. The Cartesian actual position of the robot is displayed. (>>> 6.20.3 "Displaying the Cartesian actual position" Page 124)
3	Axis torques tile Opens the Axis torques view. The axis torques of the robot are displayed if a sensitive robot is involved. (>>> 6.20.4 "Displaying axis-specific torques" Page 125)
4	Mastering tile Opens the Mastering view. The mastering status of the robot axes is displayed. The axes can be mastered or unmastered individually. (>>> 7.4 "Position mastering" Page 137)

Item	Description
5	Load data tile Opens the Load data view for automatic load data determination. (>>> 7.6 "Determining tool load data" Page 147)
6	Motion enable tile Displays whether the robot has received the motion enable.
7	Log tile Opens the Log view and displays the logged events and changes in state of the system. The display can be filtered based on various criteria. As standard, the Source(s) filter is already set on the robot in question. (>>> 20.2 "Displaying a log" Page 597)
8	Device state tile Indicates the status of the drive system of the robot.
9	Calibration tile Opens a sublevel. This contains the following tiles: <ul style="list-style-type: none"> • Base calibration (>>> 7.5.2 "Base calibration: 3-point method" Page 145) • Tool calibration (>>> 7.5.1 "Tool calibration" Page 138)

6.6 Calling the main menu

Procedure

- Press the main menu key on the smartPAD. The **Main menu** view opens.

Description

Properties of the **Main menu** view:

- The main menu is displayed in the left-hand column. The first 4 buttons are identical to the levels in the navigation bar.
- Touching a button that contains an arrow opens the relevant areas for the level, e.g. **Station**.

Further navigation options are described in the following table.



Fig. 6-11: Example view of the main menu

Item	Description
1	Back button Touch this button to return to the view which was visible before the main menu was opened.
2	Home button Closes all opened areas.
3	Button for closing the level Closes the lowest opened level.
4	The views most recently opened from the main menu are displayed here (maximum 3). By touching the view in question, it is possible to switch to these views again without having to navigate the main menu.

6.7 Setting the user interface language

Description

The user interface on the smartHMI is available in the following languages:

Language	Language abbreviation
Chinese (simplified)	zh
Danish	da
German	de
English	en
Finnish	fi
French	fr
Greek	el
Italian	it
Japanese	ja
Korean	ko
Dutch	nl
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Swedish	sv
Slovak	sk
Slovenian	sl
Spanish	es
Czech	cs
Turkish	tr
Hungarian	hu

Language selection button

The **Language selection** button on the side panel of the smartHMI (bottom left) can be used to select a different language. The button is labeled with the abbreviated name of the active language.

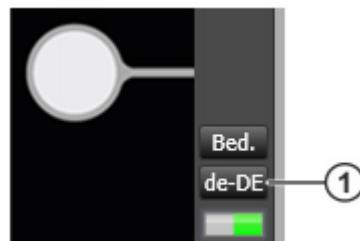


Fig. 6-12: Language selection button

1 Language selection button

Procedure

1. Touch the **Language selection** button. The **Language selection** menu is opened.
2. Select the desired language.

6.8 User groups

Description

Different functions can be executed on the robot controller, depending on the user group.

The following user groups are available as standard:

- **Operator**

The user group “Operator” is the default user group.

- **Safety maintenance technician**

The user “Safety maintenance” is responsible for starting up the safety equipment of the robot. Only he can modify the safety configuration on the robot controller.

The user group is protected by means of a password.

If the option Sunrise.RolesRights is used, there is a further user group:

- **Expert**

The user group “Expert” can perform protected functions that can no longer be performed by the “Operator”.

The user group is protected by means of a password.

User privileges

If the user group “Expert” is installed, the user rights of the operator are restricted. The user rights are then assigned as follows:

Function	Operator	Expert	Safety maintenance
Selecting/deselecting an application	✓	✓	✓
Pausing an application	✓	✓	✓
Moving the robot manually	✓	✓	✓
Selecting an operating mode (T1, T2, CRR, AUT)	✓	✓	✓
Activating/deactivating/resetting the safety configuration	✗	✗	✓
Changing the value of an output	✗	✓	✓
Teaching frames	✗	✓	✓
Creating a new frame	✗	✓	✓
Robot mastering/unmastering	✗	✓	✓

6.8.1 Changing user group

Description

When the robot controller is rebooted, the default user group is selected. The **User group** button on the side panel of the smartHMI (bottom left) can be used to switch to a different user group. The button is labeled with the abbreviated name of the active user group.

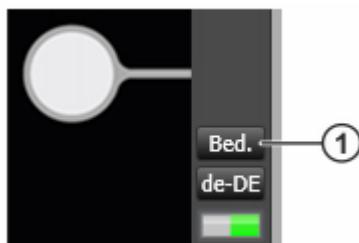


Fig. 6-13: User group button

1 User group button

If no actions are carried out on the user interface within 5 minutes, the robot controller switches to the default user group for safety reasons.

Procedure

Logging on a user group:

1. Touch the **User group** button. The **Login** window opens.
2. Select the desired user group.



If a functionality is called for which the currently logged-on user group has insufficient rights, the **Login** window opens automatically. The required user group is then preset in the window.

3. Enter the password and confirm with **Login**. The **Login** window closes and the selected user group is active.

Logging off a user group:

1. Touch the **User group** button. The **Login** window opens.
2. Touch the **Log off** button. The **Login** window closes and the default user group is active again.

6.9 CRR mode – controlled robot retraction

Description

CRR is an operating mode to which the system can be switched when the robot is stopped by the safety controller for one of the following reasons:

- Robot violates an axis-specific or Cartesian monitoring space.
- Orientation of a safety-oriented tool is outside the monitored range.
- Robot violates a force or axis torque monitoring function.
- A position sensor is not mastered or referenced.
- An axis torque sensor is not referenced.

Once the operating mode has been switched to CRR, the robot can be moved again.

Use

CRR mode can be used, for example, to retract the robot in the case of a space or force monitoring violation or to master the robot with a Cartesian velocity monitoring function active.

If the cause of the stop is no longer present and if no further stop is requested for 4 seconds by one of the specified causes, the operating mode automatically changes to T1.

Motion velocity

The motion velocity of the set working point in CRR mode corresponds to the jog velocity in T1 mode:

- Program mode: Reduced programmed velocity, maximum 250 mm/s
- Jog mode: Jog velocity, maximum 250 mm/s
- Manual guidance: No limitation of the velocity, but safety-oriented velocity monitoring functions in accordance with the safety configuration

6.10 Changing the operating mode

Description

The operating mode can be set with the smartPAD using the connection manager.



It is possible to change the operating mode while an application is running on the robot controller. The robot then stops with a safety stop 1 and the application is paused. Once the new operating mode has been set, the application can resume.

Precondition

- The key is in the switch for calling the connection manager

Procedure

1. Turn the mode selector switch on the smartPAD. The connection manager is displayed.
2. Select the operating mode.
3. Return the mode selector switch to its original position.

The selected operating mode is now active and is displayed in the navigation bar of the smartHMI.

6.11 Activating the user keys

Description

The user keys on the smartPAD can be assigned functions. All the user key functions of a running application are available to the operator. In order to be able to use the desired functions, the operator must activate the corresponding user key bar.

Procedure

1. Touch the **User key selection** button on the left side panel of the smartHMI.
The **User key selection** window opens. The user key bars currently available are displayed.

2. Select the desired user key bar by pressing the corresponding name button.
The text or image on the smartHMI next to the user keys changes according to the bar selected. The user keys now have the corresponding functions.
3. Touch the **User key selection** button or an area outside the window.
The **User key selection** window closes.

Example

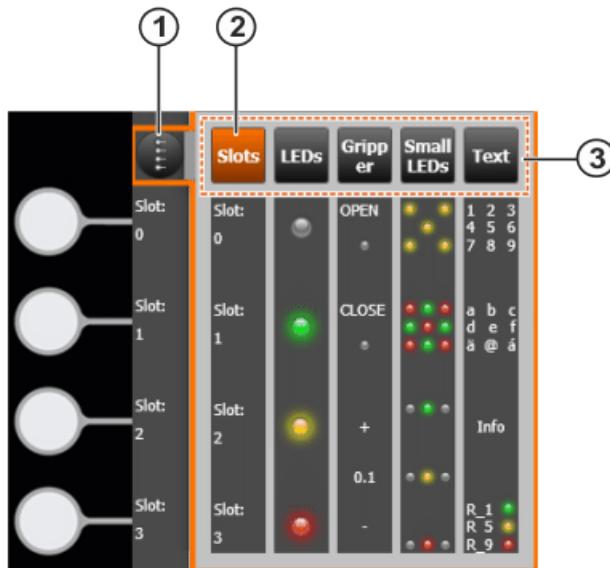


Fig. 6-14: “User key selection” window

- 1 **User key selection** button
- 2 Currently active user key bar
- 3 Available user key bars

6.12 Resuming the safety controller

Description

If there are connection or periphery errors, the safety controller is paused (after one or more occurrences depending on the error). Pausing the safety controller causes the robot to stop and all safe outputs to be switched off. The application can resume once the error has been rectified.

Procedure

1. Select the **Safety > State** tile at the Station level. The **State** view opens.
The cause of the error is displayed in the view. The **Resume safety controller** button is not active.
2. Eliminate the error. The **Resume safety controller** button is now activated.
3. Touch the **Resume safety controller** button. The safety controller is resumed.

6.13 Coordinate systems

Coordinate systems or frames determine the position and orientation of an object in space.

Overview

The following coordinate systems are relevant for the robot controller:

- World
- Robot base
- Base
- Flange
- Tool

Description

World coordinate system

The world coordinate system is a permanently defined Cartesian coordinate system. It is the root coordinate system for all other coordinate systems, in particular for base coordinate systems and the robot base coordinate system.

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

Robot base coordinate system

The robot base 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 robot base coordinate system is identical to the world coordinate system. It is possible to define a rotation of the robot relative to the world coordinate system by changing the mounting orientation in Sunrise.Workbench. By default, the mounting orientation of the floor-mounted robot is set ($A=0^\circ$, $B=0^\circ$, $C=0^\circ$).

Base coordinate system

In order to define motions in Cartesian space, a reference coordinate system (base) must be specified.

As standard, the world coordinate system is used as the base coordinate system for a motion. Additional base coordinate systems can be defined relative to the world coordinate system.

(>>> [7.5.2 "Base calibration: 3-point method" Page 145](#))

Flange coordinate system

The flange coordinate system describes the current position and orientation of the robot flange center point. It does not have a fixed location and is moved with the robot.

The flange coordinate system is used as an origin for coordinate systems which describe tools mounted on the flange.

Tool coordinate system

The tool coordinate system is a Cartesian coordinate system which is located at the working point of the mounted tool. This is called the TCP (Tool Center Point).

Any number of frames can be defined for a tool and can be selected as the TCP. The origin of the tool coordinate system is generally identical to the flange coordinate system.

(>>> [9.3.1 "Geometric structure of tools" Page 168](#))

The tool coordinate system is offset to the tool center point during calibration.

(>>> [7.5.1 "Tool calibration" Page 138](#))

Position and orientation

In order to determine the position and orientation of an object, translation and rotation relative to a reference coordinate system are specified. 6 coordinates are used for this purpose.

Translation

Coordinate	Description
Distance X	Translation along the X axis of the reference system
Distance Y	Translation along the Y axis of the reference system
Distance Z	Translation along the Z axis of the reference system

Rotation

Coordinate	Description
Angle A	Rotation about the Z axis of the reference system
Angle B	Rotation about the Y axis of the reference system
Angle C	Rotation about the X axis of the reference system

6.14 “Override” window

Procedure

To open the **Override** window:

- Touch the **Override** button.

To close the **Override** window:

- Touch the **Override** button or an area outside the window.

Description

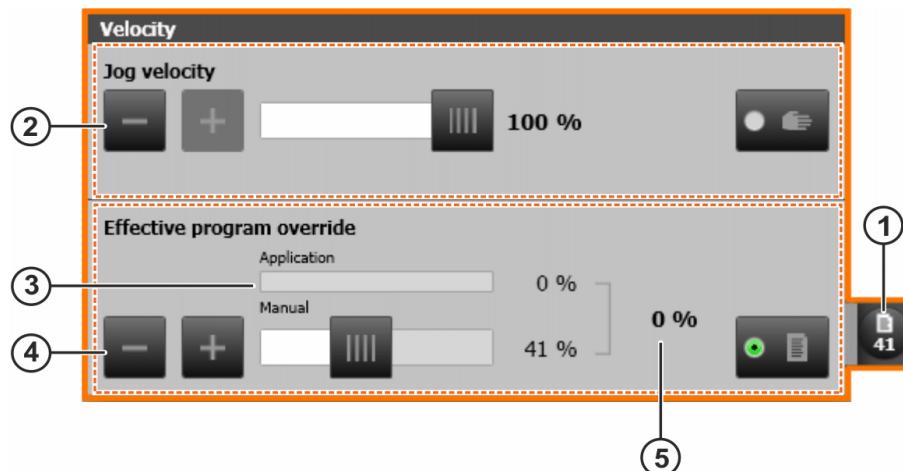


Fig. 6-15: Override window

Item	Description
1	Override button The display on the button depends on the selected option.
2	Set the jog velocity. (>> 6.16.2 "Setting the jog override" Page 105)

Item	Description
3	Display of application override If an application override set by the application is programmed, this is displayed during program execution.
4	Set the manual override. (>>> 6.19.3 "Setting the manual override" Page 118) If no application override is active, the manual override that can be set here corresponds to the effective program override.
5	Display of effective program override

The following buttons are available:

Option	Button	Description
		When the Set jog override option is selected, the Override button displays the hand icon and the jog override currently set.
		When the Set manual override option is selected, the Override button displays the program icon and the manual override currently set.

6.15 “Jogging type” window

Procedure

Open the **Jogging type** window:

- Touch the **Jogging type** button next to the Start key.

Close the **Jogging type** window.

- Touch the **Jogging type** button or an area outside the window.

Description

The functionality of the Start key can be configured in the **Jogging type** window.

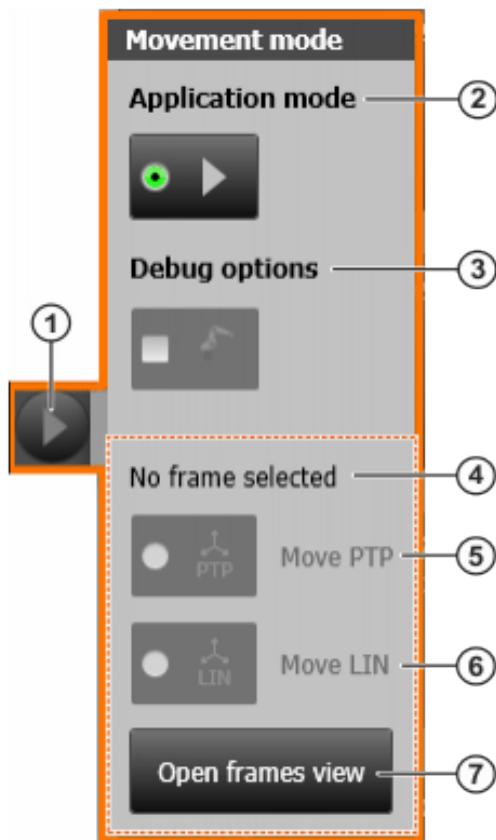


Fig. 6-16: “Jogging type” window

Item	Description
1	Jogging type button The display on the button depends on the selected jogging type.
2	Application mode jogging type In this jogging mode an application can be started by means of the Start key. Note: When switching to T2 or Automatic mode, Application mode is set automatically.
3	Changing program run mode (>>> 6.19.2 "Setting the program run mode" Page 117)
4	Frame name display The name of the frame is displayed if a frame has been selected in the Frames view.
5	Move PTP jogging type A taught frame can be addressed with a PTP motion by means of the Start key. (>>> 6.18.5 "Manually addressing frames" Page 115) The button for selecting the jogging type is only active if a frame has been selected in the Frames view. Note: In the Move PTP jogging type, the Status of the end frame is taken into consideration. This can cause the axes to move, even if the end point has already been reached in Cartesian form.

Item	Description
6	<p>Move LIN jogging type</p> <p>A taught frame can be addressed with a LIN motion by means of the Start key.</p> <p>(>>> 6.18.5 "Manually addressing frames" Page 115)</p> <p>The button for selecting the jogging type is only active if a frame has been selected in the Frames view.</p> <p>Note: In the Move LIN jogging type, the Status of the end frame is not taken into consideration.</p>
7	<p>Open frames view button</p> <p>Press the button to switch to the Frames view.</p>

Icons

The following icons are displayed on the **Jogging type** button depending on the set jogging type:

Icon	Description
	Application mode jogging type
	Move PTP jogging type
	Move LIN jogging type

6.16 Jogging the robot

Overview

There are 2 ways of jogging the robot:

- Cartesian jogging
The set TCP is jogged in the positive or negative direction along the axes of a coordinate system or rotated about these axes.
- Axis-specific jogging
Each axis can be moved individually in the positive or negative direction.

6.16.1 “Jogging options” window

Procedure

Open the **Jogging options** window:

- Touch the **Jogging options** button.

Close the **Jogging options** window.

- Touch the **Jogging options** button or an area outside the window.

Description

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

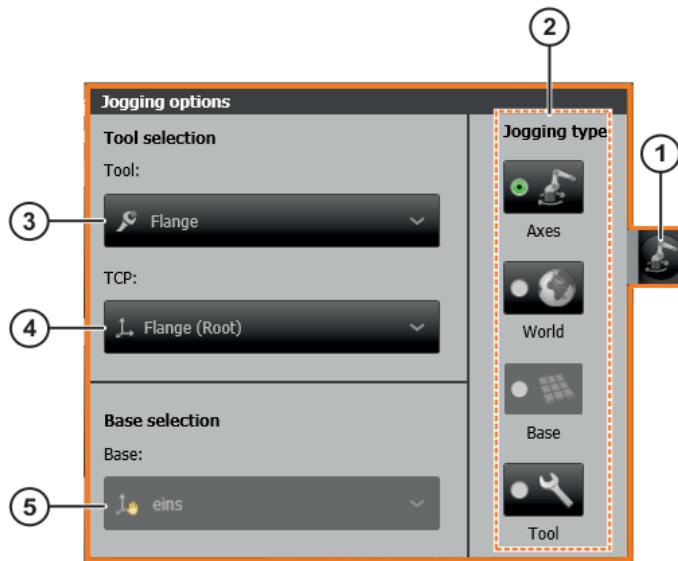


Fig. 6-17: “Jogging options” window

Item	Description
1	Jogging options button The icon displayed depends on the programmed jogging type.
2	Select the jogging type. Axis-specific jogging or Cartesian jogging of the robot in different coordinate systems is possible. The selected jogging type is indicated in green and displayed on the Jogging options button. <ul style="list-style-type: none"> • Axes: The robot is moved by axis-specific jogging. • World: The selected TCP is moved in the world coordinate system by means of Cartesian jogging. • Base: The selected TCP is moved in the selected base coordinate system by means of Cartesian jogging. • Tool: The selected TCP is moved in its own tool coordinate system by means of Cartesian jogging.
3	Select the robot flange or mounted tool. Not possible while an application is being executed. The frames of the selected tool can be selected as the TCP for Cartesian jogging. The set load data of the tool are taken into consideration. If a robot application is paused, the tool currently being used in the application is available under the name Application tool . (>>> <i>"Application tool" Page 105</i>)

Item	Description
4	Select the TCP. All the frames of the selected tool are available as the TCP. The TCP set here is retained. This is also the case if a different TCP is active in a paused application. Exception: If a robot application is paused and the application tool is set, the manually set TCP is not retained when the application is resumed. The TCP changes according to the TCP currently used in the application. (>>> <i>"Application tool" Page 105</i>)
5	Base selection. Only possible when the jogging type Base is selected. All frames which were designated in Sunrise.Workbench as a base are available as a base.

Application tool

The application tool consists of all the frames located below the robot flange during the runtime. These can be the frames of a tool or work-piece, for example, that are connected to the robot flange with the attach-To command. They may also include frames generated in the application and linked directly or indirectly to the flange during the runtime.

The application tool is then only available in the jogging options when a robot application is paused, and if a motion command was sent to the robot controller prior to pausing.

- If the application tool is set in the jogging options, all frames located hierarchically under the flange coordinate system during the runtime can be selected as the TCP for jogging. The origin frame of the application tool on the robot flange is available under the name **ApplicationTool(Root)** for selection as the TCP for jogging.
- If the application tool is set in the jogging options and the application resumed, the following occurs: the frame with which the current motion command is executed in the application is automatically set as the TCP.

6.16.2 Setting the jog override

Description

The 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. However, the velocity of the set working point cannot exceed 250 mm/s.

Procedure

1. Touch the **Override** button. The **Override** window is opened.
(>>> *6.14 "Override" window" Page 100*)
2. Activate the **Set jog override** option if it is not already active.

Option	Description
	Set jog override option activated

3. 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 override can be set in steps to the following values: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%, 0%.
 - Slider: The override can be adjusted in 1% steps.
4. Touch the **Override** button or an area outside the window to close the window.

Alternative procedure

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

The value can be set in the following steps: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%.

6.16.3 Axis-specific jogging with the jog keys

Precondition

- Operating mode T1

Procedure

1. Select the jogging type **Axes** from the jogging options.
Axes A1 to A7 are displayed next to the jog keys.
2. Set the jog override.
3. Hold down the enabling switch.
When motion is enabled, the display elements next to the jog keys are highlighted in white.
4. Press the plus or minus jog key to move an axis in the positive or negative direction.

Description

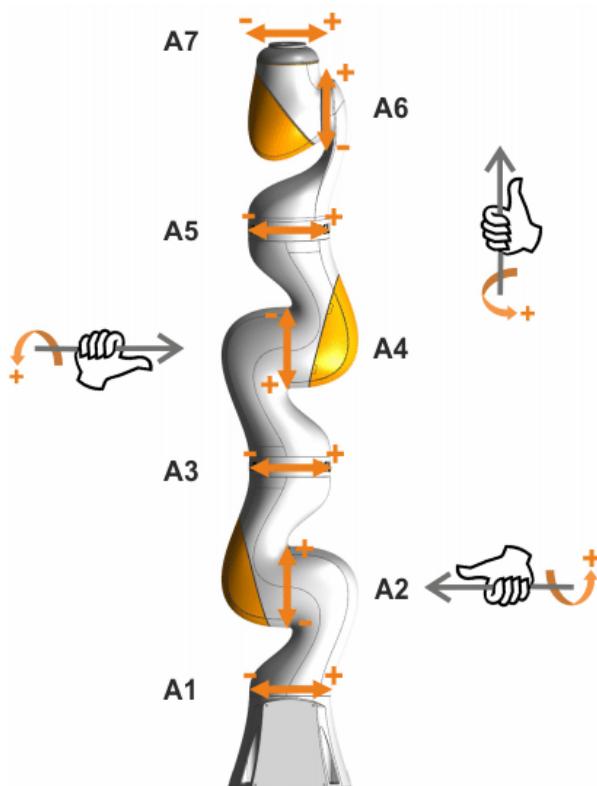


Fig. 6-18: Axis-specific jogging

The positive direction of rotation of the robot axes can be determined using the right-hand rule. Imagine the cable bundle which runs inside the robot from the base to the flange. Mentally close the fingers of your right hand around the cable bundle at the axis in question. Keep your thumb extended while doing so. Your thumb is now positioned on the cable bundle so that it points in the same direction as the cable bundle runs inside the axis on its way to the flange. The other fingers of your right hand point in the positive direction of rotation of the robot axis.

6.16.4 Cartesian jogging with the jog keys

Description

The robot can be moved in a Cartesian sense in the world, base or tool coordinate system using the jog keys.

Precondition

- **World, Base or Tool** is selected as the jogging type in the jogging options.
- In the jogging options, the tool and the desired TCP are selected.
- Only in the case of jogging type **Base**: the desired base is set in the jogging options.



All frames which were designated in Sunrise.Workbench as a base are available as a base.
(>>> [9.2.2 "Designating a frame as a base" Page 163](#))

- The desired jog override is set.
- T1 mode

Procedure

1. Press and hold down the enabling switch.

When motion is enabled, the designations next to the jog keys are highlighted in white.

- **X, Y, Z:** jog keys for linear motions along the X, Y or Z axis of the set coordinate system
- **A, B, C:** jog keys for rotational motions about the X, Y or Z axis of the set coordinate system
- **R:** jog key for null space motion
(>>> [6.16.4.1 "Null space motion" Page 108](#))

2. Press your chosen plus or minus jog key to move the robot in the positive or negative direction.

6.16.4.1 Null space motion

Description

A robot with 7 axes is kinematically redundant. This means that theoretically, it can move to every point in the work envelope with an infinite number of axis configurations.

Due to the kinematic redundancy, a so-called null space motion can be carried out during Cartesian jogging. In the null space motion, the axes are rotated in such a way that the position and orientation of the set TCP are retained during the motion.

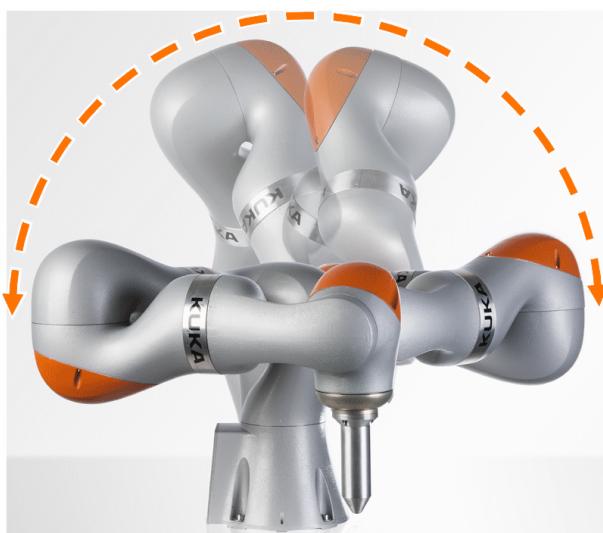


Fig. 6-19: Null space motion

Properties

- The null space motion is carried out via the “elbow” of the robot arm.
- The position of the elbow is defined by the elbow angle (R).
- In Cartesian jogging with the jog keys, the position of the elbow angle (R) can be modified.

Areas of application

- The optimal axis configuration can be set for a given position and orientation of the TCP. This is especially useful in a limited working space.

- When a software limit switch is reached, you can attempt to move the robot out of the range of the limit switches by changing the elbow angle.

6.17 Manually guiding the robot

Description

The robot can be guided using a hand guiding device.

Manual guidance is supported as standard in all operating modes except CRR mode. In the station configuration, it is possible to configure manual guidance as not allowed in Test mode and/or Automatic mode.



CAUTION

In manual guidance, incorrectly selected parameters (e.g. incorrect load data, incorrect tool) or incorrect information (e.g. from defective torque sensors) can be interpreted as external forces. This can result in unpredictable motions of the robot.



CAUTION

If the robot is manually guided, an EMERGENCY STOP device must be installed. It must always be within reach of the operator.

Precondition

- Hand guiding device with safety-oriented enabling device (enabling switch) is present and configured.
- No application is selected or the application has one of the following states:
 - Selected**
 - Motion paused**
 - Error**
- Manual guidance is allowed in the set operating mode.

Procedure

- Press and hold down the enabling switch on the hand guiding device.
- Guide the TCP to the desired position.
- Once the position has been reached, release the enabling switch.

6.18 Frame management

6.18.1 “Frames” view

Procedure

To open the view:

- Select **Frames** at the Station level. The **Frames** view opens.

Description

The view contains the frames created for the station. Additional frames can be created and the frames taught here. The position and orientation of a frame in space and the associated redundancy information are recorded during teaching.

- Taught frames can be addressed manually.
 - Taught frames can be used as end points of motions. If an application is run and the end frame of a motion is addressed, this is selected in the **Frames** view.
- (>>> *6.20.1 "Displaying the end frame of the motion currently being executed" Page 122*)

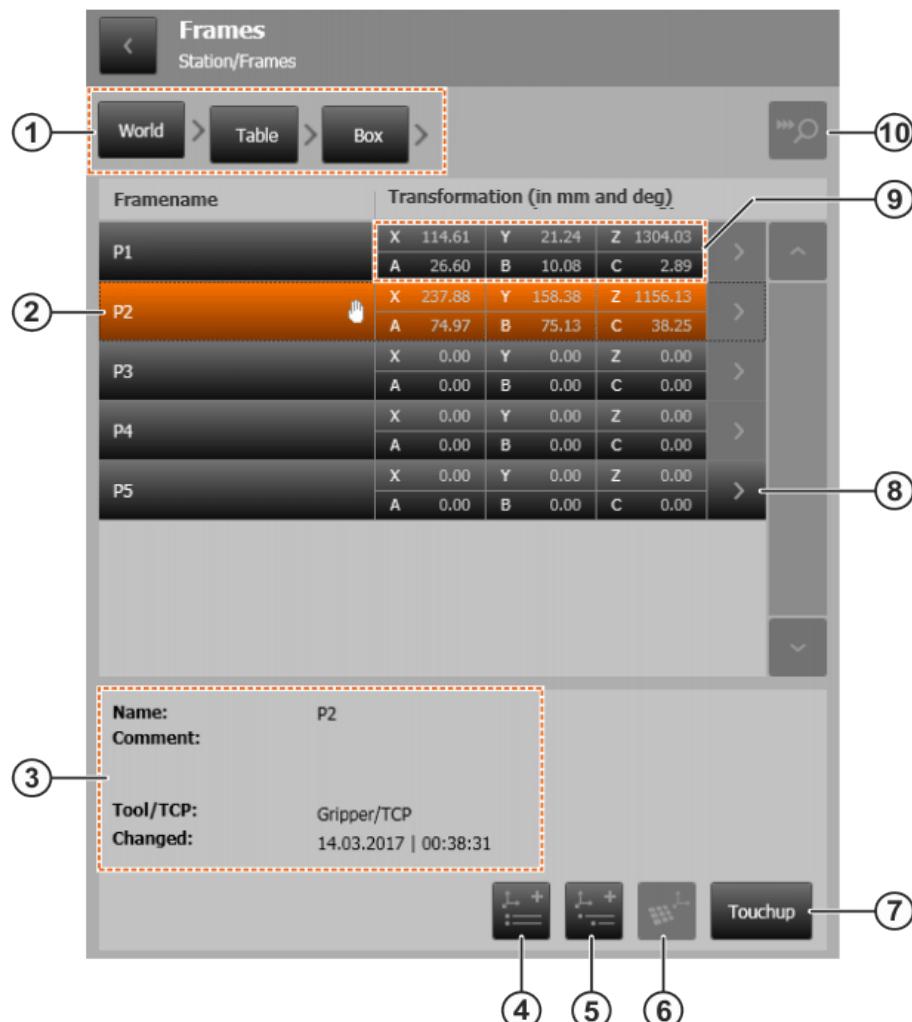


Fig. 6-20: Frames view

Item	Description
1	Frame path Path to the frames of the currently displayed hierarchy level: Goes from World to the direct parent frame (here Box)
2	Frames of the current hierarchy level A frame can be selected by touching it. The frame selected here is marked with a hand icon. The hand icon means that this frame can be used as the base for jogging and can be calibrated.
3	Properties of the selected frame <ul style="list-style-type: none"> • Name of the frame • Comment • Tool used while teaching the frame • Date and time of the last modification

Item	Description
4	Create frame button Creates a frame at the currently displayed hierarchy level.
5	Create child frame button The button can be used to create a child frame for a selected frame. If no frame is selected, the button is disabled.
6	Set base for jogging button The button sets the selected frame as the base for jogging in the jogging options. (>>> 6.16.1 "Jogging options" window Page 103) The button is only active if the Base jogging type is selected from the jogging options and the selected frame is marked as the base in Sunrise.Workbench.
7	Touchup button A selected frame can be taught. If no frame is selected, the button is disabled.
8	Display child frames button The button displays the direct child elements of a frame. The button is only active if a frame has child elements.
9	Frame coordinates with reference to the parent frame
10	Magnifying glass button The magnifying glass button is only active if an application is running and the end frame of a motion is being addressed. Use the button to switch to this end frame if it is not yet displayed.

6.18.2 Creating a frame

Description

If the desired TCP is moved to the position of a new frame, the frame is taught directly on creation. In other words, when a frame is created, the position and orientation of the TCP that is currently selected in the jogging options are automatically applied as frame coordinates.

Precondition

- The tool with the desired TCP is set in the jogging options.
(>>> [6.16.1 "Jogging options" window Page 103](#))



The application tool is only available in the jogging options if the robot application is paused. For this reason, use of the application tool for teaching frames is not recommended.

The tool corresponding to the current application tool (object template of the tool) is also available for selection in the jogging options. Teaching can be carried out with this tool instead of the application tool.

- T1 mode

Procedure

Creating a new frame:

1. Move the TCP to the desired position of the new frame.
2. Press the **Create frame** button. The current TCP coordinates are applied for the frame.



In extreme poses, i.e. if an axis is situated at the edge of, or outside, the permissible axis range, no frame can be created. A corresponding error message is displayed. In this case, move the axis out of the extreme position and address the desired position again.

3. If a child frame is to be created directly for the newly-created frame, move the TCP to the desired position of the new child frame.
4. Press the **Create child frame** button. The current TCP coordinates are applied for the child frame.

Creating a new child frame:

1. Select the frame for which a child frame is to be created.
2. Move the TCP to the desired position of the new child frame.
3. Press the **Create child frame** button. The current TCP coordinates are applied for the child frame.



In extreme poses, i.e. if an axis is situated at the edge of, or outside, the permissible axis range, no frame can be created. A corresponding error message is displayed. In this case, move the axis out of the extreme position and address the desired position again.

6.18.3 Reteaching frames

Description

The coordinates of a frame can be modified on the smartHMI. This is done by moving to the new position of the frame with the desired TCP and teaching the frame. In the process, the new position and orientation are applied.



Once the frames have been taught, it is advisable to synchronize the project immediately in order to update the frame data in the corresponding project in Sunrise.Workbench.

Precondition

- The tool with the desired TCP is set in the jogging options.
(>>> [6.16.1 "Jogging options" window](#) [Page 103](#))



The application tool is only available in the jogging options if the robot application is paused. For this reason, use of the application tool for teaching frames is not recommended.

The tool corresponding to the current application tool (object template of the tool) is also available for selection in the jogging options. Teaching can be carried out with this tool instead of the application tool.

- T1 mode

Procedure

1. Move the TCP to the desired position of the frame.
2. In the **Frames** view, select the frame whose position is to be taught.
3. Press **Touchup** to apply the current TCP coordinates to the selected frame.

4. The coordinates and redundancy information of the taught point are displayed in the **Apply touchup data** dialog. Press **Apply** to save the new values.



If a frame is changed, the change affects all applications in which the frame is used. Modified programs must always be tested first in Manual Reduced Velocity mode (T1).

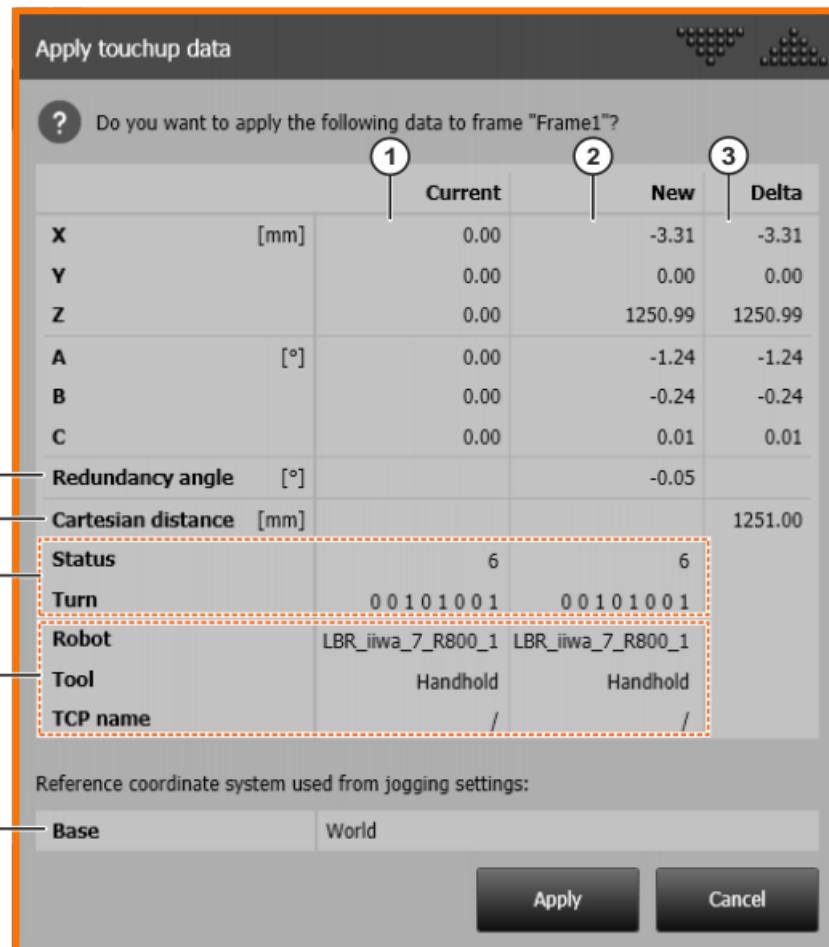


Fig. 6-21: Apply touchup data

Item	Description
1	Values saved up to now
2	New values
3	Changes between the values saved until now and new values
4	Base for jogging All coordinate values of the frame which are displayed in the dialog refer to the jogging base set in the jogging options. These values generally differ from the coordinate values of the frame with respect to its parent frame. (>> 6.16.1 "Jogging options" window Page 103)
5	Information on the robot and tool used during teaching These frame properties are adopted by Sunrise.Workbench when the project is synchronized.

Item	Description
6	Redundancy information on the taught point These frame properties are adopted by Sunrise.Workbench when the project is synchronized.
7	Cartesian distance between the current and new position of the frame

6.18.4 Teaching a frame with the hand guiding device

Description

Frames can be taught using a hand guiding device. Here, the TCP is moved by hand to the desired position.

Manual guidance is supported as standard in all operating modes except CRR mode. In the station configuration, it is possible to configure manual guidance as not allowed in Test mode and/or Automatic mode.



CAUTION

In manual guidance, incorrectly selected parameters (e.g. incorrect load data, incorrect tool) or incorrect information (e.g. from defective torque sensors) can be interpreted as external forces. This can result in unpredictable motions of the robot.



CAUTION

If the robot is manually guided, an EMERGENCY STOP device must be installed. It must always be within reach of the operator.

Precondition

- The smartPAD is connected to the robot controller.
- Hand guiding device with safety-oriented enabling device (enabling switch) is present and configured.
- The tool with the desired TCP is set in the jogging options.
- No robot application is selected or the robot application has one of the following states:
 - **Selected**
 - **Motion paused**
 - **Error**
- The **Frames** view is open.
- The frames to be taught have been created.
- Manual guidance is allowed in the set operating mode.

Procedure

1. Press and hold down the enabling switch on the hand guiding device.
2. Guide the TCP to the desired position.
3. Once the position has been reached, release the enabling switch.
4. In the **Frames** view, select the frame whose position is to be taught.
5. Press **Touchup** to apply the current TCP coordinates to the selected frame.
The coordinates and redundancy information of the taught point are displayed in the **Apply touchup data** dialog (>>> *Fig. 6-21*).
6. Press **Apply** to save the new values.

6.18.5 Manually addressing frames

Description

Taught frames can be manually addressed with a PTP or LIN motion. In a PTP motion, the frame is approached by the quickest route, whereas in a LIN motion it is approached on a predictable path.

When a frame is being addressed, a warning message is displayed in the following cases:

- The selected tool does not correspond to the tool with which the frame was taught.
- The selected TCP does not correspond to the TCP with which the frame was taught.
- The transformation of the TCP frame has been modified.

If the frame can still be reached, it is possible to move to it.

Precondition

- The frame has been taught.
- The frame can be addressed with the selected TCP.
- Operating mode T1

Procedure

1. Select the desired frame in the **Frames** view.
2. Select the desired jogging type in the **Jogging type** window.
3. Press and hold down the enabling switch.
4. Press the Start key and hold it down until the frame is reached.



If the selected working point is already at the end position or if the frame cannot be reached with the current settings, the robot will not execute any motion.

6.19 Program execution

6.19.1 Selecting a robot application

Procedure

- Select the desired robot application in the navigation bar under **Applications**.

The Applications view opens and the robot application goes into the **Selected** state.

Description

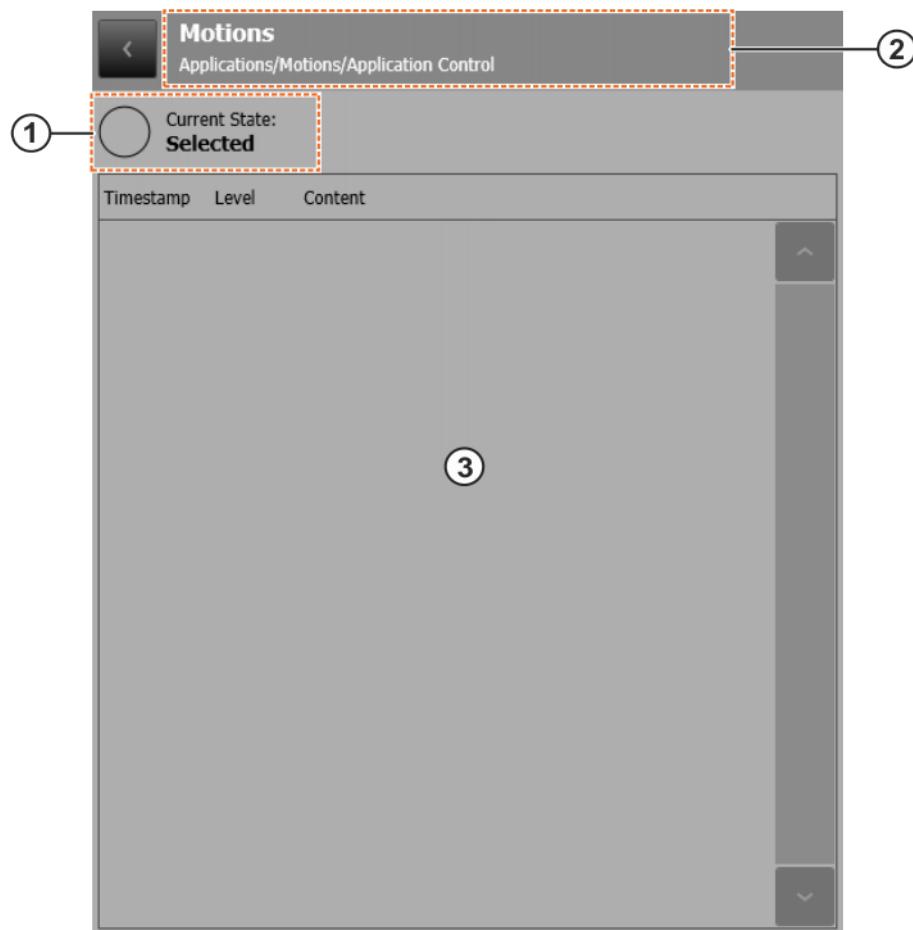


Fig. 6-22: Applications view – robot application selected

Item	Description
1	Current status of the robot application The status is displayed as text and as an icon. (>>> <i>"Status display" Page 116</i>)
2	Display of robot application The name of the selected robot application is displayed, here Motions .
3	Message window Error messages and user messages programmed in the robot application are displayed here.

Status display

The robot application can have the following states:

Icon	State	Description
	Selected	The application is selected.
	Start	The application is initialized.

Icon	State	Description
	Running	The application is executed.
	Motion paused	The application is paused. If the application is paused using the smart-PAD, for example by pressing the STOP key, only motion execution is stopped. Other commands, e.g. switching of outputs, are executed in the Motion paused state until a synchronous motion command is reached.
	Error	An error occurred while the application was running.
	Repositioning	The robot is repositioned. The application is paused because the robot has left the path.
	Stopping	The application is reset to the start of the program and goes into the Selected state.

Start key

An icon on the side panel of the smartHMI indicates the function that can be executed using the Start key.

Icon	Description
	Start application. A selected application can be started or a paused application can be continued.
	Reposition robot. If the robot has left the path, it must be repositioned in order to continue the application.

STOP key

An icon on the side panel of the smartHMI indicates the function that can be executed using the STOP key.

Icon	Description
	Pause application. A running application can be paused in Automatic mode.



If a robot application is paused, the robot can be jogged. The tool and TCP currently used in the paused application are not automatically set as the tool and TCP for Cartesian jogging.
(>>> [6.16.1 "Jogging options" window Page 103](#))

6.19.2 Setting the program run mode

Precondition

- No robot application is selected or the robot application has one of the following states:
 - Selected**

- Motion paused
- Error
- T1 or T2 mode

Procedure

1. Open the **Jogging type** window.
 2. Set the desired program run mode using the button under **Debug options**.
 - Check box not active: Program execution in standard mode
 - Check box active: Program execution in Step mode
- (>> [6.19.2.1 "Program run modes" Page 118](#))

6.19.2.1 Program run modes

Button	Description
	<p>Standard mode</p> <p>The program is executed through to the end without stopping.</p>
	<p>Step mode</p> <p>The program is executed with a stop after each motion command. The Start key must be pressed again for each motion command.</p> <ul style="list-style-type: none"> • The end point of an approximated motion is not approximated but rather addressed with exact positioning. <p>Exception: Approximated motions which were sent to the robot controller asynchronously before Step mode was activated and which are waiting there to be executed will stop at the approximate positioning point. For these motions, the approximate positioning arc will be executed when the program is resumed.</p> <ul style="list-style-type: none"> • In a spline motion, the entire spline block is executed as one motion and then stopped. • In a MotionBatch, the entire batch is not executed but rather exact positioning is carried out after each individual motion of the batch.

 The program run mode can also be set and requested in the source code of the application. (>> [15.17 "Changing and requesting the program run mode" Page 437](#))

6.19.3 Setting the manual override

Description

The manual override determines the velocity of the robot during program execution.

The manual override is specified as a percentage of the programmed velocity. In T1 mode, the maximum velocity is 250 mm/s, irrespective of the override that is set.

If no application override set by the application is active, the manual override corresponds to the effective program override with which the robot actually moves.

If an application override set by the application is active, the effective program override is calculated as follows:

$$\text{Effective program override} = \text{manual override} \cdot \text{application override}$$

Precondition

- Robot application has been selected.

Procedure

1. Touch the **Override** button. The **Override** window is opened.
(>>> [6.14 "Override" window Page 100](#))
 2. Activate the **Set manual override** option if it is not already active.
- | Option | Description |
|--------|---|
| | Set manual override option activated |
3. Set the desired manual override. It can be set using either the plus/minus keys or by means of the slider.
 - Plus/minus keys: The override can be set in steps to the following values: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%, 0%.
 - Slider: The override can be adjusted in 1% steps.
 4. Touch the **Override** button or an area outside the window to close the window.

Alternative procedure

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

The value can be set in the following steps: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%.

6.19.4 Starting a robot application forwards (manually)

Precondition

- Robot application has been selected.
- T1 or T2 mode

Procedure

1. Select the program run mode.
2. Press and hold down the enabling switch.
3. Press Start key and hold it down. The robot application is executed.

To pause a robot application that has been started manually, release the Start key. If the robot application is paused, it can be reset.

6.19.5 Starting a robot application forwards (automatically)

Precondition

- Robot application has been selected.

- Automatic mode
- The project is not controlled externally.

Procedure

- Press the Start key. The robot application is executed.

To pause a robot application that has been started in Automatic mode, press the STOP key. If the robot application is paused, it can be reset.

6.19.6 Resetting a robot application

Description

In order to restart a paused robot application from the beginning, it must be reset. On resetting, the robot application is reset to the start of the program and goes into the **Selected** state.

The button for resetting the application is available under **Applications** in the navigation bar:

Button	Description
	Reset button The button is only active when the robot application is paused.

Precondition

- Robot application is paused.

Alternative procedure

- Select the **Reset** button in the navigation bar under **Applications**.

6.19.7 Repositioning the robot after leaving the path

Description

The following events can cause the robot to leave its planned path:

- Triggering of a non-path-maintaining stop
- Jogging during a paused application

The robot can be repositioned using the Start key. Repositioning means that the robot is returned to the Cartesian position at which it left the path. The application can then be resumed from there.

Characteristics of the motion which is used to return to the path:

- A PTP motion is executed.
The path used to return to the path is different than that taken when leaving the path.
- The robot is moved at 20% of the maximum possible axis velocity and the effective program override (effective program override = manual override application override).



The currently set jog override is irrelevant for repositioning.

- The robot is moved with the load data which were set when the application was interrupted.

- The robot is moved with the controller mode which was set when the application was interrupted.
- Additional forces or force oscillations overlaid by an impedance controller are withdrawn during repositioning.

NOTICE

Repositioning a robot under impedance control may result in unexpected robot motions. The robot is always repositioned to the command position; this means that, in the case of a robot under impedance control, the actual position after repositioning does not necessarily correspond to the actual position at which it left the path. This can lead to unexpectedly high forces in contact situations.

This behavior can be avoided by moving the impedance-controlled robot manually, prior to repositioning, to a position as close as possible to the position at which it left the path.

NOTICE

Repositioning may only be carried out if there is no risk of a collision while it is returning to the path. If this is not assured, first move the robot into a suitable position from which it can be safely repositioned.

Procedure

- In “T1” or “T2” mode: press and hold down the enabling switch.
- Press Start key and hold it down. The robot returns to the path.

6.19.8 Starting/stopping a background application manually

Background applications can be manually stopped and restarted via the smartPAD at any time without the need to press an enabling switch.

**CAUTION**

Background applications can be used to control and monitor peripheral devices, e.g. grippers. Background applications also switch outputs if no robot application is currently being executed or if the robot application is paused due to an EMERGENCY STOP or missing enabling signal. This can result in a gripper closing or opening, even though the EMERGENCY STOP device has been actuated.

If work is being carried out on an operational robot or there are persons in the danger zone, additional safety measures must be taken.

6.19.8.1 Stopping a background application manually**Precondition**

- Background application is running.

Procedure

- In the navigation bar under **Applications** touch the button with the background application to be stopped.

Buttons

The button of a stoppable background application shows the Stop icon. The status indicator is green.

Icon	Status	Description
		Background application is running.

6.19.8.2 Starting a background application manually

Precondition

- Background application has been stopped or has finished.

Procedure

- In the navigation bar under **Applications** touch the button with the background application to be started.

Buttons

The button of a startable background application shows the Start icon. The status indicator can be gray or red.

Icon	Status	Description
		Background application has been stopped or has finished.
		Background has terminated with an error.

6.20 Display functions

6.20.1 Displaying the end frame of the motion currently being executed

Description

If a frame from the frame tree is addressed in an application, this is indicated in the **Frames** view. If the end frame of the motion currently being executed is located at the displayed hierarchy level, the frame name is marked with an arrow icon (3 arrowheads):

Start	X 586.45	Y 0.02	Z 555.34
	A 180.00	B 47.24	C 179.99

Fig. 6-23: The arrow icon marks the current end frame

If the end frame is located hierarchically below a displayed frame, the **Display child frames** button is marked with an additional arrow icon (3 arrowheads):

Triangle		X 427.07	Y -401.86	Z 555.35	
		A 136.74	B 47.22	C 179.99	

Fig. 6-24: The button switches to the current end frame

You can switch directly to the current end frame using the magnifying glass button in the upper right-hand area of the **Frames** view:

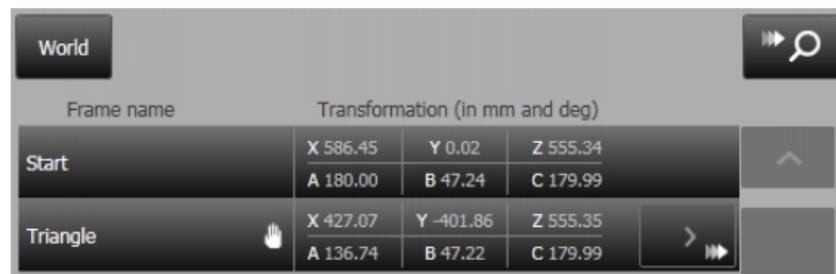


Fig. 6-25: The magnifying glass button switches directly to the current end frame

The magnifying glass button is inactive if no frame is being addressed.

Precondition

- Robot application has been selected.
- Application status **Running** or **Motion paused**
- The motion uses an end frame created in the application data.

Procedure

1. Select **Frames** at the Station level. The **Frames** view opens.
2. Switch to the end frame using the **Display child frames** button or the magnifying glass button.

6.20.2 Displaying the axis-specific actual position

Procedure

- Select the **Axis position** tile at the Robot level.

Description

The current position of axes A1 to A7 is displayed. In addition, the range within which each axis can be moved (limitation by end stops) is indicated by a white bar.

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

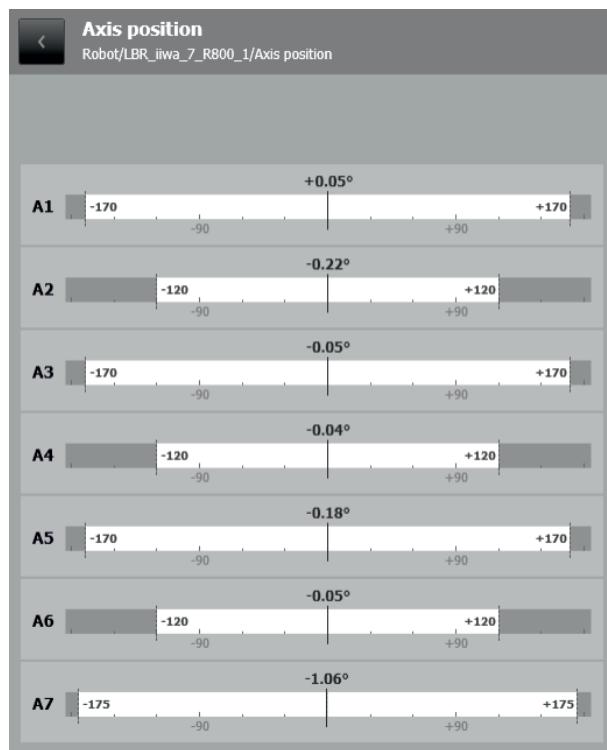


Fig. 6-26: Axis-specific actual position

6.20.3 Displaying the Cartesian actual position

Procedure

1. Select the **Cartesian position** tile at the Robot level.
2. Set the TCP and base in the **Jogging options** window.

Description

The Cartesian actual position of the selected TCP is displayed. The values refer to the base set in the jogging options.

The display contains the following data:

- Current position (X, Y, Z)
- Current orientation (A, B, C)
- Current redundancy information: Status, Turn, redundancy angle (E1)
- Current tool, TCP and base

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

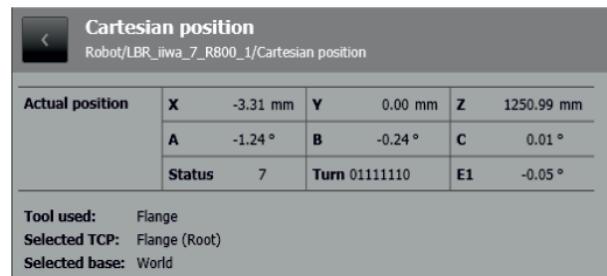


Fig. 6-27: Cartesian actual position

6.20.4 Displaying axis-specific torques

Procedure

- Select the **Axis torques** tile at the Robot level.

Description

The current torque values for axes A1 to A7 are displayed. In addition, the sensor measuring range for each axis is displayed (white bar).

If the maximum permissible torque on a joint is exceeded, the dark gray area of the bar for the axis in question turns orange. Only the violated area is indicated in color (either the negative or positive part).



The refresh rate of the displayed values is limited. Briefly occurring peak values are therefore not displayed under certain circumstances.

The display contains the following data:

- Current absolute torques
- Current external torques



The external torques are only displayed correctly if the correct tool has been specified.

- Current tool

The axis-specific torques can also be displayed while the robot is moving.

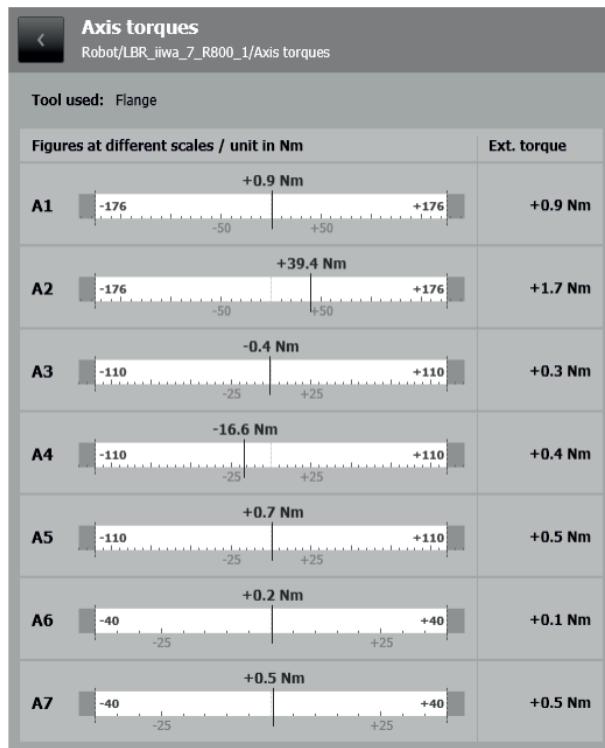


Fig. 6-28: Axis-specific torques

6.20.5 Displaying an I/O group and changing the value of an output

Precondition

- To change an output: Operating mode T1, T2 or CRR



The outputs can be changed irrespective of the safety controller status, for example even if an EMERGENCY STOP is pressed.

Procedure

1. In the navigation bar, select the desired I/O group from **I/O groups**. The inputs/outputs of the selected group are displayed.
2. Select the output to be changed.
3. An input box is displayed for numeric outputs. Enter the desired value.
4. Press and hold down the enabling switch. Change the value of the input with the appropriate button.

Description

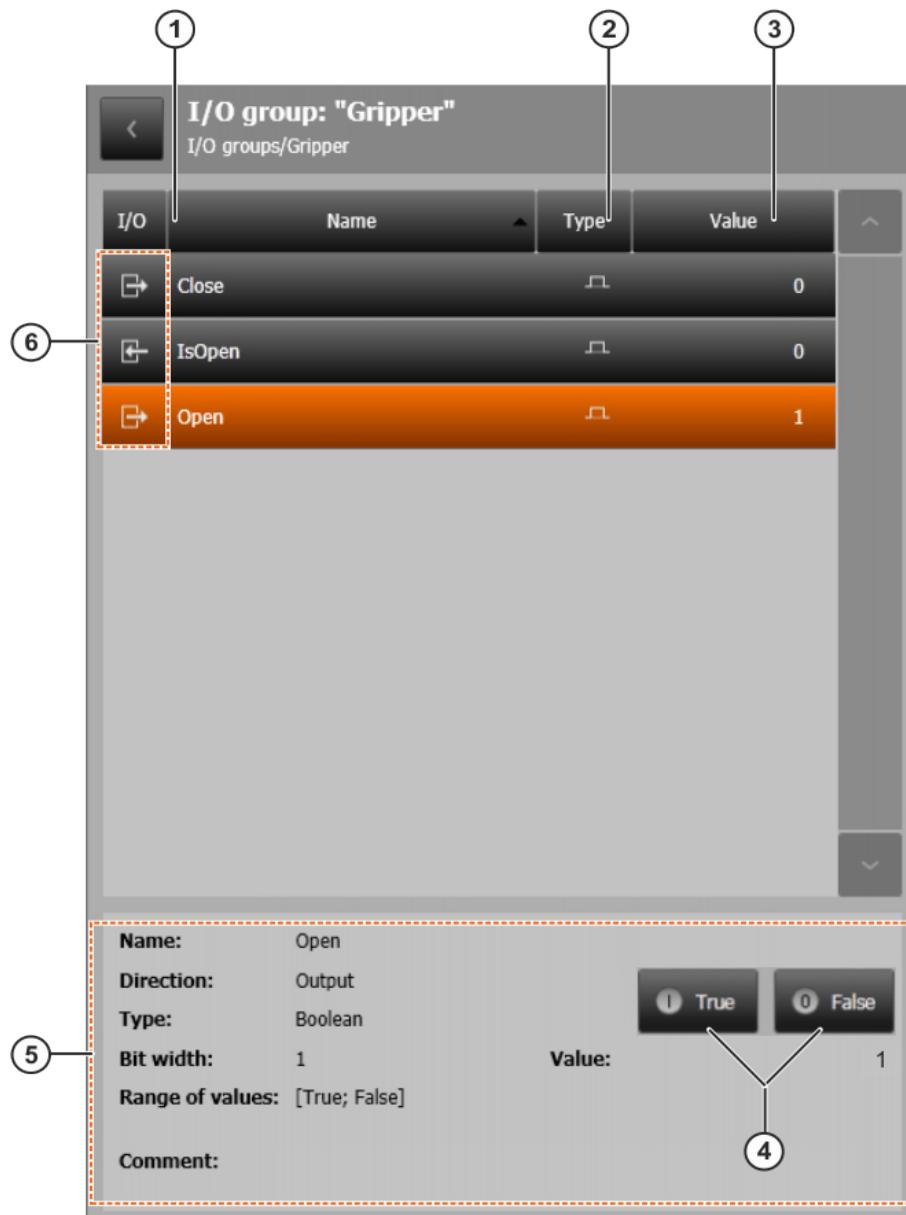


Fig. 6-29: Inputs/outputs of an I/O group

Item	Description
1	Name of the input/output
2	Type of input/output

Item	Description
3	Value of the input/output The value is displayed as a decimal number.
4	Buttons for changing outputs If an output is selected, its value can be changed. Precondition: The enabling switch is pressed. The buttons available depend on the output type.
5	Signal properties The properties and the current value of the selected input or output are displayed.
6	Signal direction The icons indicate whether the signal is an input or an output.

The following buttons are available depending on the type of the selected output:

Button	Description
True	Buttons for changing Boolean outputs
False	Sets the selected Boolean outputs to the value True (1) or False (0).
Set	Button for changing numeric outputs Sets the selected numeric output to the entered value.

Signal direction

The following icons indicate the direction of a signal:

Icon	Description
	Icon for an output
	Icon for an input

I/O types

The following icons indicate the type of input/output:

Icon	Description
	Icon for an analog signal
	Icon for a binary signal
	Icon for a signed digital signal
	Icon for an unsigned digital signal

6.20.6 Displaying information about the robot and robot controller

Procedure

- Select the **Information** tile at the Station level.

Description

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

The following information is displayed under the individual nodes:

Node	Description
Station	Station information <ul style="list-style-type: none">• Software version: Version of the installed System Software• Station server IP: IP address of the robot controller• Serial number of controller: Serial number of the robot controller
User interface	Information about the smartHMI <ul style="list-style-type: none">• Connection IP• Connection state
<Robot name>/Type plate	Robot information <ul style="list-style-type: none">• Serial number: Serial number of the connected robot• Connected robot: Type of the connected robot• Installed robot: Robot type specified in the station configuration of Sunrise.Workbench• Operating time [h] The operating hours meter is running as long as the drives are switched on.

6.21 Backup Manager

Overview

Once the Backup Manager has been installed on the robot controller, a tile for the Backup Manager is available on the smartHMI.

The Backup Manager makes it possible to back up and restore robot controller data manually. Automatic backup of data at a predefined interval can also be preconfigured in the station configuration.

The following data are backed up and restored:

- Project data
- Catalogs of the installed software
- User-specific files (directory: C:\KRC\UserData)

The target directory for backups and the source directory for restorations is preconfigured:

- Directory D:\ProjectBackup on the robot controller
- OR: Shared network directory



If the target directory for backups is on a network drive, it is advisable to perform a connection test during start-up. Test by carrying out a manual backup. If this fails, e.g. because the target directory in the network is not accessible due to a defective configuration, this is indicated in an error message.

User privileges

As standard, no special authorization is required for backing up and restoring data. If the user group “Expert” is installed, the default user may no longer execute these functions. The user must be logged on as “Expert” or higher.

Function	Operator	Expert	Safety maintenance
Backing up data manually	✗	✓	✓
Restoring data manually	✗	✓	✓

6.21.1 “Backup Manager” view

Precondition

- Backup Manager is installed.

Procedure

To open the view:

- Select the robot controller tile at the Station level and then the **Backup Manager** tile.

Description

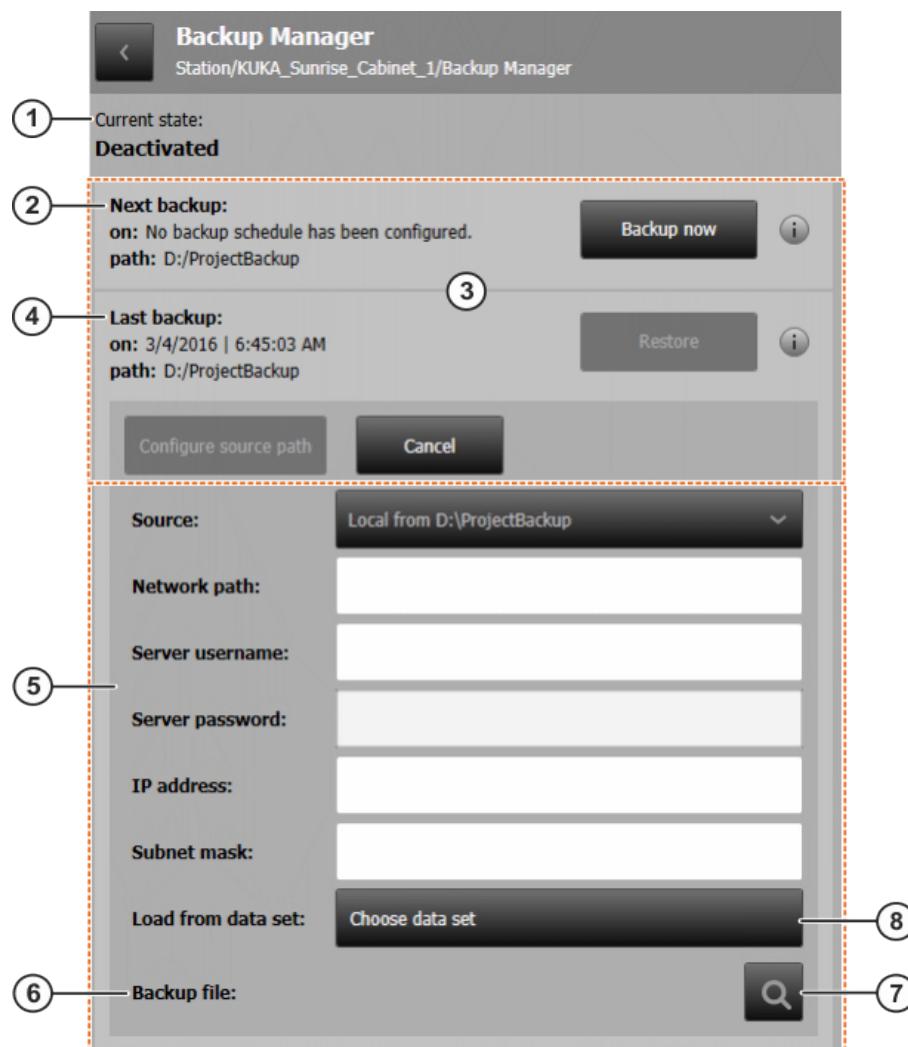


Fig. 6-30: Backup Manager view

Item	Description
1	Status indicator of the backup <ul style="list-style-type: none"> • Deactivated: automatic backup is not configured • Ready: automatic backup is activated • Running: a backup is in progress (started manually or automatically)
2	Information about the next automatic backup (if activated) <ul style="list-style-type: none"> • Date and time • Target directory

Item	Description
3	<p>“Manual backup/restoration” area</p> <p>When the view is opened for the first time, only this area and the status indicator are displayed. This is the default view.</p> <p>The area contains the following buttons:</p> <ul style="list-style-type: none"> • Backup now (>>> 6.21.2 "Backing up data manually" Page 132) • Restore The button cannot be activated until the backup copy that is to be restored has been selected using the magnifying glass button. (>>> 6.21.3 "Restoring data manually" Page 132) • Configure source path Displays the “Configure source path” area. After this the button is inactive. • Cancel Hides the “Configure source path” area again. The button is inactive in the default view.
4	<p>Information about the most recent successful backup</p> <ul style="list-style-type: none"> • Date and time • Target directory
5	<p>“Configure source path” area</p> <p>The source directory from which restoration is to be carried out can be defined here. As standard, the source directory defined in the station configuration is preset.</p> <p>The following source directories are available for selection:</p> <ul style="list-style-type: none"> • Local from D:\ProjectBackup: the source directory is the directory D:\ProjectBackup on the robot controller • Network: the source directory is located on a network drive The network path to the source directory can be configured. (>>> 6.21.4 "Configuring the network path for restoration" Page 132)
6	<p>Information about the backup copy selected for restoration</p> <ul style="list-style-type: none"> • Project name • Date and time of the backup
7	<p>Magnifying glass button</p> <p>Opens a dialog in which the backup copy to be restored can be selected. The dialog displays all backup copies contained in the configured source directory.</p>
8	<p>Load data set button</p> <p>Opens a dialog which can be used to select and apply ready-made restoration configurations.</p> <p>The button is only active if the file for restoration configurations is configured in the station configuration and the file is saved under the configured path on the robot controller.</p>

6.21.2 Backing up data manually

Description

The backup copies are saved in the target directory in the following folder structure:

- *IP address_Project name\BACKUP_No.*

Element	Description
<i>IP address</i>	IP address of the robot controller
<i>Project name</i>	Name of the project installed on the robot controller
No.	Number of the backup copy The BACKUP folder with the highest number always contains the most recent backup copy.

Precondition

- No data backup is in progress.

Procedure

- Touch the **Backup now** button in the **Backup Manager** view. The backup is carried out.

6.21.3 Restoring data manually

Precondition

- No application is selected.
- Robot is not being jogged or manually guided.
- No data backup is in progress.

Procedure

1. Touch the **Configure source path** button in the **Backup Manager** view.
2. If not already preset, select the source from which restoration is to be carried out. If required, configure the desired network path for restoration.
(>>> [6.21.4 "Configuring the network path for restoration" Page 132](#))
3. Touch the magnifying glass button. A dialog is opened. The backup copies available in the specified source directory are listed.
4. Select the desired backup copy and touch the **Select** button. The dialog is closed and information about the selected backup copy is displayed.
5. Touch the **Restore** button. Restoration commences.
A progress bar indicates how far the process is. Following an automatic reboot of the robot controller, restoration is completed.

6.21.4 Configuring the network path for restoration

Description

The network parameters can be entered manually or loaded from a pre-configured data set:

Parameter	Description
Network path	Network path to source directory, e.g. \\192.168.40.171\Backup\Restore
Server user name	User name for the network path The parameter is only relevant if authentication is required for network access.
Server password	Password for the network path The parameter is only relevant if authentication is required for network access.
IP address	IP address of the robot controller to be restored
Subnet Mask	Subnet mask in which the IP address of the robot controller is located



The IP addresses of the robot controller and the server must be located in the same range. IP address and subnet mask of the robot controller to be restored must be selected accordingly.

Precondition

For loading from a data set:

- The file for restoration configurations is configured in the station configuration.
- The file is saved under the configured path on the robot controller.

Procedure

1. Touch the **Configure source path** button in the **Backup Manager** view.
2. Select **Network** as the source if this is not already preset.
3. Enter network parameters or load them from a data set.

To load a data set, proceed as follows:

- a. Touch the **Load data set** button. The **Available restoration configurations** dialog opens. All available configurations are listed. Every entry contains the name of the robot controller that is to be restored. The network path is indicated below this.
- b. The selection can be reduced by filtering the entries by the name of the robot controller. To do so, enter the name or part of the name in the dialog. e.g. *Controller.
- c. Select the desired entry and touch the **Import** button.

7 Start-up and recommissioning

The robot controller is supplied with an operational version of the System Software. Therefore, no installation is required during initial start-up.

Installation becomes necessary, for example, if the station configuration changes.

(>>> *10 "Station configuration and installation" Page 193*)

7.1 Switching the robot controller on/off

NOTICE

If an application is still running when the robot controller is switched off, active motions are stopped. This can result in the robot being damaged. For this reason, the robot controller must only be switched off when no more applications are running and the robot is stationary.

7.1.1 Switching on the robot controller

Description

When the robot controller is switched on, the system software starts automatically.

The robot controller is ready for operation when the status indicator for the boot state of the robot controller lights up green:

- **Boot state** tile at the Station level under the robot controller tile.

Precondition

- The smartPAD is connected.

Procedure

- Turn the main switch on the robot controller to the “I” position.

7.1.2 Switching the robot controller off

Precondition

- No application is running on the robot controller.
- The robot is at a standstill.

Procedure

- Turn the main switch on the robot controller to the “0” position.

7.2 Update of the smartPAD software



The firmware update described in the following section only applies for operation with the smartPAD.

Description

The smartPAD software version is checked automatically in the following cases:

- Reboot of robot controller

- Connection of smartPAD to a running robot controller

If the version check reveals a conflict between the smartPAD software and the System Software on the robot controller, an update of the smartPAD software is required.

Sequence

1. The update starts automatically in T1, T2 and CRR modes.
2. No update is possible in Automatic mode. To start the update, switch to T1 or T2 mode.
3. No user input may be entered during the smartPAD update.

NOTICE

Material damage due to interruption of the software update

If the software update is interrupted, this may result in damage to the smartPAD.

- Do not disconnect the smartPAD from the robot controller during the update.
- Do not disconnect the robot controller from the power supply during the update.

4. In the case of a successful update, the smartPAD is automatically rebooted.
5. After the smartPAD has rebooted, the smartHMI is not yet displayed completely. Reboot the robot controller to be able to use the smartPAD again.



The firmware update for operation with the smartPAD-2 may only be carried out by KUKA Service.

This update is only possible from KUKA Sunrise.OS Med 1.15.3 onwards.

7.3 Performing a PDS firmware update

Description

If the robot controller is rebooted or the drive bus connection restored, the system checks for every connected PDS whether the current PDS firmware version matches the firmware version on the robot controller. If the firmware version of at least one of the PDSs is older than the version on the robot controller, a PDS firmware update must be performed.

The following error message is displayed under the *Device state* tile:

Firmware update is required. Select "Diagnosis" > "PDS firmware update" in the main menu in order to update the firmware.

Procedure

- In the main menu, select **Diagnosis > PDS firmware update**.

The update is started and a blocking dialog is displayed. No user input may be entered during the smartPAD update.

NOTICE

The update may take up to 5 hours and must not be interrupted:

- Do not disconnect the robot from the robot controller during the update.
- Do not disconnect the robot controller from the power supply during the update.

If the update is interrupted, it is possible that the robot controller may enter the error state with the result that the robot can no longer be moved. This fault can only be rectified by KUKA Service.

Once the update has been successfully completed, the dialog is closed.

7.4 Position mastering

During position mastering, a defined mechanical robot axis position is assigned to a motor angle. Only with a mastered robot is it possible for taught positions to be addressed with high repeatability. An unmastered robot can only be moved manually (axis-specific jogging in T1 or CRR mode).

7.4.1 Performing position mastering

Description

An LBR has a Hall effect mastering sensor in every axis. The mastering position of the axis (zero position) is located in the center of a defined series of magnets. It is automatically detected by the mastering sensor when it passes over the series of magnets during a rotation of the axis.

Before the actual mastering takes place, an automatic search run is performed in order to find a defined premastering position.

If the search run is successful, the axis is moved into the premastering position. The axis is then moved in such a way that the mastering sensor passes over the series of magnets. The motor position at the moment when the mastering position of the axis is detected is saved as the zero position of the motor.

If the search run or the mastering fails, the process is aborted and the robot stops.



Following remastering, the position of an axis is not referenced. This event does not lead to a violation of the safe position-based safety functions. The robot can be moved, but the safety integrity of the safety functions is no longer assured.

If safe position-based safety functions are used, position referencing must thus be carried out for every remastered axis. Otherwise, the safety functions may behave differently from how they were configured, creating additional hazards in the system.

Precondition

- Operating mode T1 or CRR



The repeatability and reproducibility of mastering are only guaranteed if the procedure is always identical. The following rules must be observed during mastering:

- When one axis is being mastered, all axes should be in the vertical stretch position. If this is not possible, mastering must always be carried out in the same axis position.
- For robots without absolutely accurate calibration:
Always carry out the mastering of an axis with the same load. The axes not being mastered can be in any position.



The mastering velocity is independent of the set jog override.

Procedure

1. Select the **Mastering** tile at the Robot level. The **Mastering** view opens.
2. Press and hold down the enabling switch.
3. Touch the **Master** button for the unmastered axis.
First of all, the premastering position is located by means of a search run. The mastering run is then performed. Once mastering has been carried out successfully, the axis moves to the calculated mastering position (zero position).
4. Repeat Steps 2 and 3 to master further axes.

7.4.2 Manually unmastering axes

Description

The saved mastering position of an axis can be deleted. This unmasters the axis. No motion is executed during unmastering.

Precondition

- Operating mode T1

Procedure

1. Select the **Mastering** tile at the Robot level. The **Mastering** view opens.
2. Touch the **Unmaster** button for the mastered axis. The axis is unmastered.

7.5 Calibration

7.5.1 Tool calibration

Description

During tool calibration, a Cartesian coordinate system (tool coordinate system) is assigned to the tool mounted on the mounting flange.

The tool coordinate system has its origin at a point defined by the system integrator. This is called the TCP (Tool Center Point). The TCP is generally situated at the working point of the tool. A tool can have multiple TCPs.

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.

Description

During tool calibration, a Cartesian coordinate system (tool coordinate system) is assigned to the tool mounted on the mounting flange.

The tool coordinate system has its origin at a point defined by the system integrator. This is called the TCP (Tool Center Point). The TCP is generally situated at the working point of the tool. A tool can have multiple TCPs.

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.

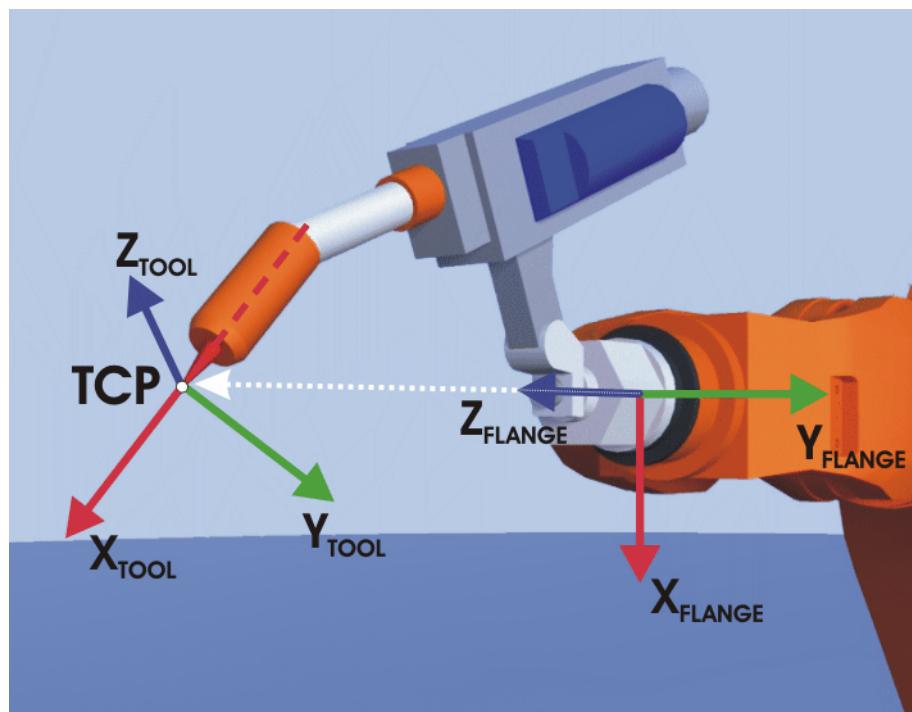


Fig. 7-1: TCP calibration principle

Overview

Tool calibration consists of 2 steps:

Step	Description
1	<p>Define the origin of the tool coordinate system</p> <p>The following methods are available:</p> <ul style="list-style-type: none"> • XYZ 4-point (>>> 7.5.1.1 "TCP calibration: XYZ 4-point method" Page 140)
2	<p>Define the orientation of the tool coordinate system</p> <p>The following methods are available:</p> <ul style="list-style-type: none"> • ABC 2-point (>>> 7.5.1.2 "Defining the orientation: ABC 2-point method" Page 142) This method is not available for safety-oriented tools. • ABC world (>>> 7.5.1.3 "Defining the orientation: ABC world method" Page 144)



Once calibration has been completed, it is advisable to synchronize the project immediately in order to update the calibration data in the corresponding project in Sunrise.Workbench.

7.5.1.1 TCP calibration: XYZ 4-point method

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 with which the reference point is addressed must maintain a certain minimum distance between one another. If the points are too close to one another, the position data cannot be saved. A corresponding error message is generated.

The quality of the calibration can be assessed by means of the translational calculation error which is determined during calibration. If this error exceeds a defined limit value, it is advisable to calibrate the TCP once more.

The minimum distance and the maximum calculation error can be modified in Sunrise.Workbench. (>>> [10.4.4 "Configuration parameters for calibration" Page 198](#))

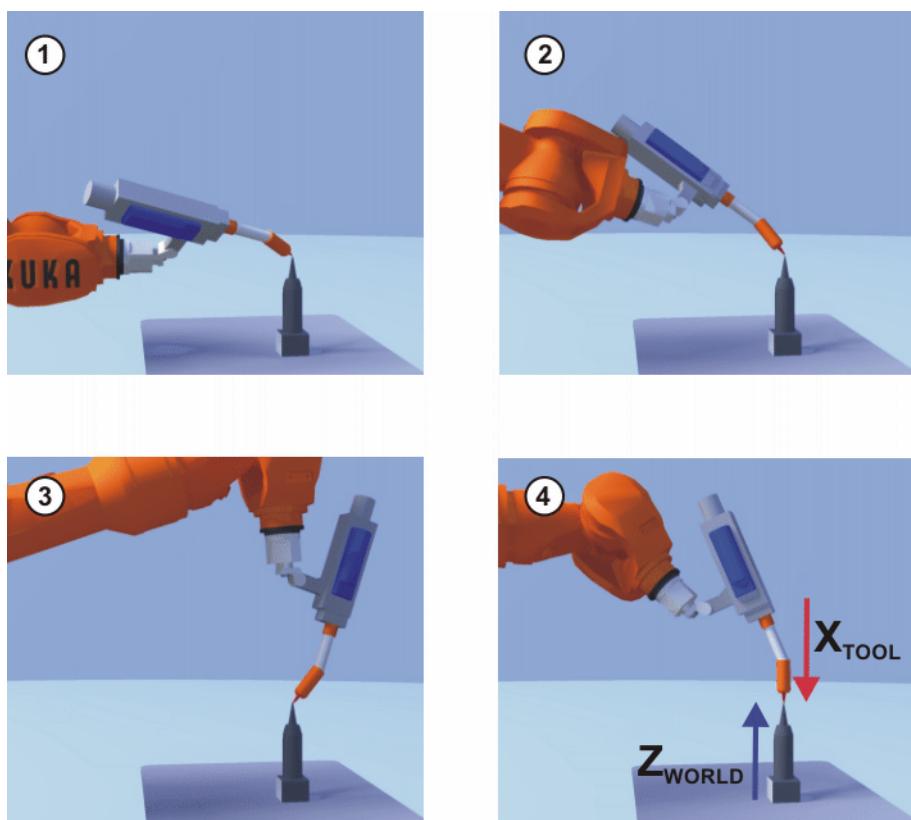


Fig. 7-2: XYZ 4-point method

Precondition

- The tool to be calibrated is mounted on the mounting flange.
- The tool to be calibrated and the frame used as the TCP have been created in the object templates of the project and transferred to the robot controller by means of synchronization.
- T1 mode

Procedure

1. Select the **Calibration> Tool calibration** tile at the Robot level. The **Tool calibration** view opens.
2. Select the tool to be calibrated and the corresponding TCP.
3. Select the **TCP calibration(XYZ 4-point)** method. The measuring points of the method are displayed as buttons:
 - **Measurement point 1 ... Measurement point 4**
 In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to any reference point. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
5. Move the TCP to the reference point from a different direction. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
6. Repeat step 5 two more times.
7. Press **Determine tool data**. The calibration data and the calculation error are displayed in the **Apply tool data** dialog.
8. If the calculation error exceeds the maximum permissible value, a warning is displayed. Press **Cancel** and recalibrate the TCP.

9. If the calculation error is below the configured limit, press **Apply** to save the calibration data.
10. Either close the Calibration view or define the orientation of the tool coordinate system with the ABC 2-point or ABC World method.
(>>> [7.5.1.2 "Defining the orientation: ABC 2-point method" Page 142](#))
(>>> [7.5.1.3 "Defining the orientation: ABC world method" Page 144](#))
11. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.5.1.2 Defining the orientation: ABC 2-point method

Description

The robot controller is notified of the axes of the tool coordinate system by addressing a point on the X axis and a point in the XY plane.

The points must maintain a defined minimum distance from one another. If the points are too close to one another, the position data cannot be saved. A corresponding error message is generated.

The minimum distance can be modified in Sunrise.Workbench.

(>>> [10.4.4 "Configuration parameters for calibration" Page 198](#))

This method is used for tools with edges and corners which can be used for orientation purposes. Furthermore, it is used if it is necessary to define the axis directions with particular precision.

This method is not available for safety-oriented tools.

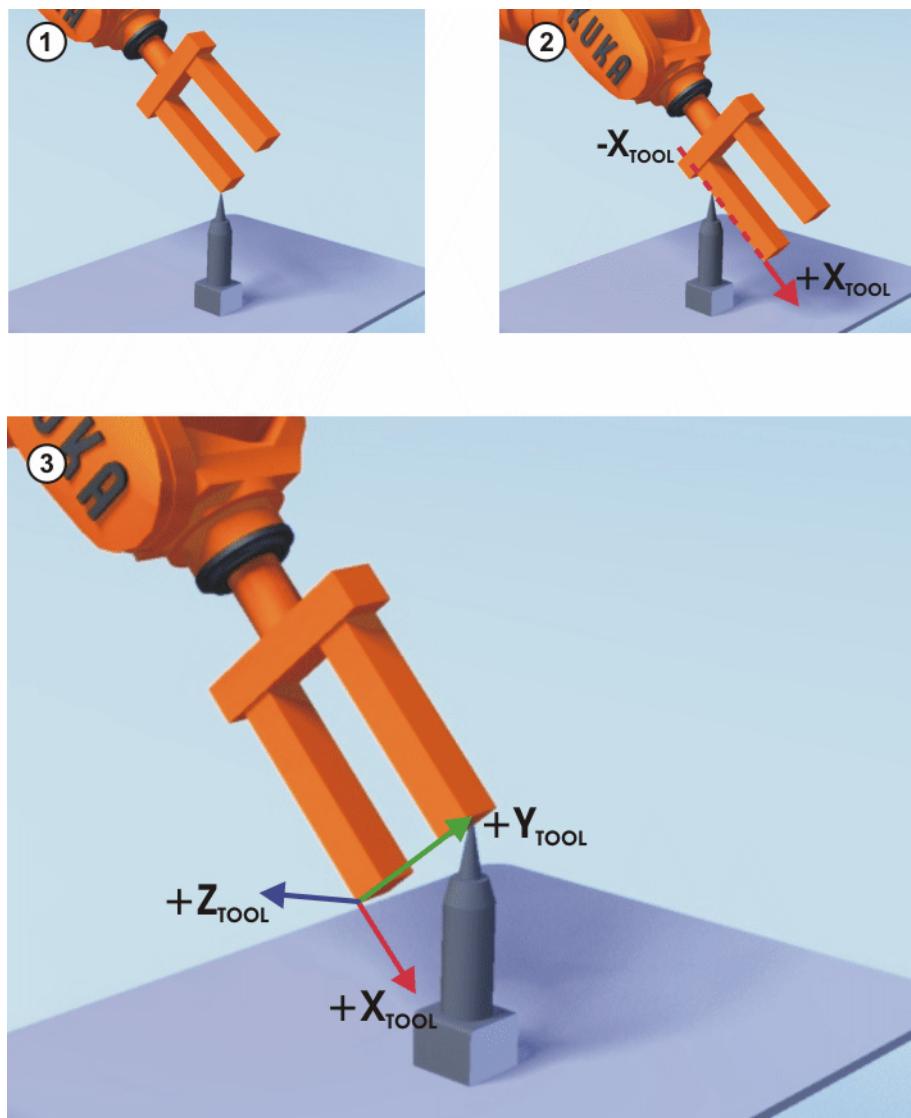


Fig. 7-3: ABC 2-point method

Precondition

- The tool to be calibrated is not a safety-oriented tool.
- The tool to be calibrated is mounted on the mounting flange.
- The TCP of the tool has already been measured.
- T1 mode

Procedure

1. Only if the Calibration view was closed following TCP calibration:
Select the **Calibration> Tool calibration** tile at the Robot level. The **Tool calibration** view opens.
2. Only if the Calibration view was closed following TCP calibration:
Select the mounted tool and the corresponding TCP of the tool.
3. Select the **Defining the orientation (ABC 2-point)** method. The measuring points of the method are displayed as buttons:
 - **TCP**
 - **Negative X axis**
 - **Positive Y value on XY plane**

In order to be able to record a measuring point, it must be selected (button is orange).

4. Move the TCP to any reference point. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
5. 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 **Record calibration point**. The position data are applied and displayed for the selected measuring point.
6. Move the tool so that the reference point in the XY plane has a positive Y value. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
7. Press **Determine tool data**. The calibration data are displayed in the **Apply tool data** dialog.
8. Press **Apply** to save the calibration data.
9. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.5.1.3 Defining the orientation: ABC world method

Description

The system integrator 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 system integrator communicates the tool direction to the robot controller. The default tool direction is the X axis. The orientation of the other axes is defined by the system and cannot be influenced by the system integrator.

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 system integrator communicates the direction of all 3 axes to the robot controller.

This method is used for tools that do not have corners which the system integrator can use for orientation purposes, e.g. rounded tools such as adhesive or welding nozzles.

Precondition

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

Procedure

1. Only if the Calibration view was closed following TCP calibration:
Select the **Calibration> Tool calibration** tile at the Robot level. The **Tool calibration** view opens.
2. Only if the Calibration view was closed following TCP calibration:
Select the mounted tool and the corresponding TCP of the tool.
3. Select the **Defining the orientation(ABC world)** method.
4. Select the **ABC World 5D or ABC world 6D** option.
5. If **ABC World 5D** is selected:

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

If **ABC world 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 $+X_{WORLD}$.

6. Press **Determine tool data**. The calibration data are displayed in the **Apply tool data** dialog.
7. Press **Apply** to save the calibration data.
8. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.5.2 Base calibration: 3-point method

Description

During base calibration, the system integrator assigns a Cartesian coordinate system (base coordinate system) to a frame selected as the base. The base coordinate system has its origin at a point defined by the system integrator.

Advantages of base calibration:

- The TCP can be jogged along the edges of the work surface or work-piece.
- 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 origin and 2 further points of a base are addressed with the 3-point method. These 3 points define the base.

The points must maintain a defined minimum distance from the origin and minimum angles between the straight lines (origin – X axis and origin – XY plane). If the points are too close to one another or if the angle between the straight lines is too small, the position data cannot be saved. A corresponding error message is generated.

The minimum distance and angles can be modified in Sunrise.Workbench.
(>>> [10.4.4 "Configuration parameters for calibration" Page 198](#))

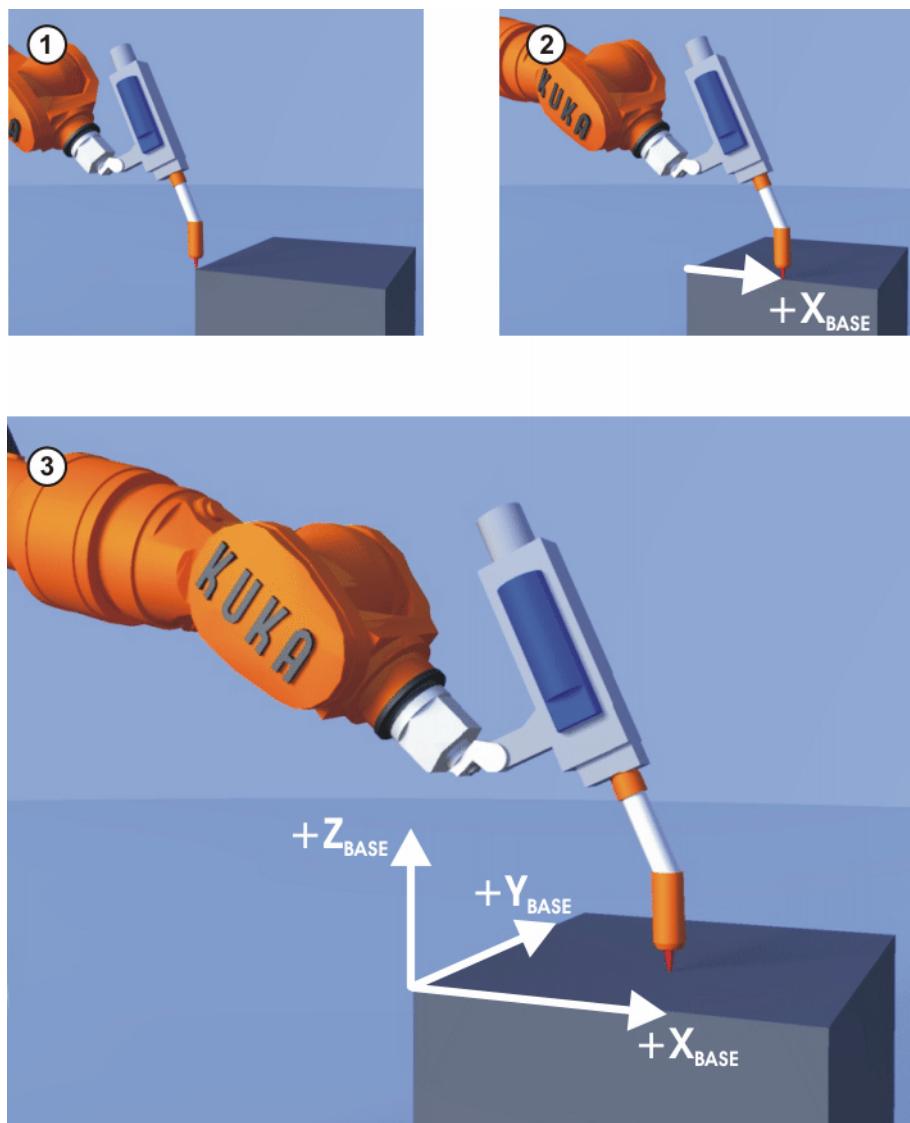


Fig. 7-4: 3-point method

Precondition

- A previously calibrated tool is mounted on the mounting flange.
- The frame to be calibrated has been selected as the base in the application data of the project and transferred to the robot controller by means of synchronization.
- T1 mode

Procedure

1. Select **Calibration > Base calibration** at the Robot level. The **Base calibration** view opens.
2. Select the base to be calibrated.
3. Select the mounted tool and the TCP of the tool with which the measuring points of the base are addressed.

The measuring points of the 3-point method are displayed as buttons:

- **Origin**
- **Positive X axis**
- **Positive Y value on XY plane**

In order to be able to record a measuring point, it must be selected (button is orange).

4. Move the TCP to the origin of the base. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
5. Move the TCP to a point on the positive X axis of the base. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
6. Move the TCP to a point in the XY plane with a positive Y value. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
7. Press **Determine base data**. The calibration data are displayed in the **Apply base data** dialog.
8. Press **Apply** to save the calibration data.
9. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.6 Determining tool load data

Description

During load data determination, the robot performs multiple measurement runs with different orientations of wrist axes A5, A6 and A7. The load data are calculated from the data recorded during the measurement runs.

The mass and the position of the center of mass of the tool mounted on the robot flange can currently be determined. It is also possible to specify the mass and to determine the position of the center of mass on the basis of the mass that is already known.

At the start of load data determination, axis A7 is moved to the zero position and axis A5 is positioned in such a way that axis A6 is aligned perpendicular to the weight. During the measurement runs, axis A6 has to be able to move between -95° and +95°, while axis A7 has to be able to move from 0° to -90°.

The remaining robot axes (A1 to A4) are not moved during load data determination. They remain in the starting position during measurement.

Quality

The quality of the load data determination may be influenced by the following constraints:

- Mass of the tool

Load data determination becomes more reliable as the mass of the tool increases. This is because measurement uncertainties have a greater influence on a smaller mass.



Load data determination cannot yet be used reliably for masses of less than one kilogram.

- Supplementary loads

Supplementary loads mounted on the robot, e.g. dress packages, lead to incorrect load data.

- Start position from which load data determination is started

A suitable start position should be determined first and meet the following criteria:

- Axes A1 to A5 are as far away as possible from singularity positions.

The criterion is relevant if the mass is to be determined during load data determination. If load data determination is only possible in poses for which axes A1 to A5 are close to singularity positions, the mass can be specified. If only the center of mass is to be determined on the basis of the specified mass, the criterion of axis position is irrelevant.

- The suitability of the start position for load data determination in the case of a robot for which automatic load data determination is to be carried out must be checked before the load that is to be determined is mounted on the robot.

A robot pose is suitable as a start position for load data determination if there are only very slight external torques (ideally $< 0.5 \text{ Nm}$) acting on the load-free robot in this position. This can be checked via the display of the external axis torques.

(>>> [6.20.4 "Displaying axis-specific torques" Page 125](#))

If the mass is to be determined during load data determination, all external axis torques are relevant and should be checked where possible in advance for the load-free robot. If only the center of mass is to be determined on the basis of a specified mass, only the external axis torque of axis A6 is relevant.

- Interference torques during the measurement runs
 - During the measurement runs, no interference torques may be applied, e.g. by pulling or pushing the robot.
 - Moving parts, e.g. dress packages, generate interference torques that shift the center of mass during the measurement run.
 - Installation on a mobile or vibrating platform can result in significant interference torques.

The best results are achieved if the installation platform is firmly anchored to the floor. If the robot is fastened to a mobile platform, e.g. an assembly cart with a simple wheel brake and without anchor bolts, the system may start to oscillate due to the motion of the robot.



The application of interference torques during load data determination results in falsified load data.



WARNING

Danger to life and limb due to incorrect loads

Operating a robot with incorrect loads may result in death, severe injuries or damage to property. For example, the braking procedure may take too long with incorrect load data.

- Use only loads that are permissible for the robot.
- Use correct load data.

Safety-oriented tools

For the load data determination for safety-oriented tools, it must be ensured that the modified load data are not automatically transferred to the safety configuration located on the robot controller. (>>> [9.3.10 "Configuring a safety-oriented tool" Page 176](#))

The load data determined for a safety-oriented tool must first be updated in Sunrise.Workbench by means of project synchronization. This changes the safety configuration of the project in Sunrise.Workbench; the project must then be re-transferred to the robot controller by synchronizing the project again.



If a changed safety configuration is activated on the robot controller, the safety maintenance technician must carry out safety acceptance. To avoid spending unnecessary time performing verifications, it is advisable to mark a tool as safety-oriented only when the load data have been correctly entered or determined and have been transferred to Sunrise.Workbench.

Preparation

- Determine the start position from which load data determination is to be started.

Precondition

- The tool is mounted on the mounting flange.
- The tool has been created in the object templates of the project and transferred to the robot controller by means of synchronization.
- The robot is in the desired start position.
- There is a sufficiently large workspace available in the wrist axis range.
- T1, T2 or AUT mode

NOTICE

If parts of the mounted tool project behind the flange plane (in the negative Z direction relative to the flange coordinate system), there is a risk of the tool colliding with the robot during the measurement runs. If the motion of the axes during load data determination is unknown, or if a collision between the tool and the robot cannot be ruled out (e.g. for the initial load data determination for a tool), it is advisable to determine the load data in T1 mode. This does not affect the quality of the measurement results.

Procedure

1. Select the **Load data** tile at the Robot level. The **Load data** view opens.
2. Select the mounted tool from the selection list.
3. If T1 or T2 mode is set, press and hold down the enabling switch until load data determination has been completed.
4. Press **Determining the load data**.
5. If the tool already has a mass, the operator will be asked if the mass is to be redetermined.
 - Select **Use existing mass** if the currently saved mass is to be retained.
 - Select **Redetermine mass** if the mass is to be determined again.



If the currently saved mass is to be retained in the load determination, it must be ensured that the specified mass value is correct. Otherwise, the center of mass cannot be determined accurately.

6. The robot starts the measurement runs and the load data are determined. A progress bar is displayed.



If the motion enable signal is withdrawn during load data determination, e.g. by an EMERGENCY STOP or by releasing the enabling switch (T1, T2), the load data determination is aborted and must be restarted.

Once load data determination has been completed, the determined load data are displayed in the **Apply load data** dialog.

Press **Apply** to save the determined load data.

7. Synchronize the project so that the load data are saved in Sunrise.Workbench.
8. When the load data for a safety-oriented tool have been determined, the safety configuration changes as a result of the project synchronization.
Synchronize the project in order to transfer the changed safety configuration to the robot controller.

Load data view

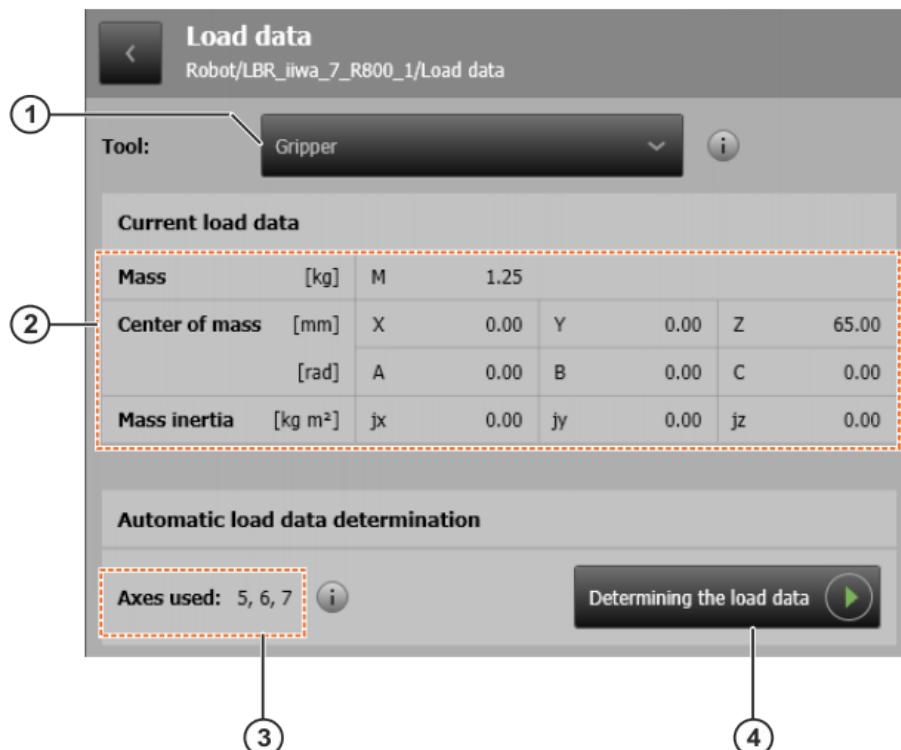


Fig. 7-5: Determining the load data

Item	Description
1	Tool selection list The tools created in the object templates are available for selection here.
2	Load data display Displays the current load data of the selected tool.
3	Display of axes used Displays the axes that are moved for load data determination.
4	Determining the load data button Starts load data determination. The button is only active if a tool has been selected and the motion enable signal has been issued.

8 Brake test

8.1 Special behavior of the brakes of the LBR Med

Description

The brakes of the LBR Med serve as a safety function. The brakes are only closed by the safety controller if it detects a fault.

(>>> [13.3 "Single fault safety" Page 242](#))

Due to this behavior, the brakes are only applied in the following cases:

- The safety controller has triggered a stop reaction (e.g. the E-STOP device on the smartPAD has been pressed)
- During the mastering of axes
- A serious error has occurred

The brakes are not applied in the following cases:

- Application is terminated
- Robot motion is terminated
- Mastering is terminated



As the brakes are not applied when the application is terminated, subsequent motions of the robot cannot be ruled out. Particularly if the last motion command of the application was commanded in impedance mode, the robot is still compliant after the end of the application. The system integrator must ensure that this cannot result in unexpected system behavior and unexpected robot motions. For this reason, it is advisable to actuate the EMERGENCY STOP to ensure that the brakes are applied after the end of the application.

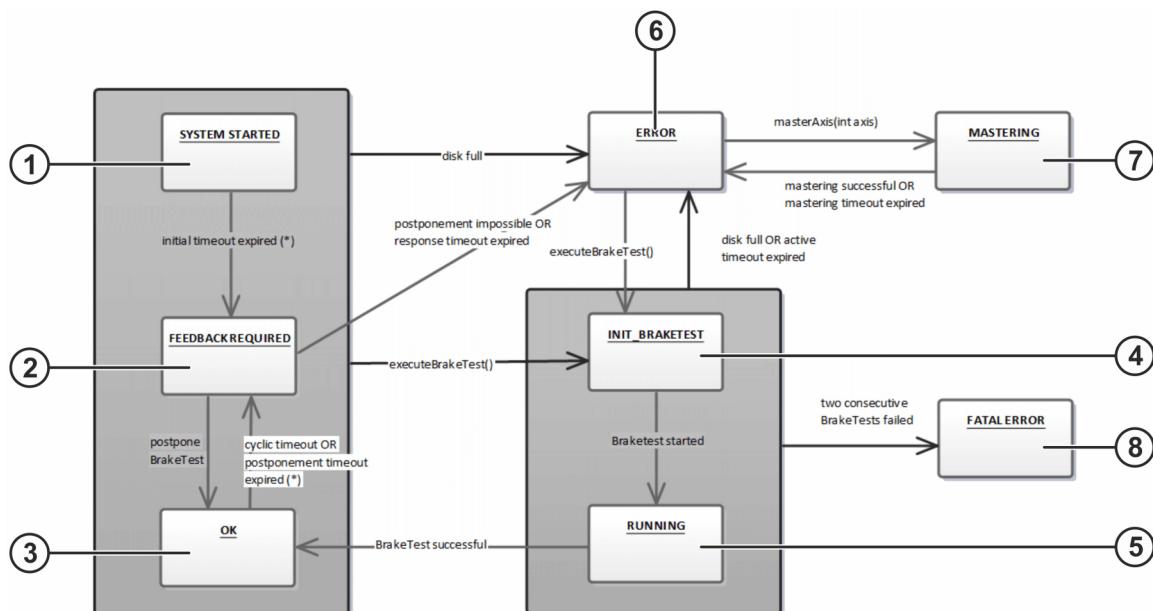
8.2 Cyclical brake test

The CyclicBrakeTestMonitor function is used to ensure the functionality of the robot brakes as a safety measure during a medical intervention. The user is informed when the next brake test is due. CyclicBrakeTestMonitor additionally enables the brake test to be started or postponed. After every restart of the LBR Med, the brakes must be checked before the robot is used. During the check, the brakes are tested in the sequence A1 ... A7 against the maximum hardware-specific torque of each brake. The functionality ensures that no robot motions are possible in the application (or during start-up via the smartPAD) in the event of a fault during the medical intervention. Furthermore, the results of the brake test are reliably saved on the controller and can be monitored.



The CyclicBrakeTestMonitor function forces a specific test of the brakes that is required in the medical environment.

Execution of the application for the brake test of the LBR iiwa from the folder **Application examples > LBR iiwa** remains possible. The results are ignored by CyclicBrakeTestMonitor, however.

**Fig. 8-1: State machine**

Item	Status	Description
1	SYSTEM_STARTED	Initial state of brake test monitoring after a restart of the robot.
2	FEEDBACK_REQUIRED	Brake test is due. The brake test must be started or postponed.
3	OK	The brake test has been executed successfully.
4	INIT_BRAKETEST	The brake test is being prepared. The robot moves to the start position.
5	RUNNING	The brake test is being performed.
6	ERROR	<ul style="list-style-type: none"> The brake test has failed. The announcement of the brake test has been ignored. The disk is full. Logging of the brake test is no longer possible.
7	MASTERING	Mastering is being carried out. Following successful mastering, the brake test switches back to the ERROR state.
8	FATAL_ERROR	2 successive brake tests have failed.

**WARNING**

If the **ERROR** status is active, all motion commands from the application and smartPAD are ignored. The robot does not move. The motion command cannot be executed until a successful brake test has been performed (status **OK**).

**DANGER**

In the error state **FATAL_ERROR**, the functionality of the brakes can no longer be assured. The robot may sag if it is no longer under servo control or is switched off. Death or severe injuries may result.

Configuration of the state machine

- initialTimeout: 1 hour
Maximum wait time after a restart of the robot before the first brake test is due.
- cyclicTimeout: 24 hours
Maximum cyclical wait time before the next brake test is due.
- postponementTimeout: 8 hours
Maximum wait time after a postponement before the next brake test is due.
- maxNumberOfPostponements: 3
Maximum number of permissible postponements of the brake test.
- activeBrakeTestTimeout: 100 seconds
Maximum time within which a running brake test must be completed.
- masteringTimeout: 30 seconds
Maximum time within which mastering must be completed.
- maxNumberOfFailedBrakeTests: 2
Maximum number of failed brake tests in succession before the FATAL_ERROR state is reached.
- limitDiskSpaceGettingLow: 1 GB
Limit value for the warning that the available disk capacity has been reached.
- limitDiskSpaceNotEnough: 50 MB
Limit value for the error message that there is no further disk space available.

8.2.1 Functionality of the cyclical brake test

Description

The Java package **com.kuka.med.medController** contains the Java classes **MedController.java** and **BrakeTestMonitor.java** as interfaces for application development.

The MedController must be initialized at the start of the application in order to be able to control the robot in the application.

Precondition

The following conditions must be met before the brake test is performed:

- All robot axes must be mastered.
- The disk space status must not be NOT_ENOUGH_SPACE.
- The brake test status must not be FATAL_ERROR.



WARNING

In the case of the error state FATAL_ERROR, the LBR Med must no longer be used. All robot motions are disabled, including performance of the brake test and mastering. Contact KUKA Customer Support.

8.2.2 Interfaces of the cyclical brake test

Overview

The BrakeTestMonitor class provides the following interfaces:

Method	Description
BrakeTestMonitor(controller)	<p>Constructor for creating an instance of the BrakeTestMonitor.</p> <p>Requires an instance of the MedController as the transfer parameter. The transfer parameter is initialized at the start of the robot application.</p>
getCurrentState()	<p>Return value type: Enum of type CyclicMonitoringState</p> <p>Returns the current status of the brake test monitoring.</p>
isPostponementPossible()	<p>Return value type: boolean</p> <p>Enabling of postponement of the brake test.</p> <ul style="list-style-type: none"> • true: the brake test can be postponed to a later time • false: the brake test cannot be postponed Causes: <ul style="list-style-type: none"> – The number of permissible postponements has been exceeded. – The brake test monitoring is in the ERROR state.
postponeBrakeTest()	<p>Return value type: boolean</p> <p>If a postponement is possible, the performance of the brake test is postponed.</p>
setResponseTime(<i>time</i>)	<p>Return value type: int (unit: s)</p> <p>Transfer parameter <i>time</i> (type: int; unit: s)</p> <p>Sets a new value for the time (> 0) before the brake test is due that a reminder is to be issued (default: 1 hour).</p> <p>Returns the currently accepted value, as the interval must be less than <i>postponementTimeout</i>. If, when the method is called, the brake test monitoring is waiting for feedback from the user, the new value is not taken into consideration at the current state transition, but in the next cycle.</p>
executeBrakeTest(<i>robot</i>)	<p>Return value type: Enum of type BrakeTestOutcome</p> <p>Transfer parameter: <i>robot</i> (type: Robot)</p> <p>If all preconditions have been met, the robot <i>robot</i> is moved to the initial configuration of the axis angles and the brake test is then executed for all axes.</p> <p>The return value contains the following information:</p> <ul style="list-style-type: none"> • Enum of type BrakeTestDiskSpaceState with information about disk space: (>>> "Enum" Page 156) • Enum of type CyclicMonitoringState with information about the monitoring status: (>>> "Enum" Page 156) • Enum of type OverallBrakeTestResult with information about the results of the brake test: (>>> "Enum" Page 157)

Method	Description
isBrakeTestExecutable()	<p>Return value type: boolean</p> <p>Returns a value indicating whether performance of the brake test is possible:</p> <ul style="list-style-type: none"> • true: due brake test can be performed • false: due brake test cannot be performed because 2 successive brake tests have failed
getTimeTillBraketestOverdue()	<p>Return value type: int</p> <p>Time remaining until the brake test is due (unit: s)</p> <ul style="list-style-type: none"> • > 0 • = 0 <p>Brake test is due or is currently being performed.</p>
checkFreeDiskSpace()	<p>Return value type: Enum of type BrakeTestDiskSpaceState</p> <p>Returns the current status of the hard disk space:</p> <p>(>>> <i>"Enum" Page 156</i>)</p>
isDiskFull()	<p>Return value type: boolean</p> <p>Checks whether the hard disk space for performance of the brake test is full.</p> <ul style="list-style-type: none"> • true: the disk is full; performance of the brake test is not possible • false: the disk is not full
addMonitoringListener(<i>listener</i>) removeMonitoringListener(<i>listener</i>)	<p>These two methods can be used to register or remove listeners (type: IBrakeTestMonitorListener).</p> <p>Registered listeners are informed of all changes to the brake test monitoring using <i>BrakeTestMonitorEvent</i>.</p>
setInitialPositionForBrakeTest(<i>joints, robot</i>)	<p>Return value type: boolean</p> <p>Sets the initial axis position in accordance with the transferred parameter <i>joints</i> for the robot <i>robot</i>.</p> <p>Before actual brake test begins, the robot moves to this axis position by means of a PTP motion.</p> <p>For all axis positions, the parameter <i>joints</i> must be within the axis limits minus 5°, otherwise setting of the values fails. The HOME position of the robot is always used as the default value.</p> <p>(>>> <i>15.15 "HOME position" Page 428</i>)</p> <ul style="list-style-type: none"> • true: setting of the initial position was successful • false: at least one axis position is outside the permissible range

Method	Description
setRelativeVelocityForInitialBrakeTestMotion()	<p>Return value type: boolean</p> <p>Sets the relative axis velocity for the first PTP motion which brings the robot into the initial axis position.</p> <p>The parameter is not set if the relatively axis velocity is outside of the permissible range.</p> <ul style="list-style-type: none"> • 0.03 ... 0.2 <p>Default: 0.2</p> <ul style="list-style-type: none"> • true: setting of the parameter was successful • false: the relative axis velocity is outside the permissible range



WARNING

The brake test must be concluded within 100 seconds, otherwise the monitoring function for the brake test switches to the ERROR state. For this reason, the first PTP motion of the brake test which brings the robot into the initial axis position must not take too long. Depending on the selected relative velocities for this first PTP motion, the following angle sizes may not be exceeded between the axis position of the robot prior to the first PTP motion and the initial axis position of the brake test:

- $\pm 170^\circ$ at 20% of the axis velocity
- 170° at 10% of the axis velocity
- 90° at 5% of the axis velocity
- 45° at 3% of the axis velocity

Enum

Type BrakeTestDiskSpaceState

The Enum of type BrakeTestDiskSpaceState has the following values:

Value	Description
UNKNOWN	It is not known how much disk space is still available. The information has not been checked.
ENOUGH_SPACE	There is enough disk space available for logging the brake test.
LOW_SPACE	There is only enough disk space available for logging one brake test. Contact KUKA Customer Support.
NOT_ENOUGH_SPACE	Brake test cannot be executed. The disk is full. Logging of the brake test is no longer possible. Contact KUKA Customer Support.

Enum

Type CyclicMonitoringState

The Enum of type CyclicMonitoringState has the following values:

Value	Description
SYSTEM_STARTED	Initial state of brake test monitoring after a restart of the robot.
FEEDBACK_REQUIRED	Brake test is due. The brake test must be started or postponed.
OK	The brake test has been executed successfully.
INIT_BRAKETEST	The brake test is being prepared. The robot moves to the start position.
RUNNING	The brake test is being performed.
MASTERING	Mastering is being carried out. Following successful mastering, the brake test switches back to the ERROR state.
ERROR	<ul style="list-style-type: none"> The brake test has failed. The announcement of the brake test has been ignored. The disk is full. Logging of the brake test is no longer possible.
FATAL_ERROR	2 successive brake tests have failed.

Enum

Type OverallBrakeTestResult

The Enum of type OverallBrakeTestResult has the following values:

Value	Description
UNKNOWN	<p>The overall result is not known.</p> <ul style="list-style-type: none"> Logging of the brake test could not be carried out. The disk is full. The start motion failed.
ERROR	At least one brake test was not completed successfully.
WARNING	The brake tests were successful, but there is a warning active for at least one axis, indicating a possible defect.
SUCCESSFUL	All brake tests were completed successfully.



WARNING

If the ERROR status is active, all motion commands from the application and smartPAD are ignored. The robot does not move. The motion command cannot be executed until a successful brake test has been performed (WARNING or SUCCESSFUL status).



WARNING

If the WARNING status is active, KUKA Customer Support must be contacted.



WARNING

In the case of the error state FATAL_ERROR, the robot can no longer be moved. Renewed performance of the brake test is not possible. Contact KUKA Customer Support.

8.2.3 Interface IBrakeTestMonitorListener

The interface IBrakeTestMonitorListener provides the callback method onMonitoringStateChanged(event) that is triggered by any change in the monitoring of the brake test.

BrakeTestMonitorEvent event contains the following information:

Method	Description
getCurrentCyclicMonitoringState()	Return value type: Enum of type CyclicMonitoringState Returns the current status of the brake test monitoring.
isPostponementAcceptable()	Return value type: Boolean Returns a value indicating whether a postponement of the brake test is permissible: <ul style="list-style-type: none"> • true: Postponement is permissible. • false: Postponement is not permissible. The brake test must be performed.
isBrakeTestExecutable()	Return value type: Boolean Returns a value indicating whether performance of the brake test is possible: <ul style="list-style-type: none"> • true: Brake test can be performed. • false: Brake test cannot be performed because 2 successive brake tests have failed.
getStateOfFreeDiskSpace()	Return value type: Enum of type BrakeTestDiskSpaceState Returns the current status of the hard disk space.
isDiskFull()	Return value type: Boolean Checks whether the hard disk space for performance of the brake test is full. <ul style="list-style-type: none"> • true: There is not enough hard disk space available. Performance of the brake test is not possible. • false: There is enough hard disk space available.

Example

```
public class BrakeTestMonitorSampleApp extends
RoboticsAPIApplication
implements IBrakeTestMonitorListener{

    @Inject
    private LBR _lbr;
    private MedController _controller;
    private BrakeTestMonitor _brakeTestMonitor;

    @Override
    public void initialize(){
        _controller = (MedController) _lbr.getController();
        _brakeTestMonitor = new BrakeTestMonitor(_controller);
        _brakeTestMonitor.addMonitoringListener(this);
        // User specific initialization
    }

    @Override
```

```

public void dispose() {
    _brakeTestMonitor.removeMonitoringListener(this);
    super.dispose();
}

@Override
public void run() {
    // User specific application
}

@Override
public void
onMonitoringStateChanged(BrakeTestMonitorEvent event) {
    // User specific behavior when monitoring state
    changes
}
}

```



A detailed example is contained in the sample project or can be installed in Sunrise.Workbench via com.kuka.med.lbrMedExamples.

8.2.4 Checking the LOG files



The brake test LOG files must be checked annually or if a check of the available disk space with checkFreeDiskSpace() returns the state LOW_SPACE.

(>>> [8.2.2 "Interfaces of the cyclical brake test" Page 153](#))

Procedure

1. Create a KRCDiag package and load it from the robot controller.
(>>> [20.5 "Collecting diagnostic information for error analysis at KUKA" Page 604](#))
2. Check whether the state “Brake test error” has been displayed in a LOG file of the KRCDiag package \Files\KRC\Roboter\Log\CyclicBrake-TestMonitor\CyclicBrakeTestMonitorYYYY-MM-DD_HH-MM-SS.CSV.
3. If the WARNING status has been displayed for a brake test, or if the brake test has failed repeatedly since the last maintenance, KUKA Customer Support must be contacted.



The medical device manufacturer must archive the results of the execution of the brake test.
It is advisable to archive the LOG files in such a way that they can be accessed by Service personnel.

9 Project management

9.1 Overview of a Sunrise project

A Sunrise project contains all the data which are required for the operation of a station. A Sunrise project comprises:

- Station configuration

The station configuration describes the static properties of the station. Examples include hardware and software components.

- Sunrise applications

Sunrise applications contain the source code for executing a task for the station. They are programmed in Java with KUKA Sunrise.Workbench and are executed on the robot controller. A Sunrise project can have any number of Sunrise applications.

- Runtime data

Runtime data are all the data which are used by the Sunrise applications during the runtime. These include, for example, end points for motions, tool data and process parameters.

- Safety configuration

The safety configuration contains the configured safety functions.

- I/O configuration (optional)

The I/O configuration contains the inputs/outputs of the used field buses mapped in WorkVisual. The inputs/outputs can be used in the Sunrise applications.

Sunrise projects are created and managed with KUKA Sunrise.Workbench.

(>>> [5.3 "Creating a Sunrise project with a template" Page 61](#))

There may only be 1 Sunrise project on the robot controller at any given time. This is transferred from Sunrise.Workbench to the robot controller by means of project synchronization.

(>>> [9.5 "Project synchronization" Page 187](#))

9.2 Frame management

Overview

Frames are coordinate transformations which describe the position of points in space or objects in a station. The coordinate transformations are arranged hierarchically in a tree structure. In this hierarchy, each frame has a higher-level parent frame with which it is linked through the transformation.

The root element or origin of the transformation is the world coordinate system which is located as standard in the robot base. This means that all frames are directly or indirectly related to the world coordinate system.

A transformation describes the relative position of 2 coordinate systems to each other, i.e. how a frame is offset and oriented relative to its parent frame.

The position of a frame relative to its parent frame is defined by the following transformation data:

- X, Y, Z: Offset of the origin along the axes of the parent frame
- A, B, C: Rotational offset of the axis angles of the parent frame

Rotational angle of the frames:

- Angle A: Rotation about the Z axis
- Angle B: Rotation about the Y axis
- Angle C: Rotation about the X axis
- Redundancy data
 - Status: Saves the position of axes A4 and A6
 - Turn: Saves the position of all axes
 - Redundancy angle: Saves the angle of axis E1
- Calibration data

9.2.1 Creating a frame for an application

Description

Frames can be created in the application data in Sunrise.Workbench. These frames are project-specific and can be used in every robot application of the project.

Once the project has been synchronized, the frames are available on the smartHMI. There, additional frames can be created and the frames taught in order to determine the position of the frames in space. Taught frames can be addressed manually.

(>>> [6.18.1 "“Frames” view" Page 109](#))

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Right-click on the desired parent frame in the **Application data** view and select **Insert new frame** from the context menu.
The new frame is created and inserted in the frame tree as a child element of the parent frame.
3. The system automatically generates a frame name. It is advisable to change the name in the frame properties. (>>> [9.2.5 "Displaying/editing frame properties" Page 165](#))
A descriptive frame name makes both programming and orientation within the program easier. The frame names must be unique within the hierarchy level and may not be assigned more than once.

Example

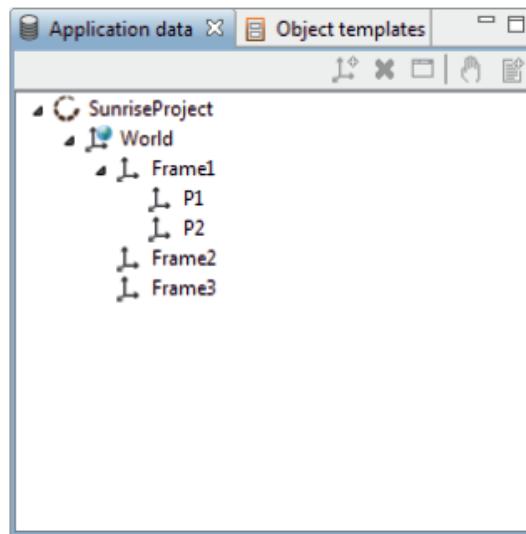


Fig. 9-1: Application data – frames

Frame1, 2 and 3 are child elements of World and are located on the same hierarchical level. P1 and P2 are child elements of Frame1 and are located one level below it.

9.2.2 Designating a frame as a base

Description

Frames created in the application data can be marked as a base: hand icon on the frame icon:

Only frames marked in this way can be selected and calibrated on the smartHMI as a base for jogging after synchronization of the project.

(>>> [6.16.1 "Jogging options" window" Page 103](#))

(>>> [7.5.2 "Base calibration: 3-point method " Page 145](#))



It is advisable to assign clear names to these frames to make it easier for the operator to select the jogging base on the smartPAD.

Procedure

Via the context menu:

- Right-click on the desired frame in the **Application data** view and select the hand icon **Base** from the context menu.

Via the toolbar:

- Select the desired frame in the **Application data** view and click on the hand icon **Base**.

Example

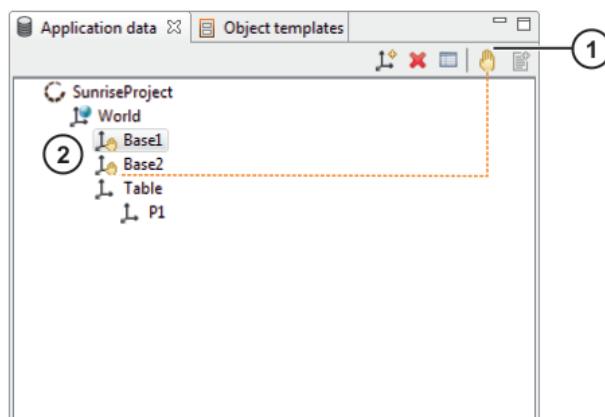


Fig. 9-2: Designating a frame as a base

- 1 **Base** hand icon
- 2 Base1 and Base2 are designated as the base

9.2.3 Moving a frame

Description

A frame can be moved in the application data and assigned to a new parent frame. The following points must be taken into consideration:

- The subordinate frames are automatically moved at the same time.
- The absolute position of the moved frames in space is retained. The relative transformation of the frames to the new parent frame is adapted.
- Frames cannot be inserted under one of their child elements.
- The names of the direct child elements of a frame must be unique.



If a frame is moved, its path changes. If this frame is used in the source code of an application, the path specification must be corrected accordingly in the application.

Procedure

1. Click on the desired frame in the **Application data** view and hold down the left mouse button.
2. Drag the frame to the new parent frame with the mouse.
3. When the desired new parent frame is selected, release the mouse button.

9.2.4 Deleting a frame

Description

Frames created in the application data can be removed from the frame tree again. If a frame has child elements, the following options are available:

- **Move children to parent:** Only the selected frame is deleted. The subordinate frames are retained, are moved up a level and assigned to a new parent frame.

The absolute position of the moved frames in space is retained. The relative transformation of the frames to the new parent frame is adapted.



If a frame is moved, its path changes. If this frame is used in the source code of an application, the path specification must be corrected accordingly in the application.

- **Delete parent and child frames:** Deletes the selected frame and all subordinate frames.

Procedure

1. Right-click on the frame to be deleted in the **Application data** view and select **Delete** from the context menu.
2. If the frame has no child elements, the system asks whether it really should be deleted. Answer the query with **Delete**.
3. If the frame has child elements, the system asks whether these should also be deleted. Select the desired option:
 - **Move children to parent**
 - **Delete parent and child frames**
4. Only with the option **Move children to parent**: If a name conflict occurs when moving the child elements, a notification message appears and the delete operation is canceled.
Remedy: Rename one of the frames in question and repeat the delete operation.

9.2.5 Displaying/editing frame properties



The position and orientation of a frame is generally defined during teaching with the robot. However, it is also possible to enter the position values of a frame manually or to change them at a later stage.

The following points must be taken into consideration:

- Modifying the transformation data not only moves the current frame, but also all of its subordinate child elements. This affects all applications in which these frames are used.
- The taught values of Status, Turn and redundancy angle are retained. Under certain circumstances, it may no longer be possible to address the frame or its child elements.
- After a modification to the transformation data, all programs in which the frame is used must be tested in Manual Reduced Velocity mode (T1).

Description

The properties of frames are displayed in the **Properties** view, distributed over various tabs. Some of the properties can be edited, others are for display only.

Procedure

1. Select the desired frame in the **Application data** view. The properties of the frame are displayed in the **Properties** view.
2. Edit the properties as required on the tabs.



For physical variables, the values can be entered with the unit. If this is compatible with the preset unit, the values are converted accordingly, e.g. cm into mm or ° into rad. If no unit is entered, the preset unit is used.

9.2.5.1 Frame properties – General tab

The **General** tab contains general information relating to the frame.

Parameter	Description
Name	Name of the frame
Comment	A comment on the frame can be entered here (optional).
Project	Project in which the frame was created (display only)
Last modification	Date and time of the last modification (display only)

9.2.5.2 Frame properties – Transformation tab

The **Transformation** tab contains the transformation data of the frame.

Parameter	Description
X, Y, Z	Translational offset of the frame relative to its parent frame
A, B, C	Rotational offset of the frame relative to its parent frame



If the transformation data of a frame that has been calibrated as a base are edited, the calibration information is deleted.

9.2.5.3 Frame properties – Redundancy tab

The **Redundancy** tab contains the redundancy information relating to the frame.

Parameter	Description
E1	Value of the redundancy angle (>>> 14.10.1 "Redundancy angle" Page 362)
Status	(>>> 14.10.2 "Status" Page 362)
Turn	(>>> 14.10.3 "Turn" Page 363)

9.2.5.4 Frame properties – Teach information tab

The **Teach information** tab contains information about a taught frame (display only).

Parameter	Description
Device	Robot that was used for teaching
TCP	Frame path for the TCP that was used for teaching
Tool	Tool that was used for teaching

Parameter	Description
X, Y, Z	Translational offset of the TCP relative to the origin frame of the tool
A, B, C	Rotational offset of the TCP relative to the origin frame of the tool



If a frame that has been calibrated as a base is retaught, the calibration information is deleted.

9.2.5.5 Frame properties – Measurement tab

The **Measurement** tab contains information about base calibration (for frames marked as a base; display only).

Parameter	Description
Measurement method	Method used
Last modification	Date and time of the last modification



If the data of a calibrated base are saved in Sunrise.Workbench by means of synchronization, the transformation data of the frame change in accordance with the calibration. The transformation data of the child elements of the frame are not changed by the calibration, i.e. only the position of the frame relative to the world coordinate system changes. The redundancy information also remains unchanged.

9.2.6 Inserting a frame in a motion instruction

Description

A frame created in the application data can be inserted as the end point in a motion instruction.

Procedure

1. Program the motion instruction, e.g. `robot.move(ptp());`.
2. In the **Application data** view, click on the frame which is to be used as the end point and hold down the left mouse button.
3. Drag the frame to the editor area with the mouse and position it so that the mouse pointer is between the brackets of the motion.
4. Release the mouse button. The frame is inserted as the end point of the motion.

Example

```
robot.move(ptp(getApplicationContext().getFrame("/P2/Target")));
```

The `getApplicationContext().getFrame(...)` method indicates that a frame created in the application data has been inserted. The end point of the motion is the `Target` frame.

As the transfer parameter, the method receives the path of the frame in the frame tree. The `Target` frame is a child element of `P2`.

9.3 Object management

Description

Tools and workpieces can be created and managed as object templates in Sunrise.Workbench. They belong to the runtime data of a project.

Tools

Properties:

- Tools are mounted on the robot flange.
- Tools can be used as movable objects in the robot application.
- The tool load data affect the robot motions.
- Tools can have any number of working points (TCPs) which are defined as frames.

Workpieces

Characteristics:

- Workpieces can be a wide range of objects which are used, processed or moved in the course of a robot application.
- Workpieces can be coupled to tools or other workpieces.
- Workpieces can be used as movable objects in the robot application.
- The workpiece load data affect the robot motions, e.g. when a gripper grips the workpiece.
- Workpieces can have any number of frames which mark relevant points, e.g. points on which a gripper grips a workpiece.

9.3.1 Geometric structure of tools

Every tool has an origin frame (root). As standard, the origin of the tool is defined to match the flange center point in position and orientation when the tool is mounted on the robot flange. The origin frame is always present and does not have to be created separately.

A tool can have any number of working points (TCPs), which are defined relative to the origin frame of the tool (root) or to one of its child elements.

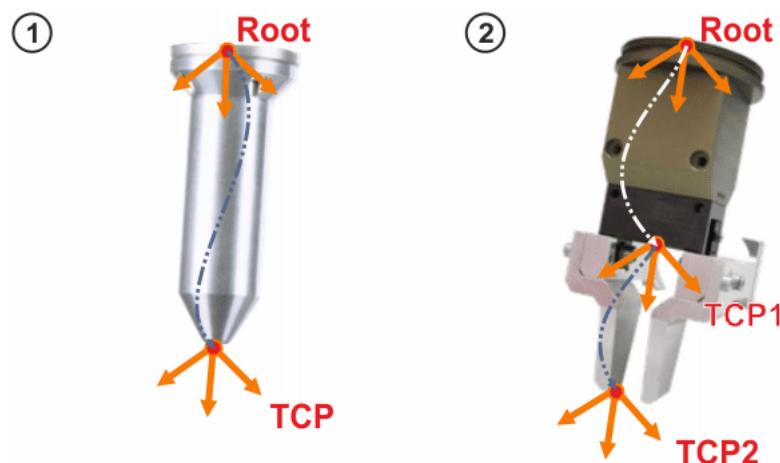


Fig. 9-3: Examples of TCPs of tools

1 Guiding tool with 1 TCP

2 Gripper with 2 TCPs



The transformation of the frames is static. For active tools, e.g. grippers, this means that the TCP does not adapt to the current position of jaws or fingers.

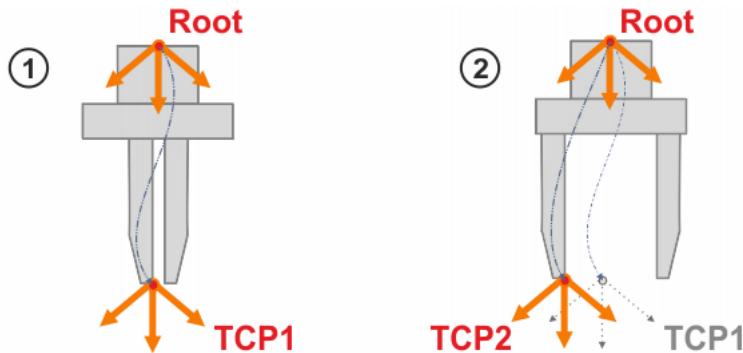


Fig. 9-4: Static TCP on a gripper

1 Gripper closed

2 Gripper open

9.3.2 Geometric structure of workpieces

Every workpiece has an origin frame (root). The origin frame is always present and does not have to be created separately.

A workpiece can have any number of frames, which are defined relative to the origin frame of the workpiece (root) or to one of its child elements.

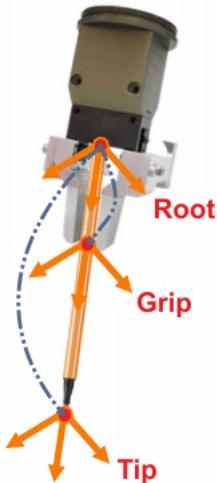


Fig. 9-5: Examples of frames of workpieces

9.3.3 Creating a tool or workpiece

Description

Tools and workpieces created as object templates for a project can be used in every robot application of the project.

The tools can be selected on the smartHMI for jogging after the project has been synchronized.

(>>> [6.16.1 "Jogging options" window](#) [Page 103](#))

Procedure

1. Select the desired project in the **Package Explorer** view.

2. Select the **Object templates** view.
3. To create a tool, right-click on the object type **Tools** and select **Insert new tool** from the context menu. The object template for the tool is created.
4. To create a workpiece, right-click on the object type **Workpieces** and select **Insert new workpiece** from the context menu. The object template for the workpiece is created.
5. The system automatically assigns an object name. It is advisable to change the name in the object properties. (>>> [9.3.5 "Displaying/editing object properties" Page 171](#))
The object names must be unique. A descriptive name makes both programming and orientation within the program easier.
6. Enter the load data in the object properties.
(>>> [9.3.4 "Entering load data" Page 170](#))

9.3.4 Entering load data

Description

Load data are all loads mounted on or connected to the robot flange. They form an additional mass mounted on the robot which must also be moved together with the robot.

The load data of tools and workpieces can be specified when the corresponding object templates are created. If several tools and workpieces are connected to the robot, the resulting total load is automatically calculated from the individual load data.

The load data are integrated into the calculation of the paths and accelerations. Correct load data are an important precondition for the optimal functioning of the servo control and help to optimize the cycle times.



WARNING

Danger to life and limb due to incorrect loads

Operating a robot with incorrect loads may result in death, severe injuries or damage to property. For example, the braking procedure may take too long with incorrect load data.

- Use only loads that are permissible for the robot.
- Use correct load data.

Sources

Load data can be obtained from the following sources:

- Manufacturer information
 - Manual calculation
 - CAD programs
 - The load data of tools can be determined automatically.
- (>>> [7.6 "Determining tool load data" Page 147](#))

Procedure

1. In the **Object templates** view, select the desired tool or workpiece.
2. In the **Properties** view, select the **Load data** tab and enter the load data.
(>>> [9.3.5.2 "Object properties – Load data tab" Page 171](#))

9.3.5 Displaying/editing object properties

Description

The properties of object templates are displayed in the **Properties** view, distributed over various tabs. Some of the properties can be edited, others are for display only.

Procedure

1. Select the object template in the **Object templates** view. The properties of the object template are displayed in the **Properties** view.
2. Edit the properties as required on the tabs.



For physical variables, the values can be entered with the unit. If this is compatible with the preset unit, the values are converted accordingly, e.g. cm into mm or ° into rad. If no unit is entered, the preset unit is used.

9.3.5.1 Object properties – General tab

The **General** tab contains general information about the tool or workpiece.

Parameter	Description
Name	Name of the tool or workpiece The name can be edited.
Template class	Runtime class of the tool or workpiece The class of the template can be edited.
Comment	Comment for the tool or workpiece (optional)
Frame	Frame of the tool or workpiece that is to be moved as standard As standard, the origin frame of the tool or workpiece is used as the standard frame for motions. Here, a different frame can be selected as the standard frame for motions, provided that a suitable frame has already been created for the tool or workpiece. (>>> 9.3.6 "Creating a frame for a tool or workpiece" Page 172)

9.3.5.2 Object properties – Load data tab

The load data of tools and workpieces can be entered on the **Load data** tab.

Parameter	Description
Mass	Mass of the tool or workpiece
Center of mass	Position of the center of mass relative to the origin frame of the tool or workpiece <ul style="list-style-type: none"> • MS X: X coordinate of the center of mass • MS Y: Y coordinate of the center of mass • MS Z: Z coordinate of the center of mass

Parameter	Description
Principal inertia axes	<p>Orientation of the principal inertia axes relative to the origin frame of the tool or workpiece</p> <ul style="list-style-type: none"> • MS A: orientation of the Z axis • MS B: orientation of the Y axis • MS C: orientation of the X axis <p>The principal inertia axes run through the center of mass.</p>
Principal moments of inertia	<p>Mass moments of inertia about the principal inertia axes of the tool or workpiece</p> <ul style="list-style-type: none"> • jX: moment of inertia about the X axis • jY: moment of inertia about the Y axis • jZ: moment of inertia about the Z axis

Example

Principal moment of inertia **jX**:

jX is the inertia about the X axis of the principal inertia axes. This is rotated through **MS A**, **MS B** and **MS C** relative to the origin frame of a tool and shifted in the center of mass.

jY and **jZ** are the analogous principal moments of inertia about the Y and Z axes.

9.3.6 Creating a frame for a tool or workpiece

Description

Each frame created for a tool or workpiece can be programmed in the robot application as the reference point for motions.

After the project is synchronized, the frames of a tool can be selected as the TCP for Cartesian jogging on the smartHMI.

(>>> [6.16.1 "Jogging options" window](#) [Page 103](#))

The frames of a tool (TCPs) can be calibrated with robot relative to the flange coordinate system.

(>>> [7.5.1 "Tool calibration"](#) [Page 138](#))

If the data of a calibrated tool are saved in Sunrise.Workbench by means of synchronization, the transformation data of the frame change in accordance with the calibration.



The tool data of the TCP used to execute a Cartesian motion influence the robot velocity. Incorrectly entered tool data can cause unexpectedly high Cartesian velocities at the installed tool. The velocity of 250 mm/s may be exceeded in T1 mode.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Right-click on the desired object template in the **Object templates** view and select **Insert new frame** from the context menu. The frame is created.

At the top hierarchy level, the parent frame of the created frame is the origin frame of the object.

3. To insert a new frame under an existing frame of the object, right-click on this parent frame and select **Insert new frame** from the context menu. The frame is created.
4. The system automatically generates a frame name. It is advisable to change the name in the frame properties. (>>> [9.3.7 "Displaying/editing frame properties" Page 173](#))
A descriptive frame name makes both programming and orientation within the program easier. The frame names must be unique within the hierarchy level and may not be assigned more than once.
5. In the frame properties, enter the transformation data of the frame relative to its parent frame:
 - Boxes **X**, **Y**, **Z**: offset of the frame along the axes of the parent frame
 - Boxes **A**, **B**, **C**: orientation of the frame relative to the parent frame
 (>>> [9.3.7 "Displaying/editing frame properties" Page 173](#))

9.3.7 Displaying/editing frame properties

Description

The properties of frames are displayed in the **Properties** view, distributed over various tabs. Some of the properties can be edited, others are for display only.

Procedure

1. Select the desired frame in the **Object templates** view. The properties of the frame are displayed in the **Properties** view.
2. Edit the properties as required on the tabs.



For physical variables, the values can be entered with the unit. If this is compatible with the preset unit, the values are converted accordingly, e.g. cm into mm or ° into rad. If no unit is entered, the preset unit is used.

9.3.7.1 Frame properties – General tab

The **General** tab contains general information relating to the frame.

Parameter	Description
Name	Name of the frame
Comment	A comment on the frame can be entered here (optional).

9.3.7.2 Frame properties – Transformation tab

The **Transformation** tab contains the transformation data of the frame.

The value ranges apply to safety-oriented frames. Frames with transformation data outside this range of values cannot be used as safety-oriented frames.

Parameter	Description
X, Y, Z	Translational offset of the frame relative to its parent frame <ul style="list-style-type: none"> • -10,000 mm ... +10,000 mm
A, B, C	Rotational offset of the frame relative to its parent frame <ul style="list-style-type: none"> • Any



If the transformation data of a frame that is used as the TCP of a calibrated tool are edited, the calibration information is deleted.

9.3.7.3 Frame properties – Measurement tab

The **Measurement** tab contains information about tool calibration (display only).

Parameter	Description
Measurement method	Method used
Calculation error	Translational or rotational calculation error which specifies the quality of the calibration (unit: mm or °)
Last modification	Date and time of the last modification



If the transformation data of a calibrated tool are edited, the calibration information is deleted.

9.3.7.4 Frame properties – Safety tab

The **Safety** tab is only available for tools and frames of tools. The following properties of safety-oriented tool frames can be configured:

Parameter	Description
Radius	Radius of the sphere on the safety-oriented frame <ul style="list-style-type: none"> • 25 ... 10,000 mm Provided that the check box for the property Safety-oriented is activated and the frame belongs to a safety-oriented tool, the sphere on the safety-oriented frame is active and can be used in the AMFs for monitoring Cartesian spaces and velocities.
Safety-oriented	Activation as safety-oriented frame <ul style="list-style-type: none"> • Check box active: Frame is a safety-oriented frame. • Check box not active: Frame is not a safety-oriented frame. Precondition for activating the check box: <ul style="list-style-type: none"> • A permissible value has been entered for the radius.

Parameter	Description
Is point for the tool-related velocity component AMF	<p>Use of the safety-oriented frame as a point for the <i>Tool-related velocity component</i> AMF</p> <ul style="list-style-type: none"> Check box active: Frame is used as a point for the AMF. Check box not active: Frame is not used as a point for the AMF. <p>Precondition for activating the check box:</p> <ul style="list-style-type: none"> The check mark is set at Safety-oriented. The frame belongs to a safety-oriented tool.

9.3.8 Defining a default motion frame

Description

As standard, the origin frame of an object template is used as the default frame for motions.

Any other suitable frame can be defined as the default frame for motions, e.g. if a tool or workpiece has a frame with which a large part of the motions must be executed. This simplifies motion programming.

(>>> [15.10.3 "Moving tools and workpieces" Page 408](#))

The default frame used for motions is indicated with an icon  in the **Object templates** view.

Procedure

Via the context menu:

- Right-click on the desired frame of the object template in the **Object templates** view and select **Default motion frame** from the context menu.

Via the toolbar:

- Select the desired frame of the object template in the **Object templates** view and click on the **Default motion frame** icon.

In the properties of the object template:

- Select the object template in the **Object templates** view.
- In the **Properties** view, select the **General** tab and set the frame that is to be moved as standard.

Example

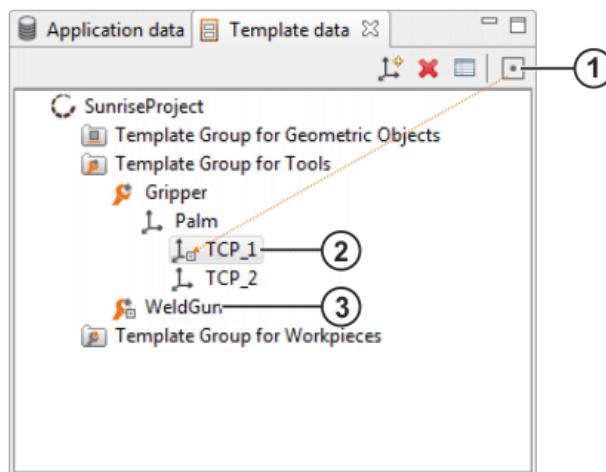


Fig. 9-6: Default frame for motions

- 1 Default motion frame icon
- 2 Default frame of the **Gripper** tool: TCP_1
- 3 Default frame of the **WeldGun** tool: Origin frame

9.3.9 Copying an object template

Description

When an object template is copied, a copy of the object templates including all frames is created. The properties of the object template and its frames, with the exception of the safety properties, are included in the copy.

Procedure

- Right-click on the object template in the **Object templates** view and select **Create copy** from the context menu.

9.3.10 Configuring a safety-oriented tool

Description

Up to 50 safety-oriented tools can be defined in a Sunrise project. Just like any other tool, a safety-oriented tool can have any number of frames. In order to configure the safety-oriented tool, suitable frames must be defined as safety-oriented frames (max. 6). The monitoring spheres that model the safety-oriented tool are configured at these frames:

- The center of the sphere is situated, by definition, at the origin of the safety-oriented frame.
- The radius of the sphere is defined in the frame properties.

Monitoring functions

Safety-oriented tools are relevant for the following safety monitoring functions:

- Monitoring of Cartesian spaces
The spheres can be monitored against the limits of activated Cartesian monitoring spaces.
- Monitoring of the translational Cartesian velocity
The velocity of the sphere center points is monitored.
(>>> [13.14.8 "Velocity monitoring functions" Page 282](#))
- Collision detection and TCP force monitoring functions
Only correctly specified load data ensure the accuracy of these monitoring functions. The load data of safety-oriented tools, in particular the mass and center of mass of the tools, must be configured. In the case of tools with comparatively high moments of inertia ($> 0.1 \text{ kg m}^2$), these data must also be specified in order to ensure the accuracy of these monitoring functions.
(>>> [13.14.13.2 "Collision detection" Page 301](#))
(>>> [13.14.13.3 "TCP force monitoring" Page 303](#))
(>>> [13.14.13.4 "Direction-specific monitoring of the external force on the TCP" Page 304](#))



If workpieces are used that are to be taken into consideration for safety-oriented Cartesian space or velocity monitoring, e.g. due to the dimensions of the workpieces, the spheres of the safety-oriented tool must be configured accordingly.

Safety-oriented frames can additionally be defined for the following tool-specific safety monitoring functions:

- Monitoring of the tool orientation (only available for robots)
One of the safety-oriented frames can be defined as the tool orientation frame. Safety-oriented monitoring of the orientation of this frame can be carried out.
(>>> [13.14.10 "Monitoring the tool orientation" Page 296](#))
- Monitoring of the tool-related velocity component (available for robots and mobile platforms)
Up to 6 safety-oriented frames of a safety-oriented tool can be defined as monitoring points for the tool-related velocity monitoring. A safety-oriented frame can additionally be defined as the orientation for the monitoring. This orientation frame defines the orientation of the coordinate system in which the velocity of the monitoring points is described. In tool-related velocity monitoring, a component of this velocity can be monitored.
(>>> [13.14.8.3 "Direction-specific monitoring of Cartesian velocity" Page 285](#))

Precondition

- Tool and corresponding frames have been created.
- When using the AMFs *Collision detection*, *TCP force monitoring* and *Base-related TCP force component*: The correct load data of the tool, in particular the mass and center of mass of the tool, are known.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. In the **Object templates** view, select the tool that is to be safety-oriented.
3. In the **Properties** view, select the **Safety** tab and activate the **Safety-oriented** check box.

The tool icon in the **Object templates** view is highlighted in yellow and marked with a sphere symbol .

4. Select the **Load data** tab and enter any missing tool load data.
Tools with load data outside the specified range of values cannot be used as safety-oriented tools.

(>>> [9.3.10.1 "Tool properties – Load data tab" Page 179](#))



To avoid spending unnecessary time performing verifications, mark a tool as safety-oriented only when the load data have been correctly entered or determined and have been transferred to Sunrise.Workbench. (>>> [9.3.4 "Entering load data" Page 170](#))

5. In the **Object templates** view, select the tool frame that is to be safety-oriented.
6. In the **Properties** view, select the **Transformation** tab and enter any missing transformation data of the frame with respect to its parent frame.
Frames with transformation data outside the specified range of values cannot be used as safety-oriented frames.
(>>> [9.3.10.2 "Frame properties – Transformation tab" Page 179](#))
7. Select the **Safety** tab and enter the radius of the monitoring sphere on the safety-oriented frame.
(>>> [9.3.10.3 "Frame properties – Safety tab" Page 180](#))
8. Set the check mark at **Safety-oriented**.
The frame icon in the **Object templates** view is highlighted in yellow .
9. If the safety-oriented frame is to be used as a point for the *Tool-related velocity component* AMF, activate the check box next to **Is point for the tool-related velocity component AMF**.
Safety-oriented frames that are used for tool-specific safety monitoring functions are additionally marked with a  symbol in the **Object templates** view.
10. Repeat steps 5 to 9 to define further safety-oriented tool frames.
11. In the safety properties of the tool, set the safety-oriented frames that are to be used for the tool-specific safety monitoring functions if required:
 - a. Select the safety-oriented tool in the **Object templates** view.
 - b. In the **Properties** view, select the **Safety** tab.
 - c. Under **Safety properties** assign the desired safety-oriented frames to the tool-specific safety monitoring functions.
(>>> [9.3.10.4 "Tool properties – Safety tab" Page 180](#))
12. Map the safety-oriented tool in the safety configuration.
(>>> [13.12.5 "Mapping safety-oriented tools" Page 271](#))

Alternative procedure

The properties of safety-oriented tools and frames can also be set via the context menu:

- Right-click on the tool or frame in the **Object templates** view and select the desired property from the context menu.

- Safety-oriented
- Tool orientation frame
- Orientation for tool-related velocity
- Point for tool-related velocity
 - The menu item is only available for frames.

Example

For a safety-oriented gripper, 3 monitoring spheres are configured.

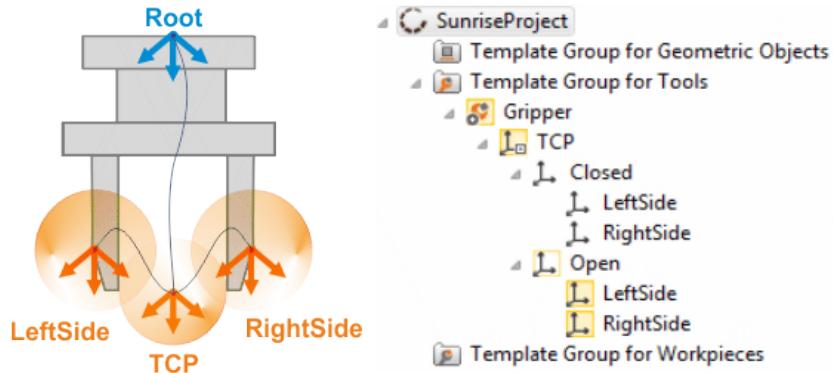


Fig. 9-7: Safety-oriented gripper

9.3.10.1 Tool properties – Load data tab

The **Load data** tab contains the load data of the tool.

The value ranges apply to safety-oriented tools. Tools with load data outside these ranges of values cannot be used as safety-oriented tools.

Parameter	Description
Mass	Mass of the tool <ul style="list-style-type: none"> • 0 ... 2,000 kg
MS X, MS Y, MS Z	Position of the center of mass relative to the origin frame of the tool <ul style="list-style-type: none"> • -10,000 ... +10,000 mm
MS A, MS B, MS C	Orientation of the principal inertia axes relative to the origin frame of the tool <ul style="list-style-type: none"> • Any
jX, jY, jZ	Mass moments of inertia of the tool <ul style="list-style-type: none"> • 0 ... 1,000 kg·m²

9.3.10.2 Frame properties – Transformation tab

The **Transformation** tab contains the transformation data of the frame.

The value ranges apply to safety-oriented frames. Frames with transformation data outside this range of values cannot be used as safety-oriented frames.

Parameter	Description
X, Y, Z	Translational offset of the frame relative to its parent frame <ul style="list-style-type: none"> • -10,000 mm ... +10,000 mm
A, B, C	Rotational offset of the frame relative to its parent frame <ul style="list-style-type: none"> • Any



If the transformation data of a frame that is used as the TCP of a calibrated tool are edited, the calibration information is deleted.

9.3.10.3 Frame properties – Safety tab

The **Safety** tab is only available for tools and frames of tools. The following properties of safety-oriented tool frames can be configured:

Parameter	Description
Radius	Radius of the sphere on the safety-oriented frame <ul style="list-style-type: none"> • 25 ... 10,000 mm Provided that the check box for the property Safety-oriented is activated and the frame belongs to a safety-oriented tool, the sphere on the safety-oriented frame is active and can be used in the AMFs for monitoring Cartesian spaces and velocities.
Safety-oriented	Activation as safety-oriented frame <ul style="list-style-type: none"> • Check box active: Frame is a safety-oriented frame. • Check box not active: Frame is not a safety-oriented frame. Precondition for activating the check box: <ul style="list-style-type: none"> • A permissible value has been entered for the radius.
Is point for the tool-related velocity component AMF	Use of the safety-oriented frame as a point for the <i>Tool-related velocity component</i> AMF <ul style="list-style-type: none"> • Check box active: Frame is used as a point for the AMF. • Check box not active: Frame is not used as a point for the AMF. Precondition for activating the check box: <ul style="list-style-type: none"> • The check mark is set at Safety-oriented. • The frame belongs to a safety-oriented tool.

9.3.10.4 Tool properties – Safety tab

The safety-oriented tool can be configured on the **Safety** tab.

Parameter	Description
Safety-oriented	Activation as safety-oriented tool <ul style="list-style-type: none"> Check box active: The tool is a safety-oriented tool Check box not active: The tool is not a safety-oriented tool
Tool orientation frame	Safety-oriented frame, the orientation of which can be monitored using the AMF <i>Tool orientation</i> . If no tool orientation frame is defined, the pickup frame of the tool is used as the tool orientation frame. (>>> <i>"Pickup frame"</i> Page 181)
Points for the velocity	Display of the safety-oriented frames defined in the frame properties as points for the <i>Tool-related velocity component</i> AMF. If no safety-oriented frame has been defined as a point for the tool-related velocity component, the following entry is displayed: <ul style="list-style-type: none"> DEFAULT: / In this case, the pickup frame of the tool is used and the velocity at the origin of the pickup frame is monitored. (>>> <i>"Pickup frame"</i> Page 181)
Orientation for the velocity	Safety-oriented frame, the orientation of which determines the directions in which the Cartesian velocity can be monitored using the <i>Tool-related velocity component</i> AMF. If no orientation frame is defined for the tool-related velocity component, the pickup frame of the tool is used. The orientation of the pickup frame determines the monitoring direction. (>>> <i>"Pickup frame"</i> Page 181)

Pickup frame

The pickup frame of a tool is dependent on the kinematic system on which it is mounted and on the tool configuration:

- The tool is mounted on the robot flange: the pickup frame is the flange coordinate system of the robot.
- The tool is mounted on a mobile platform: the pickup frame is the coordinate system at the center point of the platform.
- The tool is mounted on a fixed tool: the pickup frame is the standard frame for motions of the fixed tool.

9.3.11 Safety-oriented use of workpieces

Description

Loads picked up by the robot, e.g. a gripped workpiece, exert an additional force on the robot and influence the torques measured by the axis torque sensors.

The following AMFs require the workpiece load data for calculation of the external forces and torques:

- TCP force monitoring*
(>>> *13.14.13.3 "TCP force monitoring"* Page 303)

- Base-related TCP force component
(>>> [13.14.13.4 "Direction-specific monitoring of the external force on the TCP" Page 304](#))
- Collision detection
(>>> [13.14.13.2 "Collision detection" Page 301](#))

Programming

If one of these load-specific AMFs is active and workpieces are picked up at the same time, the current workpiece must be transferred to the safety controller. For this, the KUKA RoboticsAPI offers the method setSafetyWorkpiece().

(>>> [15.10.5 "Transferring workpiece load data to the safety controller" Page 412](#))

Once the workpiece has been transferred, the workpiece load data are taken into consideration by the safety controller. A load change, e.g. if a workpiece is set down again, must also be communicated with setSafetyWorkpiece(...).

Requirements

The workpiece transferred to the safety controller must meet the following requirements:

- The workpiece load data must be within the specified limits. If this is not the case, the load data are invalid and cannot be transferred to the safety controller.
(>>> [9.3.11.2 "Workpiece properties – Load data tab" Page 183](#))
- In order to be able to use workpieces as safety-oriented workpieces, the mass of the heaviest workpiece that could possibly be picked up by the robot must additionally be configured in the safety-oriented project settings.
(>>> [9.3.11.3 "Configuring the mass of the heaviest workpiece" Page 184](#))

The mass from the workpiece load data must not exceed the configured mass of the heaviest workpiece. Otherwise, load-specific AMFs are violated.

Workpiece pick-up

The way in which the load data of a workpiece influence the workpiece load-dependent AMFs depends on how the workpiece is picked up. For a workpiece, the safety controller requires the origin frame of the workpiece to be identical to the standard frame for motions of the safety-oriented tool.

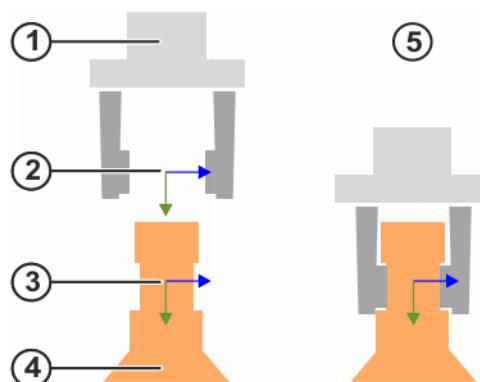


Fig. 9-8: Configuration of workpiece pick-up

Item	Description
1	Safety-oriented tool
2	Standard frame for motions of the safety-oriented tool: Frame of the safety-oriented tool on which the workpiece must be picked up. It is not necessary for this frame to be a safety-oriented frame.
3	Origin frame of the workpiece Frame of the workpiece on which the safety-oriented tool must pick up the workpiece.
4	Workpiece
5	Status after transfer of the workpiece to the safety controller The origin frame of the workpiece is identical to the standard frame for motions of the safety-oriented tool.

9.3.11.1 Entering workpiece load data

Precondition

- Workpiece has been created.
- When using the AMFs *Collision detection*, *TCP force monitoring* and *Base-related TCP force component*: The correct load data of the workpiece, in particular the mass and center of mass of the workpiece, are known.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Select the workpiece in the **Object templates** view.
3. Select the **Load data** tab and enter the workpiece load data.
(>>> [9.3.11.2 "Workpiece properties – Load data tab" Page 183](#))

Once the project has been synchronized, the workpiece can be used in the application.

9.3.11.2 Workpiece properties – Load data tab

The workpiece load data can be entered on the **Load data** tab.

The value ranges apply for workpieces that are used as safety-oriented workpieces. Workpieces with load data outside these ranges of values cannot be used as safety-oriented workpieces.

Parameter	Description
Mass	Mass of the workpiece <ul style="list-style-type: none"> • 0.001 ... 2,000 kg
MS X, MS Y, MS Z	Position of the center of mass relative to the origin frame of the workpiece <ul style="list-style-type: none"> • -10,000 ... +10,000 mm
MS A, MS B, MS C	Orientation of the principal inertia axes relative to the origin frame of the workpiece <ul style="list-style-type: none"> • 0° ... 359°

Parameter	Description
jX, jY, jZ	Mass moments of inertia of the workpiece <ul style="list-style-type: none">• 0 ... 1,000 kg·m²

9.3.11.3 Configuring the mass of the heaviest workpiece

Description

If the following load-specific AMFs are used, the mass of the heaviest workpiece that could possibly be picked up by the robot must be configured in the safety-oriented project settings:

- *TCP force monitoring*
- *Base-related TCP force component*
- *Collision detection*

Each of the load-specific AMFs checks whether the workpiece mass transferred to the safety controller with the load data exceeds the configured mass of the heaviest workpiece. If the mass of the heaviest workpiece is not configured, it is initialized with the default value (= 0.0 kg) and the AMF is violated if the workpiece load data from the safety controller are used.



Incorrect configuration of the mass of the heaviest workpiece can result in loss of the safety integrity with the following AMFs:

- *TCP force monitoring*
- *Base-related TCP force component*

The configuration must therefore be verified during safety acceptance.
(>>> [13.17.7.4 "Mass of the heaviest workpiece" Page 342](#))

Procedure

1. Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Change project settings** from the context menu.
The **Properties for [Sunrise Project]** window opens.
2. Select **Sunrise > Safety** in the directory in the left area of the window.
3. Enter the mass of the heaviest workpiece under **Heaviest workpiece** in the right-hand area of the window:
 - **0.0 ... 2000.0 kg**
4. Click on **OK** to apply the settings and close the window.

9.4 User management

Different functions can be executed on the robot controller, depending on the user group. The passwords of the user groups are managed in the project settings.

The following user groups are available as standard:

- **Administrator**

The Administrator manages the passwords of the user groups. The user group is protected by means of a password.

The default password is “kuka”.

- **Operator**

The user group “Operator” is the default user group.

- **Safety maintenance technician**

The user “Safety maintenance” is responsible for starting up the safety equipment of the robot. Only he can modify the safety configuration on the robot controller.

The user group is protected by means of a password.

The default password is “argus”.



Prior to start-up, the passwords for the user groups must be modified by the administrator, transferred to the robot controller in an installation procedure and activated. The passwords must only be communicated to authorized personnel. (>>> [9.4.2 "Changing and activating the password" Page 186](#))



If the administrator password is forgotten, KUKA Service must be notified and restore the default passwords. After this, the passwords must be reassigned.

9.4.1 Overview of Sunrise.RolesRights

Description

Sunrise.RolesRights is an add-on option package that expands the user groups available as standard to include the user group **Expert**. At the same time, the rights of the user groups are reassigned.

User groups

The following user groups are available with Sunrise.RolesRights:

- **Administrator**

Only available in Sunrise.Workbench. The Administrator manages the passwords of the user groups.

The user group is protected by means of a password.

The default password is “kuka”.

- **Operator**

The user group “Operator” is the default user group.

- **Expert**

Additional protected functions, that may not be performed by the “Operator”, are available to the user group “Expert”.

The user group is protected by means of a password.

The default password is “kuka”.

- **Safety maintenance technician**

The user “Safety maintenance” is responsible for starting up the safety equipment of the robot. All functions of the user group “Expert” are available to the user group “Safety maintenance”. Users in this user group can additionally modify the safety configuration on the robot controller.

The user group is protected by means of a password.

The default password is “argus”.



Prior to start-up, the passwords for the user groups must be modified by the administrator, transferred to the robot controller in an installation procedure and activated. The passwords must only be communicated to authorized personnel. (>>> [9.4.2 "Changing and activating the password" Page 186](#))

Functions

Depending on the installed software, the users can execute the following functions:



Installation of Sunrise.RolesRights restricts the user rights of the operator.

Basic software functions in Sunrise.Workbench:

Function	Operator	Expert	Safety maintenance
Project synchronization with unchanged safety configuration	✗	✓	✓
Project synchronization with changed safety configuration	✗	✗	✓

Basic software functions on smartHMI:

Function	Operator	Expert	Safety maintenance
Selecting/deselecting an application	✓	✓	✓
Pausing an application	✓	✓	✓
Moving the robot manually	✓	✓	✓
Selecting an operating mode (T1, T2, CRR, AUT)	✓	✓	✓
Activating/deactivating/resetting the safety configuration	✗	✗	✓
Changing the value of an output	✗	✓	✓
Teaching frames	✗	✓	✓
Creating a new frame	✗	✓	✓
Robot mastering/unmastering	✗	✓	✓

Sunrise.BackupRestore functions on smartHMI:

Function	Operator	Expert	Safety maintenance
Backing up data manually	✗	✓	✓
Restoring data manually	✗	✓	✓

9.4.2 Changing and activating the password

Description

The passwords for the user groups on the robot controller are defined in the project settings in Sunrise.Workbench. If these passwords are changed, they can only be activated by an installation of the system software on the robot controller.



The Administrator merely manages the passwords in Sunrise.Workbench. If only the Administrator password is modified, no installation is required. The modified Administrator password takes effect immediately and is saved as part of the project on the robot controller.



If the administrator password is forgotten, KUKA Service must be notified and restore the default passwords. After this, the passwords must be reassigned.

Precondition

- Administrator user group
- Network connection to the robot controller

Procedure

1. Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Change project settings** from the context menu.
The **Properties for [Project]** window opens.
 2. Select **Sunrise > Passwords** in the directory in the left area of the window.
 3. Click on **Login** and enter the Administrator password. Confirm the password with **OK**.
 4. Select the user group for which the password is to be changed.
 5. Enter the new password twice.
For security reasons, the entries are displayed encrypted. Upper and lower case are distinguished.
-
- The password must consist of at least one character. Only characters that can be entered via the smartHMI are permissible.
6. Click on **Save** and close the window.
 7. Install the system software on the robot controller. The modified passwords are active after the reboot of the robot controller.

9.5 Project synchronization

Description

In project synchronization, project data are transferred between Sunrise.Workbench and the robot controller. In the process, the projects are compared with one another. If there are different projects or version conflicts, the user can choose the direction in which to transfer the project data.

The following cases are distinguished:

- There is no project on the robot controller yet or there is a different project from the one to be transferred
(>>> [9.5.1 "Transferring the project to the robot controller" Page 188](#))
 - The same project exists on the robot controller and in Sunrise.Workbench but in different versions, e.g.:
 - When the project data are modified in Sunrise.Workbench only
 - When the project data are modified on the robot controller only
 - When the project data are modified on both sides
- (>>> [9.5.2 "Updating the project" Page 189](#))

Authorization

If the option package Sunrise.RolesRights is installed, only the user groups “Expert” and “Safety maintenance” are authorized to transfer a project to the robot controller by means of synchronization:

- User group **Expert**
Default user group for performing project synchronization
- User group **Safety maintenance technician**
Only required if the safety configuration has been changed.

The **Authorization** dialog is automatically opened when project synchronization is performed. The required user group is already preset here. It only remains to enter the correct user password.

9.5.1 Transferring the project to the robot controller

Description

The procedure described here applies if no project is on the robot controller yet or if there is a different project from the one to be transferred.

Precondition

- Network connection to the robot controller
- The system software is installed.
- The installed system software is compatible with the station configuration of the project to be transferred.
- No application is running on the robot controller.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Click on the **Synchronize project** button.
The system scans the robot controller for existing project data. If the scan fails, the cause of the error is displayed in a message.
3. If the scan is successful, the **Project synchronization** window opens.

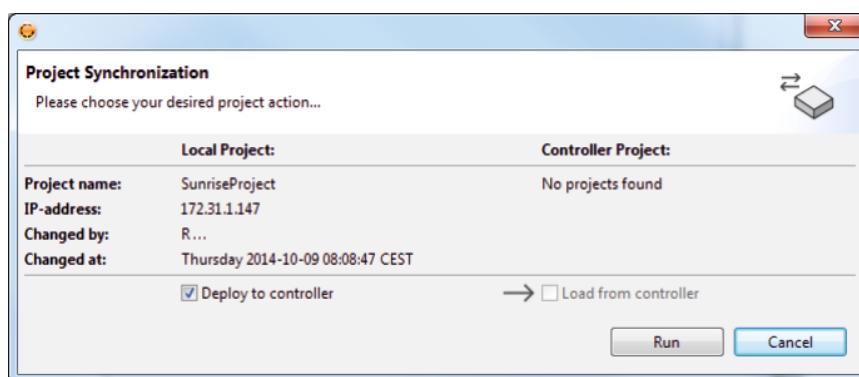


Fig. 9-9: Transferring the project to the controller

4. Click on **Execute**.
5. If the safety configuration or I/O configuration is modified, a dialog indicates that the robot controller must be rebooted in order to complete the synchronization.
 - Click on **OK** to transfer the project to the robot controller. Once the transfer is completed, the robot controller automatically reboots.
 - Transfer of the project can be stopped with **Cancel**.
6. Only if the option package Sunrise.RolesRights is used: The “**Authorization**” dialog opens. The required user group is preset. Enter password and confirm with **OK**.

7. The progress of the project transfer is displayed in a dialog both in Sunrise.Workbench and on the smartPAD. Once the transfer is completed, the dialog is closed and the robot controller automatically reboots.
If the transfer fails, a corresponding dialog is displayed both in Sunrise.Workbench and on the smartPAD. In addition, the cause of the error is displayed in Sunrise.Workbench.
Confirm the dialog in Sunrise.Workbench and on the smartPAD with **OK**.
8. If the safety configuration is modified, activate this on the robot controller.

9.5.2 Updating the project

Description

The procedure described here applies if the same project exists in Sunrise.Workbench and on the robot controller, but in different versions.

Precondition

- Network connection to the robot controller
- If a project is transferred to the robot controller: No application is running on the robot controller.

Procedure

1. Select the desired project in the **Package Explorer** view.
2. Click on the **Synchronize project** button.
The system scans the robot controller for existing project data. If the scan fails, the cause of the error is displayed in a message.
3. If the project in Sunrise.Workbench is identical to the project on the robot controller, a dialog indicates that no synchronization is necessary. Confirm the dialog with **OK**. Synchronization is aborted.
4. If the scan is successful, the **Project synchronization** window opens.

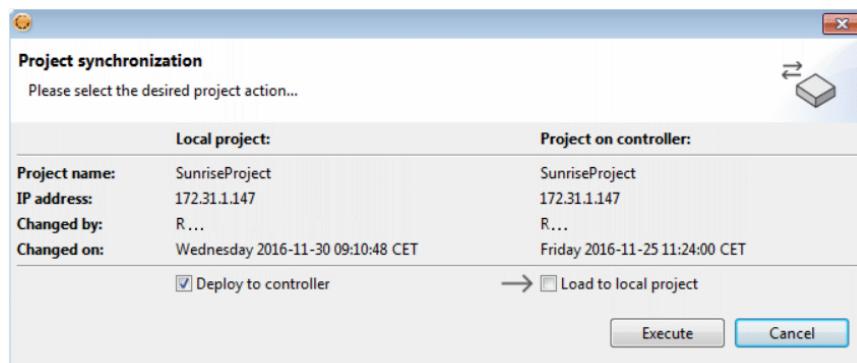


Fig. 9-10: Updating a project

Information is displayed for both projects. As standard, the direction of synchronization is set to transfer the more recent project version.

If modifications have been made to the project data on both sides, the system recognizes this as a conflict and displays a warning. The direction of synchronization can be set:

- Check mark activated at **Deploy to controller**: The project is transferred from Sunrise.Workbench to the robot controller.

- Check mark activated at **Load to local project**: The project is transferred from the robot controller to Sunrise.Workbench.
5. If required, change the direction of synchronization.
 6. Click on **Execute**.
 7. Only in the case of transfer to the robot controller: If the safety configuration or I/O configuration is modified, a dialog indicates that the robot controller must be rebooted in order to complete the synchronization.
 - Click on **OK** to transfer the project to the robot controller. Once the transfer is completed, the robot controller automatically reboots.
 - Transfer of the project can be stopped with **Cancel**.
 8. Only for transfer to the robot controller and if the option package Sunrise.RolesRights is used: The “**Authorization**” dialog opens. The required user group is preset.
Enter password and confirm with **OK**.
 9. The progress of the project transfer is displayed in a dialog both in Sunrise.Workbench and on the smartPAD. When the transfer is completed, the dialog is automatically closed. If the safety configuration or I/O configuration is modified, the robot controller is automatically rebooted.
If the transfer fails, a corresponding dialog is displayed both in Sunrise.Workbench and on the smartPAD. In addition, the cause of the error is displayed in Sunrise.Workbench.
Confirm the dialog in Sunrise.Workbench and on the smartPAD with **OK**.
 10. If the safety configuration is modified, activate this on the robot controller.

9.6 Loading the project from the robot controller

Description

A project can be loaded from the robot controller if the project is not located in the workspace of Sunrise.Workbench.

Precondition

- Network connection to the robot controller
- The workspace does not contain any project with the name of the project to be loaded.

Procedure

1. Select the menu sequence **File > New > Sunrise project**. The project creation wizard opens.
2. Enter the IP address of the robot controller from which the project is to be loaded in the **IP address of controller:** box.

The IP address of the robot controller can be displayed on the smartHMI. (>>> [6.20.6 "Displaying information about the robot and robot controller" Page 128](#))
3. Select the **Load project from controller** option.
4. Click on **Next >**. The system checks whether there is a project on the robot controller.

5. If there is a project is on the robot controller and there is no project with the same name in the workspace, a summary of information on the project is displayed.
Click on **Finish**. The project is created in the workspace and then displayed in the **Package Explorer**.

10 Station configuration and installation

10.1 Station configuration overview

Procedure

Open the station configuration:

1. In the **Package Explorer** view, open the node of the project that is to be configured.
2. Double-click on the file **StationSetup.cat**. The file is opened in the editor.

The file contains the station configuration of the project.

The station configuration can be edited and installed using the following tabs:

Topology

The **Topology** tab displays the hardware components of the station. The topology can be restructured or modified.

Software

The **Software** tab displays the software catalog of Sunrise.Workbench. Catalog elements to be installed or uninstalled in the project can be selected here.

The elements that can be selected depend on the topology and the option packages installed in Sunrise.Workbench.

Configuration

The **Configuration** tab displays the configuration of the robot controller. The configuration can be changed. The parameters that can be configured depend on the topology and the installed option packages.

- IP address and subnet mask of the robot controller
- IP address range for KUKA Line Interface (KLI)
- Handguiding Support
- General safety settings
- Parameters for calibration
- Type of media flange (if present on robot)
- Installation direction (default: floor-mounted installation)



The installation and use of option packages in the project may cause further parameters to be added.

Installation

The system software is installed on the robot controller via the **Installation** tab.

10.2 Adapting the network settings

Description

The development computer with KUKA Sunrise.Workbench is connected to the robot controller via the KLI. A prerequisite for communication via the network is that the IP addresses of the development computer and ro-

bot controller are in the same address range. For this, the network settings must be adapted.

Example:

- Robot controller (default): 172.31.1.147
- Laptop/PC: 172.31.1.148
- Subnet mask: 255.255.0.0

Procedure

1. Open the Network and Sharing Center via the Windows Control Panel or Windows Explorer.
2. In the top left-hand area, click the **Change adapter settings** entry. The network connections are displayed.
3. Right-click on **Local Area Connection** and select **Properties** in the context menu. The **Local Area Connection Properties** window is opened.
4. Double-click on **Internet Protocol Version 4 (TCP/IPv4)**. The **Internet Protocol Version 4 (TCP/IPv4) Properties** window is opened.
5. Carry out the desired IP address settings, e.g.:



Fig. 10-1: IP address settings of the LAN connection

6. Close the window by clicking on **OK**.

10.3 “Software” tab

10.3.1 Eliminating errors in the software catalog

Description

A software catalog containing errors prevents installation of the System Software on the robot controller. The errors must be eliminated before installation.

Possible causes of errors are:

- Missing reference to a catalog element

Some catalog elements are dependent on others. If a catalog element that is required by another one is deselected in the software catalog or removed by being uninstalled, the remaining catalog element is marked in red.

- Catalog element used, but not installed

If a catalog element that is not installed in Sunrise.Workbench is used in a project, this catalog element is indicated and marked in red.

Example

This example describes the options for elimination of errors.

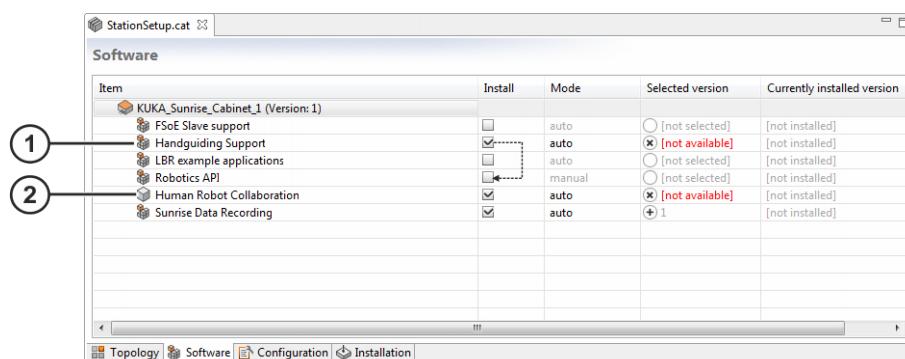


Fig. 10-2: Error display in the software catalog

- 1 Missing reference to a catalog element
- 2 Catalog element used, but not installed

Item	Description
1	<p>The catalog element Handguiding Support is not available because the catalog element Robotics API has been deselected.</p> <p>Possible remedies:</p> <ul style="list-style-type: none"> • Deselect the catalog element that is not available (deactivate check box) and save the station configuration. • Select the required catalog element (activate check box) and save the station configuration.
2	<p>The project uses functions of the safety option KUKA Sunrise.HRC. The catalog element Human Robot Collaboration is not available because the option is not installed in Sunrise.Workbench.</p> <p>Possible remedies:</p> <ul style="list-style-type: none"> • Deselect the catalog element that is not available (deactivate check box) and save the station configuration. • Install the safety option in Sunrise.Workbench (only necessary if the safety configuration has not yet been completed and AMFs of the safety option are required for the configuration).

10.4 “Configuration” tab

10.4.1 IP address range for KUKA Line Interface (KLI)

Description

The KLI is the Ethernet interface of the robot controller for external communication. In order for external PCs, e.g. the development computer with KUKA Sunrise.Workbench, to be able to connect to the robot controller via a network, the KLI must be configured accordingly.

The following IP address ranges are used internally by the robot controller.

- 169.254.*.*
- 192.*.*.*
- 172.16.*.*
- 172.17.*.*

If one or more KLI network devices (e.g. the robot controller, bus devices or other network devices) use IP addresses from one of these ranges, this IP address range must be set. Sunrise then reconfigures the internal network to ensure that there are no IP address conflicts.

Parameter	Description
IP address range for KUKA Line Interface	The following IP address ranges are available: <ul style="list-style-type: none">• 192.*.*.*• 172.16.*.*• 172.17.*.*• Other Default: Other

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:

- KUKA Sunrise.PROFINET M/S

10.4.2 Handguiding Support

Robots that have a hand guiding device with a safety-oriented enabling device can be guided manually if no application is selected or if an application is paused.

An application is paused if it has one of the following states:

- **Selected**
- **Motion paused**
- **Error**

Manual guidance is supported as standard in all operating modes except CRR mode. It is possible to configure manual guidance as not allowed in Test mode and/or Automatic mode.

Configuration parameters under **Handguiding Support**:

Parameter	Description
Enable manual guidance in Automatic mode	<p>Manual guidance in Automatic mode</p> <ul style="list-style-type: none"> • <i>True</i>: Manual guidance is allowed in Automatic mode. • <i>False</i>: Manual guidance is not allowed in Automatic mode. <p>Default: <i>True</i></p>
Enable manual guidance in the test modes	<p>Manual guidance in Test mode (T1, T2)</p> <ul style="list-style-type: none"> • <i>True</i>: Manual guidance is allowed in Test mode. • <i>False</i>: Manual guidance is not allowed in Test mode. <p>Default: <i>True</i></p>



In order to be able to use the motion command `handGuiding()` for manual guidance, the parameter **Enable manual guidance in Automatic mode** must be set to *False*. If the parameter is set to *True*, manual guidance is not possible for the running application.

10.4.3 General safety settings

The smartPAD can be configured as unpluggable.

Unplugging of the smartPAD is a safety function. The correct functioning of this safety function must be tested during initial start-up and recommissioning of the robot.

Configuration parameters under **General safety settings**:

Parameter	Description
smartPAD unplugging allowed	<p>Unplugging the smartPAD</p> <ul style="list-style-type: none"> • <i>True</i>: Unplugging of the smartPAD is allowed. The robot can be moved with the smartPAD unplugged. • <i>False</i>: Unplugging of the smartPAD is not allowed. The robot cannot be moved with the smartPAD unplugged. An EMERGENCY STOP is triggered. <p>Default: <i>True</i></p>



WARNING

If the smartPAD is disconnected, the system can no longer be switched off by means of the EMERGENCY STOP device on the smartPAD. If the smartPAD is configured as unpluggable, at least one external EMERGENCY STOP device must be installed that is accessible at all times.

Failure to observe this can lead to death, injury or property damage.



WARNING

Risk of fatal injury due to non-operational EMERGENCY STOP device

The user is responsible for ensuring that the decoupled smartPAD is removed from the system immediately. Measures must be taken to prevent operational and non-operational EMERGENCY STOP devices from being mixed up. Death, injuries or damage to property may result.

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

10.4.4 Configuration parameters for calibration

The parameters for calibration can be modified.

Configuration parameters under **smartHMI > Measurement**:

Parameter	Description
Minimum calibration point distance (tool) in mm	Minimum distance which must be maintained between 2 measuring points (XYZ 4-point and ABC 2-point methods) during tool calibration <ul style="list-style-type: none">• 0 ... 200 Default: 8
Maximum calculation error in mm	Maximum translational calculation error during tool calibration up to which the quality of the calibration is considered sufficient <ul style="list-style-type: none">• 0 ... 200 Default: 5
Minimum calibration point distance (base) in mm	Minimum distance which must be maintained between 2 measuring points during base calibration <ul style="list-style-type: none">• 0 ... 200 Default: 50
Minimum angle in °	Minimum angle to be maintained between the straight lines which are defined by the 3 measuring points during base calibration (3-point method) <ul style="list-style-type: none">• 0 ... 360 Default: 2.5

10.4.5 Configuration parameters for Backup Manager

If the KUKA Sunrise.BackupRestore option package is installed, the configuration parameters for the Backup Manager are available on the **Configuration** tab.

Configuration parameters under **BackupRestore**:

Parameter	Description
Automatic backup active/inactive	<p>Activation/deactivation of automatic backup</p> <ul style="list-style-type: none"> • Active: The robot controller automatically carries out backups. <p>The following parameters determine the time and the interval:</p> <ul style="list-style-type: none"> – Time [hh:mm]: Time of backup Default: 00:00 – Time interval [days]: Backup interval in days Default: 7 <p>Note: If the robot controller was switched off at the configured time, it carries out a data backup as soon as it is switched on at the next configured time. It only carries out one backup, even if the time was missed more than once.</p> <ul style="list-style-type: none"> • Inactive: No automatic backup. <p>Default: Inactive</p>
Backup mode	<p>Target and source directory for backups and restorations</p> <ul style="list-style-type: none"> • Local: The target directory for backups and the source directory for restorations is the directory D:\ProjectBackup on the robot controller. <p>Note: If the backup of the projects and user data takes up too much memory, the local memory may be full before the maximum configured number of backup copies has been reached. In this case, no further backup is possible.</p> <ul style="list-style-type: none"> • Network: The target directory for backups and the source directory for restorations is a network path: <ul style="list-style-type: none"> – Network path If during backup and/or restoration the robot controller must access the network and an authentication is required, the user name and password for the network path must be specified: <ul style="list-style-type: none"> – User name for network path – Password for network path <p>Note: Any other network path can be set on the robot controller for restorations.</p> <p>Default: Local</p>
Maximum number of backups	<p>Maximum number of backup copies</p> <ul style="list-style-type: none"> • 1 ... 50 <p>Once the maximum number of backup copies has been reached, the oldest backup copy is overwritten.</p> <p>If more backup copies than the permissible number are present, e.g. because the maximum number has been reduced, the excess backup copies will be deleted next time a backup is made (starting with the oldest).</p> <p>Default: 1</p>

Parameter	Description
Restore-configuration file	<p>Path to a file with network configurations for restorations</p> <p>The file must be present in CSV format and copied manually to the robot controller.</p> <p>Note: It is advisable to save the file on drive D:\. If it is saved on C:\, it is not possible to rule out the possibility of it being overwritten in the case of a restoration or installation.</p>

CSV file

Network configurations for restorations can be entered in a CSV file and saved on the robot controller. The data set with the network configurations can then be loaded using the Backup Manager and the source directory from which the data are to be restored can be selected.

Example of a CSV file:

```
IP_adress;subnetmask;BM_Username;BM_Password;BM_ProjectRestor
eDirectoryPath;Server
192.168.0.131;255.255.0.0;User41;pwd82p;\Server\Path
\Restore;Restore3Backup857
192.168.0.239;255.255.0.0;User66;pwd24ppp;\Server\Path
\Restore;Restore0Remote415
192.168.0.151;255.255.0.0;User38;pwd75ppp;\Server\Path
\Kontakt;KontaktRestore705
...
...
```

The following points must be observed when creating the CSV file:

- The header data set must contain the columns specified in the example file.
- The column names must not be modified.
- The columns can be saved in any order.
- Further columns can be added, e.g. to save additional information.

10.5 “Installation” tab

10.5.1 Installing System Software on the robot controller

Description

During installation, all configuration data relevant for operation of the robot are transferred from Sunrise.Workbench to the robot controller. These include:

- Station configuration
- Safety configuration
- Passwords for user groups

The following points must be observed during installation:

- The robot type and media flange (if present) set in the station configuration must match the robot connected to the robot controller (see identification plate). If the data do not match, the robot cannot be moved after installation.
- The safety configuration is not yet active after installation. The robot cannot be moved until the safety configuration has been activated.
(>> [13.13.1 "Activating the safety configuration" Page 274](#))

Reinstallation

If the station configuration or the password for a user group on the robot controller changes, installation must be carried out again:

- Change to the station configuration on the **Topology** tab
- Change to the station configuration on the **Software** tab

Examples:

- Installation of additional option packages
 - Incompatible version changes of existing software packages
- Incompatible version changes can occur if a project that was created with an older version of Sunrise.Workbench is loaded into the workspace.



Software updates may result in undesired modifications to the Sunrise project. If the robot controller is reinstalled following a software update, all applications must therefore be tested in Manual Reduced Velocity mode (T1).

- Change to the station configuration on the **Configuration** tab
- Change of password for a user group on the robot controller

Precondition

- Network connection to the robot controller
- The station configuration is completed.

Procedure

1. Select the **Installation** tab.

As standard, the **Installation events** window displays the warnings and errors which occur during installation (check mark next to **Show only warnings and errors.**).

2. If all events which occur during installation are to be displayed, remove the check mark next to **Show only warnings and errors..**
3. Click on **Install**. The installation is prepared and the **Installation** window opens.

The **Configured IP** box is marked in color:

- Marked in green: Network connection is present. Continue with step 5.
- Marked in red: Network connection is not present.

Possible causes include:

- The network cable is not connected correctly.
- The configured IP address does not match the IP address of the robot controller.

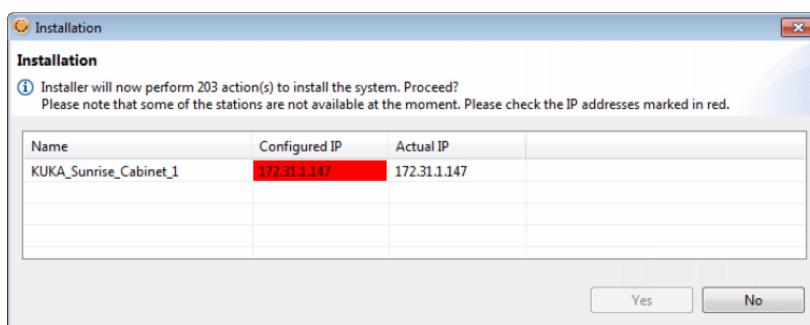


Fig. 10-3: The robot controller cannot be reached

4. Only if the **Configured IP** box is marked in red:
 - If the configured IP address matches the actual address of the robot controller, there is no network connection to the robot controller. Establish network connection.
 - If the IP address of the robot controller is different from the configured address, enter the current IP address of the robot controller in the **Actual IP** box. To do this, double-click in the box.
5. To continue the installation, click on **OK**.
6. Only relevant if the IP address in the **Actual IP** box has been changed: If the red marking under **Configured IP** persists or the installation fails, there is no network connection to the robot controller with this IP address.
Establish a network connection and restart the installation process (return to step 3).
7. Confirm the reboot prompt with **OK**. The robot controller is rebooted and the installation is completed.

10.6 Loading an old project and converting the safety configuration

Description

If a new software version of Sunrise.Workbench is installed, a Sunrise project which was created with an earlier software version can be loaded to the workspace and continue to be used.

The station configuration changes when the Sunrise project is loaded. Saving the station configuration will transfer the corresponding safety configuration to the new version.

Error message

The following error message may be generated on saving the station configuration:

Not all parameters could be converted to the current version of the safety configuration. Please check that the safety configuration is complete and correct before installation.

This message is displayed if safety settings have been added or removed in the new software version, e.g.:

- **Allow muting via input** (available from software version 1.10 or higher)
- **Allow external position referencing** (available from software version 1.11 or higher)

- Safety-oriented workpieces no longer configurable as object templates (software version 1.12 or higher)
The message is only displayed if at least one safety-oriented workpiece is configured in the loaded project. Instead, only the mass of the heaviest workpiece now needs to be specified in the safety-oriented project settings.

Precondition

- The Sunrise project is archived or saved in any directory.
- New version of Sunrise.Workbench is installed.
- User group **Safety maintenance technician**

Procedure

1. Load the Sunrise project into the workspace.
2. Open the station configuration of the project (file **StationSetup.cat**) and click on **Save**.
3. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
4. In the case of an error message indicating that not all parameters could be converted to the current version of the safety configuration: Confirm the error message with **OK**.
5. The safety configuration is updated and its parameters are converted. When the operation is completed, this is indicated by a message. Confirm with **OK**.
6. Further steps are required in order to be able to use the updated project on the robot controller:
 - a. In the case of an error message indicating that not all parameters could be converted to the current version of the safety configuration: Check that the safety configuration is complete and correct and modify as required.
 - b. Install the system software on the robot controller.
 - c. Transfer the updated project to the robot controller by means of project synchronization.
 - d. Reactivate the safety configuration.
 - e. Carry out safety acceptance.

10.7 Option packages

The functionalities of the following option packages are described in this documentation:

- KUKA Sunrise.AntiVirus
Virus scanner for protection against viruses
(>>> [10.7.2 "Installing or updating the virus scanner" Page 205](#))
- KUKA Sunrise.BackupRestore
Backup manager for backing up and restoring data
(>>> [6.21 "Backup Manager" Page 128](#))
The properties of the Backup Manager can be configured.
(>>> [10.4.5 "Configuration parameters for Backup Manager" Page 198](#))

- KUKA Sunrise.RolesRights
Expansion of the user groups available as standard to include the user group **Expert**
(>>> [9.4.1 "Overview of Sunrise.RolesRights" Page 185](#))
- KUKA Sunrise.SafeOperation
Safety option with additional safety monitoring functions, e.g. velocity monitoring functions or Cartesian workspace monitoring functions
- KUKA Sunrise.HRC
Safety option with additional safety monitoring functions for HRC applications, e.g. collision detection or TCP force monitoring
- KUKA Sunrise.EnhancedVelocityControl
Option for limitation of Cartesian velocities
(>>> [17 "KUKA Sunrise.EnhancedVelocityControl" Page 549](#))

10.7.1 Installing the option package

Description

If an option package is supplied together with KUKA Sunrise.Workbench, it is automatically installed during installation of Sunrise.Workbench. If Sunrise.Workbench is already installed, the option package must be installed subsequently.

Option packages are provided as ZIP archives for installation. The archive names have the following composition:

- *Article number;revision number (2-digit);product name;build version*

Installation is carried out in 3 parts:

- Installation of the option in Sunrise.Workbench
Depending on the option, the software catalog of Sunrise.Workbench is expanded by one or more entries.
- Selection of the new software for installation in the station configuration of the project
- Installation of the system software on the robot controller
Once the robot controller has been rebooted, the new software is available for the station.

Precondition

- Local administrator rights
- Sunrise.Workbench is installed.
- The option package is available as a ZIP archive.

Procedure

1. Select the menu sequence **Help > Install new software ...**. The **Install** window is opened.
2. To the right of the **Work with** box, click on **Add** The **Add repository** window is opened.
Alternatively: Drag the ZIP archive of the option into the window, then continue with step 5.
3. Click on **Archive ...**, navigate to the directory in which the ZIP archive of the option is located and select the archive.
4. Confirm your selection with **Open**. The **Position** box now displays the installation path. Confirm the path with **OK**.

5. In the **Install** window, the installation path is adopted in the **Work with box**.
The window now also displays a check box with the name of the option. Activate the check box.
6. Leave the other settings in the **Install** window as they are and click on **Next >**.
7. An installation details overview is displayed. Click on **Next >**.
8. A license agreement is displayed. In order to be able to install the software, the agreement must be accepted. Then click on **Finish**. The installation is started.
9. A safety warning concerning unsigned contents is displayed. Confirm with **OK**.
10. A message indicates that Sunrise.Workbench must be restarted in order to apply the changes. Click on **Restart now**.
11. Sunrise.Workbench restarts. This completes installation in Sunrise.Workbench.
12. Open the station configuration of the desired project (file **StationSetup.cat**). The new software entries are displayed on the **Software** tab.
13. If the check mark is set in the **Install** column for the new entries, the new software has automatically been selected for installation.
If not, set the check mark for the new entries.
14. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
15. Install the system software on the robot controller. Once the robot controller has been rebooted, the new software is available for the station.

10.7.2 Installing or updating the virus scanner

Description

Once the virus scanner has been installed on the robot controller, a tile for the virus scanner is available on the smartHMI. This tile can be used, for example, to display the version of the installed virus scanner and messages about viruses that have been found.

(>>> *20.4 "Displaying messages of the virus scanner" Page 603*)



It is advisable to check what version is currently installed on the robot controller before updating the virus scanner. Do not perform a downgrade.

Precondition

- Local administrator rights
- Sunrise.Workbench is installed.
- The option package is available as a ZIP archive.
- In the case of an update on the robot controller: Network connection without Internet access or with an active firewall
During the update, the virus scanner is briefly inactive.

Procedure

(>>> *10.7.1 "Installing the option package" Page 204*)

10.7.3 Installing a language package

Description

The user interface on the smartHMI is available in the following languages:

Language	Language abbreviation
Chinese (simplified)	zh
Danish	da
German	de
English	en
Finnish	fi
French	fr
Greek	el
Italian	it
Japanese	ja
Korean	ko
Dutch	nl
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Swedish	sv
Slovak	sk
Slovenian	sl
Spanish	es
Czech	cs
Turkish	tr
Hungarian	hu

Languages which are only available after software is delivered can be installed later if required.

Precondition

- Local administrator rights
- Sunrise.Workbench is installed.
- The option package is available as a ZIP archive.

Procedure

(>>> [10.7.1 "Installing the option package" Page 204](#))

10.7.4 Uninstalling the option package

Description

Option packages that are no longer required can be uninstalled in Sunrise.Workbench.



It is advisable to archive all relevant data before uninstalling a software package.

Precondition

- Option package has been removed from the robot controller.

Procedure

1. Select the menu sequence **Help > Install new software ...**. The **Install** window is opened.
2. Click on the link by **What is already installed?**. The **Installation details for Sunrise Workbench** window is opened.
3. Select the **Installed software** tab (if it is not already selected).
4. In the list of installed software, select the option that is no longer required.



It is possible to select several options simultaneously and uninstall them together.

5. Click on **Uninstall**. The **Uninstall** window is opened. The details of the software to be uninstalled can be viewed here.
6. Click on **Finish**. The uninstallation is started.
A progress bar indicates the progress of the uninstallation.
7. A message indicates that Sunrise.Workbench must be restarted in order to apply the changes. Click on **Restart now**.
8. Sunrise.Workbench restarts. This completes uninstallation in Sunrise.Workbench.

10.7.4.1 Instructions for uninstallation of safety options

The following points must be observed when uninstalling safety options:

- Following uninstallation, the AMFs provided by the uninstalled safety option are no longer available in newly created projects.
- In the case of existing projects, an AMF that is no longer available is only displayed in the selection table if a cell that uses the AMF is selected in the safety configuration.
- The safety configuration of existing projects is retained, even if it uses AMFs of an uninstalled safety option. The user can continue to use it without restrictions.
- If the existing safety configuration uses AMFs of an uninstalled safety option, it can no longer be modified. Saving of the configuration is prevented.

10.7.4.2 Removing an option package from the robot controller

Description

In order to remove an option package from the robot controller, the system software must be reinstalled.

Precondition

- Network connection to the robot controller
- No application is running on the robot controller.

Procedure

1. In Sunrise.Workbench, open the station configuration of the project on the robot controller (file **StationSetup.cat**).
2. On the **Software** tab, deactivate the check box for the option package that is to be removed.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
4. Install the system software on the robot controller.

Once the robot controller has been rebooted, the option package is no longer present on the robot controller.

11 Bus configuration

11.1 Overview: Configuration and I/O mapping in WorkVisual

Step	Description
1	<p>Install the Sunrise option package in WorkVisual.</p> <p>The option package is available as a KOP file and is supplied together with Sunrise.Workbench (file Sunrise.kop in the directory WorkVisual AddOn).</p> <p>Note: The option package supplied with Sunrise.Workbench must always be used. If an old version of Sunrise.Workbench is uninstalled and a new version installed, the option package must also be exchanged in WorkVisual.</p>
2	<p>Terminate WorkVisual and create a new I/O configuration in Sunrise.Workbench or open an existing I/O configuration. WorkVisual is started automatically and the WorkVisual project corresponding to the I/O configuration is opened.</p> <p>(>>> 11.3 "Creating a new I/O configuration" Page 210)</p> <p>(>>> 11.4 "Opening an existing I/O configuration" Page 210)</p>
3	<p>Only necessary if devices are used for which no device description files have yet been imported:</p> <ol style="list-style-type: none"> 1. Close the WorkVisual project. 2. Import the required device description files. 3. Reopen the WorkVisual project.
4	<p>Configure the field bus.</p> <p>(>>> 11.2 "Overview of field buses" Page 209)</p>
5	<p>Create the Sunrise I/Os and map them.</p> <p>(>>> 11.5 "Creating Sunrise I/Os" Page 211)</p> <p>(>>> 11.6.3 "Mapping Sunrise I/Os" Page 219)</p>
6	<p>Export the I/O configuration to the Sunrise project.</p> <p>(>>> 11.7 "Exporting the I/O configuration to the Sunrise project" Page 219)</p>
7	<p>Transfer the I/O configuration to the robot controller by means of project synchronization and reboot the robot controller.</p> <p>(>>> 9.5 "Project synchronization" Page 187)</p>



Information about installing and managing option packages can be found in the **WorkVisual** documentation.



Information about importing device description files and general information about configuring field buses can be found in the **WorkVisual** documentation.

11.2 Overview of field buses

The following field buses are supported by Sunrise and can be configured with WorkVisual:

Field bus	Description
PROFINET	An Ethernet-based field bus. Data exchange is carried out on a client-server basis. PROFINET is installed on the robot controller.
PROFIBUS	Universal field bus which enables communication between devices from different manufacturers without special interface adaptations. Data exchange is carried out on a master-slave basis.
EtherCAT	An Ethernet-based field bus suitable for real-time requirements.



For configuration of a field bus, the documentation of the field bus is required.



The I/O configuration is created automatically for the media flange set in the project. If a media flange with an EtherCAT output (e.g. media flange IO pneumatic) is used and additional EtherCAT devices are connected, these must be configured using WorkVisual.



When connecting additional EtherCAT devices to a media flange with an EtherCAT output, e.g. media flange IO pneumatic, it must be ensured that the number of available signals on the bus is limited. If there are too many connected devices, this can result in overloading of the bus and loss of communication. Under certain circumstances, the robot can then no longer be moved.



If the robot controller is used as a PROFINET master or device, hardware problems can result in an inability to access bus devices. In this case, use of a diagnostic tool, such as WorkVisual, Step 7 or Wireshark, is recommended.

11.3 Creating a new I/O configuration

Precondition

- Sunrise project without I/O configuration

Procedure

1. Select the project in the **Package Explorer**.
2. Select the menu sequence **File > New > I/O configuration**.

WorkVisual is started and the WorkVisual project corresponding to the I/O configuration is opened. The file **IOConfiguration.wvs** is inserted into the Sunrise project; this can be used to call the corresponding WorkVisual project.

11.4 Opening an existing I/O configuration

Precondition

- Sunrise project with I/O configuration

Procedure

1. Double-click on the file **IOConfiguration.wvs**. WorkVisual is started and the WorkVisual project corresponding to the I/O configuration is opened.

2. Right-click on the inactive robot controller on the **Hardware** tab in the **Project structure** window.
3. Select **Set as active controller** from the context menu. The **I/O Mapping** window opens. The Sunrise I/Os can be edited.

11.5 Creating Sunrise I/Os

Precondition

- Field bus configuration has been completed.
- The robot controller has been set as the active controller.

Procedure

1. Select an input or output module of the configured bus on the **Field buses** tab in the top right-hand corner of the **I/O Mapping** window.
(>>> [11.6.1 "I/O Mapping window" Page 217](#))
2. Select the **Sunrise I/Os** tab in the top left-hand corner of the **I/O Mapping** window.
3. In the bottom left-hand corner of the **I/O Mapping** window, click on the **Creates signals at the provider** button. The **Create I/O signals** window is opened.
(>>> [11.5.1 ""Create I/O signals" window" Page 212](#))
4. Create an I/O group and inputs/outputs within the group.
(>>> [11.5.2 "Creating an I/O group and inputs/outputs within the group" Page 214](#))
5. Click on **OK**. The Sunrise I/Os are saved. The **Create I/O signals** window is closed.

The created I/O group is displayed on the **Sunrise I/Os** tab of the **I/O Mapping** window. The signals can now be mapped.

(>>> [11.6.3 "Mapping Sunrise I/Os" Page 219](#))

11.5.1 “Create I/O signals” window

Overview

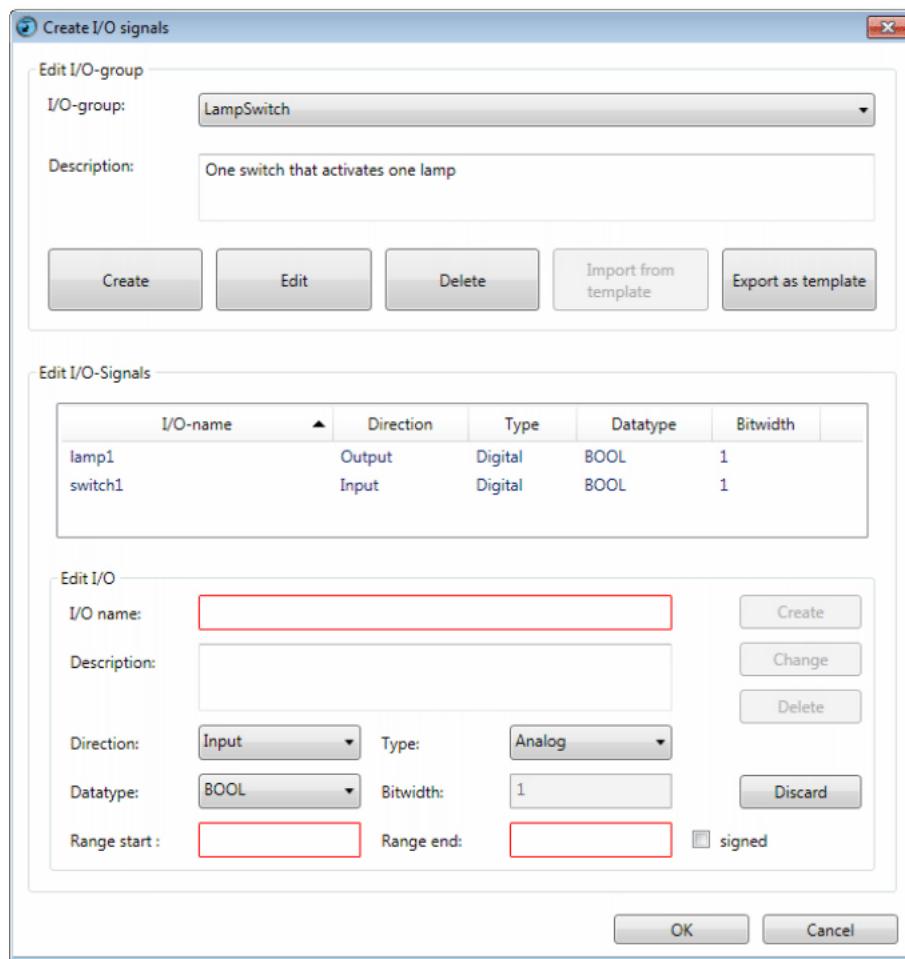


Fig. 11-1: “Create I/O signals” window

The window for creating and editing the Sunrise I/Os and Sunrise I/O groups consists of the following areas:

Area	Description
Edit I/O group	In this area, I/O groups are created and edited. It is also possible to save I/O groups as a template or to import previously created templates.
Edit I/O signals	In this area, the input/output signals of an I/O group are displayed.
Edit I/O	In this area, the inputs/outputs of an I/O group are created and edited.



Input boxes are displayed with a red frame if values must be entered or if incorrect values have been entered. A help text is displayed when the mouse pointer is moved over the box.

Signal properties

In the **Edit I/O** area, new signals can be created and the signal properties defined:

Property	Description
I/O name	Name of the input/output
Description	Description for the input/output (optional)
Direction	<p>Signal direction</p> <ul style="list-style-type: none"> • Input: Signal is an input. • Output: Signal is an output.
Type	<p>Signal type</p> <ul style="list-style-type: none"> • Analog: Signal is an analog signal. • Digital: Signal is a digital signal.
Data type	<p>Data type of the signal</p> <p>In WorkVisual, a total of 15 different data types are available for selection. For use with Java, these data types are mapped to the following data types:</p> <ul style="list-style-type: none"> • integer, long, double, boolean
Bit width	<p>Number of bits that make up the signal. With the data type BOOL, the bit width is always 1.</p> <p>Note: The value must be a positive integer which does not exceed the maximum permissible length of the selected data type.</p>

The following signal properties are only relevant for analog inputs/outputs:



The values to be set for analog inputs/outputs can generally be found in the data sheet of the field bus module used.

Property	Description
Start range	<p>The smallest possible value of an analog connection without a physical unit. This is the value to which the smallest possible number that can be generated on the bus is mapped.</p> <p>Note: The start range must be lower than the end range. It is also possible to enter decimal values.</p>
End range	<p>The largest possible value of an analog connection without a physical unit. This is the value to which the largest possible number that can be generated on the bus is mapped.</p> <p>Note: The end range must be greater than the start range. It is also possible to enter decimal values.</p>
Signed	<p>Defines whether the number generated on the bus is interpreted as signed or unsigned.</p> <ul style="list-style-type: none"> • Check box active: Signed • Check box not active: Unsigned

Property	Description
	<p>Examples:</p> <ul style="list-style-type: none"> • In the case of an analog output module with a measurement range from 0 to 10 V, 0 must be specified for the start range and +10 for the end range. • In the case of an analog input module with a measurement range from 4 to 20 mA, 4 must be specified for the start range and 20 for the end range. • In the case of an analog input module with a measurement range of +/-10 V plus 1.76 V overflow, -11.76 must be specified for the start range and +11.76 for the end range.

11.5.2 Creating an I/O group and inputs/outputs within the group

Precondition

- The **Create I/O signals** window is open.

Procedure

1. In the **Edit I/O group** area, click on **Create**.
The **Create I/O group** window is opened.

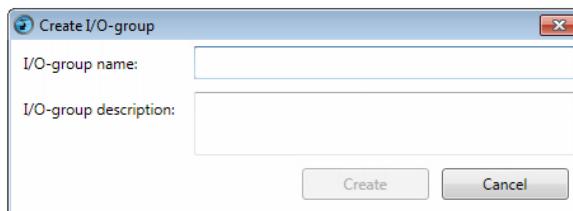


Fig. 11-2: Create I/O group

2. Enter a name for the I/O group.
3. Enter a description for the I/O group (optional).

 It is advisable to enter a description in all cases. This description is displayed later as a help text in the robot application and on the smartHMI.
4. Click on **Create**. The I/O group is created and displayed in the selection menu **I/O group**.
5. In the **Edit I/O** area, enter a name for the input or output of the group and define the signal properties.
(>>> *"Signal properties" Page 212*)
6. In the **Edit I/O group** area, click on **Create**. The input or output signal is created and displayed in the **Edit I/O Signals** area.
7. Repeat steps 5 and 6 to define further inputs/outputs in the group.

11.5.3 Editing an I/O group

Precondition

- The **Create I/O signals** window is open.
- The inputs/outputs of the group are not mapped.

Procedure

1. Select the desired I/O group from the **I/O group** selection menu.
2. Click on **Edit**. The **Rename I/O group** window is opened.
3. Change the name of the I/O group and/or the corresponding description (optional). Confirm with **Apply**.

11.5.4 Deleting an I/O group**Precondition**

- The **Create I/O signals** window is open.
- The inputs/outputs of the group are not mapped.

Procedure

1. Select the desired I/O group from the **I/O group** selection menu.
2. Click on **Delete**. If signals have already been created for the I/O group, a request for confirmation is displayed.
3. Reply to the request for confirmation with **Yes**. The I/O group is deleted.

11.5.5 Changing an input/output of a group**Precondition**

- The **Create I/O signals** window is open.
- The signals that are to be changed are not mapped.

Procedure

1. Select the I/O group of the signal from the **I/O group** selection menu.
 2. In the **Edit I/O Signals** area, click on the desired input or output.
 3. In the **Edit I/O** area, edit the signal properties as required.
(>>> *"Signal properties" Page 212*)
-  All the changes can be discarded by clicking on the **Discard** button.
4. Click on **Change**. The changes are saved.

11.5.6 Deleting an input/output of a group**Precondition**

- The **Create I/O signals** window is open.
- The signals that are to be deleted are not mapped.

Procedure

1. Select the I/O group of the signal from the **I/O group** selection menu.
2. In the **Edit I/O Signals** area, click on the desired input or output.
3. Click on **Delete**.

11.5.7 Exporting an I/O group as a template

Description

I/O groups can be saved as a template. The template contains all the inputs/outputs belonging to the saved I/O group. This enables I/O groups, once created, to be reused. The mapping of the inputs and outputs is not saved, however.

After exporting the template, the templates created in WorkVisual are available in Sunrise.Workbench in the **IOTemplates** folder of the project.

Precondition

- The **Create I/O signals** window is open.

Procedure

1. In the **Edit I/O group** area, select the I/O group that is to be exported as a template.
 2. Click on **Export as template**. The **Save I/O group as template** window is opened.
 3. Enter a name for template.
-
- If a template with the same name already exists in the Sunrise project, it will be overwritten during the export operation.
4. Enter a description for the template (optional).
 5. Click on **Export**. The template is saved.

11.5.8 Importing an I/O group from a template

Precondition

- The **Create I/O signals** window is open.
- There is at least one I/O group available in Sunrise.Workbench as a template in the Sunrise project.

Procedure

1. In the **Edit I/O group** area, click on **Import from template**. The **Import I/O group from template** window is opened.
2. In the selection list **Used template**, select the template to be imported.
3. Enter a name in the **I/O-group name** box for the I/O group to be created.
4. Click on **Import**. An I/O group configured in accordance with the template is imported and can be edited.

11.6 Mapping the bus I/Os

11.6.1 I/O Mapping window

Overview

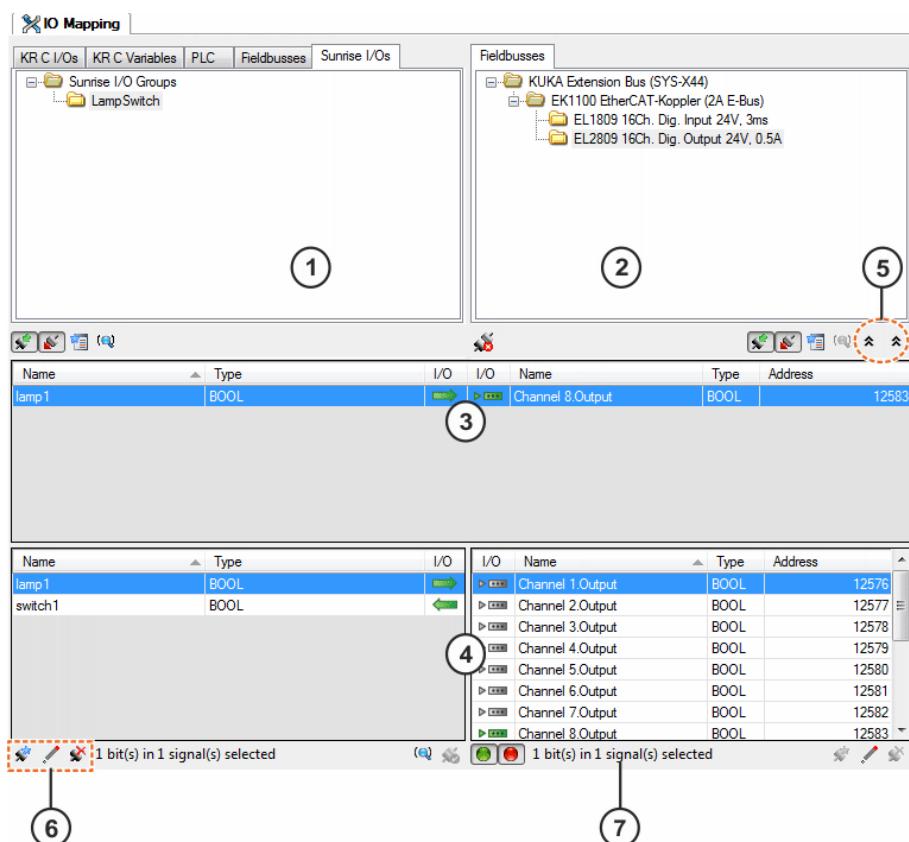


Fig. 11-3: “I/O Mapping” window

Item	Description
1	Displays the Sunrise I/O groups The signals in the I/O group selected here are displayed in the overviews lower down.
2	Displays the inputs/outputs of the bus modules The signals in the module selected here are displayed in the overviews lower down.
3	Connection overview Displays the mapped signals. These are the signals of the I/O group selected under Sunrise I/Os , which are mapped to the bus module selected under Field buses .
4	Signal overview Here the signals can be mapped. (>> 11.6.3 "Mapping Sunrise I/Os" Page 219)

Item	Description
5	The arrow buttons allow the connection and signal overviews to be collapsed and expanded independently of one another. <ul style="list-style-type: none"> • Collapse connection view (left-hand arrow symbol pointing up) • Expand connection view (left-hand arrow symbol pointing down) • Collapse signal view (right-hand arrow symbol pointing up) • Expand signal view (right-hand arrow symbol pointing down)
6	Buttons for creating and editing the Sunrise I/Os
7	Displays how many bits the selected signals contain.



For the I/O mapping in Sunrise, only the **Sunrise I/Os** and **Field buses** tabs are relevant.

11.6.2 Buttons in the “I/O Mapping” window

Some of these buttons are displayed in several places. In such cases, they refer to the side of the **I/O Mapping** window on which they are located.

Edit

Button	Name/description
	Creates signals at the provider Opens the Create I/O signals window. (>>> 11.5.1 “Create I/O signals” window Page 212) The button is only active if an input or output module is selected on the Field buses tab and a signal of the I/O group is selected in the signal overview.
	Edit signals at the provider Opens the Edit I/O signals window. The button is only active if an I/O group is selected on the Sunrise I/Os tab and a signal of the I/O group is selected in the signal overview.
	Deletes signals at the provider Deletes all the selected inputs/outputs. If all the inputs/outputs of a group are selected, the I/O group is also deleted. The button is only active if an I/O group is selected on the Sunrise I/Os tab and a signal of the I/O group is selected in the signal overview.

Mapping

Button	Name/description
	Disconnect Disconnects the selected mapped signals. It is possible to select and disconnect a number of connections simultaneously.
	Connect Connects signals which are selected in the signal overview.

11.6.3 Mapping Sunrise I/Os

Description

This procedure is used to map the Sunrise I/Os to the inputs/outputs of the field bus module. It is only possible to map inputs to inputs and outputs to outputs if they are of the same data type. For example, it is possible to map BOOL to BOOL or INT to INT, but not BOOL to INT or BYTE.

Precondition

- The robot controller has been set as the active controller.

Procedure

- On the **Sunrise I/Os** tab in the left-hand half of the window, select the I/O group for which the I/Os are to be mapped.
The signals of the group are displayed in the bottom area of the **I/O Mapping** window.
- On the **Field buses** tab in the right-hand half of the window, select the desired input or output module.
The signals of the selected field bus module are displayed in the bottom area of the **I/O Mapping** window.
- Drag the signal of the group onto the input or output of the module.
(Or alternatively, drag the input or output of the device onto the signal of the group.)
The signals are now mapped. Mapped signals are indicated by green arrows.

Alternative procedure for mapping:

- Select the signals to be mapped and click on the **Connect** button.

11.7 Exporting the I/O configuration to the Sunrise project

Description

When exporting an I/O configuration from WorkVisual, a separate Java class is created for each I/O group in the corresponding Sunrise project. Each of these Java classes contains the methods required for programming, in order to be able to read the inputs/outputs of an I/O group and write to the outputs of an I/O group.

The classes and methods are saved in the Java package **com.kuka.generated.ioAccess** in the source folder of the Sunrise project.



The source code of the Java classes of the package **com.kuka.generated.ioAccess** must not be changed manually. To expand the functionality of an I/O group, it is possible to derive further classes from the classes created or to continue to use objects from these classes, e.g. as arrays of their own classes (aggregating).

The structure of the Sunrise project after exporting an I/O configuration is described here:

(>>> *15.11 "Using inputs/outputs in the program" Page 415*)

Precondition

- The robot controller has been set as the active controller.
- The automatic change recognition is activated in Sunrise.Workbench.
(>>> *5.9 "Activating the automatic change recognition" Page 73*)

Procedure

1. Select the menu sequence **File > Import / Export**. The import/export wizard for files opens.
2. Select **Export the I/O configuration to the Sunrise Workbench project..**
3. Click on **Next >** and then on **Finish**. The configuration is exported to the Sunrise project.
4. A message is displayed as to whether the export was successfully completed. If Sunrise I/Os have not been mapped, this is also indicated.



It is not essential to map all the Sunrise I/Os that have been created.

Click **Close** to terminate the wizard.

5. Close WorkVisual by selecting **File > Exit**.

12 External control

12.1 Overview of external controller

If the processes of the station are to be controlled by an external controller in Automatic mode, the Sunrise project on the robot controller must be configured for external control.

Default application

A default application must be assigned to every project that is to be controlled externally.

The default application has the following characteristics:

- It is automatically selected when the operating mode is switched to Automatic.
- It can only be started via the input signal **App_Start** (not by means of the Start key on the smartPAD).
- It cannot be deselected again in Automatic mode.

Interfaces

External controller and robot controller can communicate via the following interfaces:

- I/O system of the robot controller
- Network protocol UDP

The input/output signals for communication are predefined:

- The external controller can start, pause and resume the default application via the input signals.
(>>> [12.4 "External controller input signals" Page 222](#))
- The output signals can be used to provide information about the status of the default application and the station to the external controller.
(>>> [12.5 "External controller output signals" Page 223](#))

Precondition

In order to be able to start an application, the following preconditions must be met:

- The robot is mastered (all axes).
- A Sunrise project has been configured for external control.
- AUT mode
- If configured: the input signal **App_Enable** has a HIGH level or is TRUE.
- The motion enable signal is present.

12.2 Configuring the external controller via the I/O system

The following steps are required for configuring the external controller via the I/O system:

Step	Description
1	Configure and map inputs/outputs for communication with the external controller in WorkVisual. (>>> 12.4 "External controller input signals" Page 222) (>>> 12.5 "External controller output signals" Page 223)
2	Export I/O configuration from WorkVisual to Sunrise.Workbench.
3	Create the default application for the external controller.
4	Configure the external controller in the project settings. (>>> 12.7 "Configuring the external controller in the project settings" Page 225)
5	Transfer the project to the robot controller by means of synchronization.



The physical inputs/outputs used for communication with the external controller must not be multiply mapped.

12.3 Configuring the external controller via the UDP interface

The following steps are required for configuring the external controller via the UDP network protocol:

Step	Description
1	Create the default application for the external controller.
2	Configure the external controller in the project settings. (>>> 12.7 "Configuring the external controller in the project settings" Page 225)
3	Transfer the project to the robot controller by means of synchronization.

Use of the UDP is illustrated by the following example:

(>>> [12.9 "External control via UDP – Start-up example" Page 231](#))

12.4 External controller input signals

App_Start

The input signal is absolutely vital for an externally controlled project.

The default application is started and resumed in Automatic mode by the external controller by means of a rising edge of the signal (change from FALSE to TRUE).

App_Enable

The input signal is optionally configurable.

The default application can be paused by the higher-level controller in Automatic mode using this signal. For this, the signal must have a LOW level or be FALSE.

Get_State

The input signal is only available if the UDP interface is used.

The external controller can use this signal to request application and station statuses from the robot controller. The value of the signal can be TRUE or FALSE.

System response

The input signal **App_Enable** has a higher priority than the input signal **App_Start**. If the input signal **App_Enable** is configured, the default application can only be started if **App_Enable** has a HIGH level or is TRUE.

The following table describes the system behavior when the **App_Enable** signal is configured.

App_Start	App_Enable	Application status	Reaction
FALSE --> TRUE	FALSE	Selected	None
FALSE --> TRUE	FALSE	Motion paused	None
FALSE --> TRUE	TRUE	Selected	Application is started.
FALSE --> TRUE	TRUE	Motion paused	Application is resumed.
			If the path is left: the robot is repositioned. The application is then paused.
Any	TRUE --> FALSE	Running	Application is paused.
Any	TRUE --> FALSE	Repositioning	Application is paused.

12.5 External controller output signals

The configuration of these output signals is optional.

AutExt_Active

The output signal has a HIGH level or is TRUE if Automatic mode is active and the project on the robot controller can be controlled externally via the interface.

AutExt_AppReadyToStart

The output signal has a HIGH level or is TRUE if the default application is ready to start.

The application is ready to start in the following states:

- **Selected**
- **Motion paused**

DefaultApp_Error

The output signal has a HIGH level or is TRUE if an error occurred when the default application was run.

Station_Error

The output signal has a HIGH level or is TRUE if the station is in an error state.

There is an active error state in the following cases:

- Motion enable signal is not present.

- Drive error or bus error active.
- At least one robot axis is not mastered and the operating mode is not set to T1.

NOTICE

It is not permissible to set outputs in a robot application that signal system states to the external controller. Failure to observe this precaution may result in malfunctioning of the external controller and damage to property.

12.6 Signal diagrams

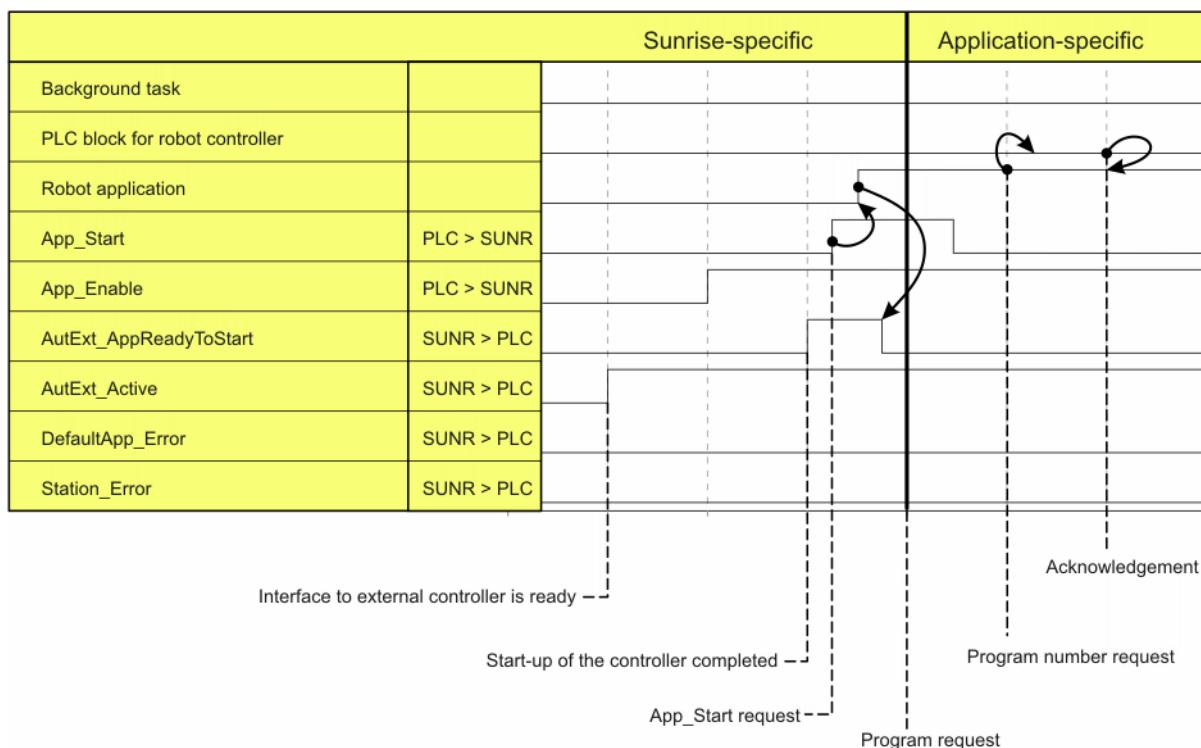


Fig. 12-1: Automatic system start and normal operation

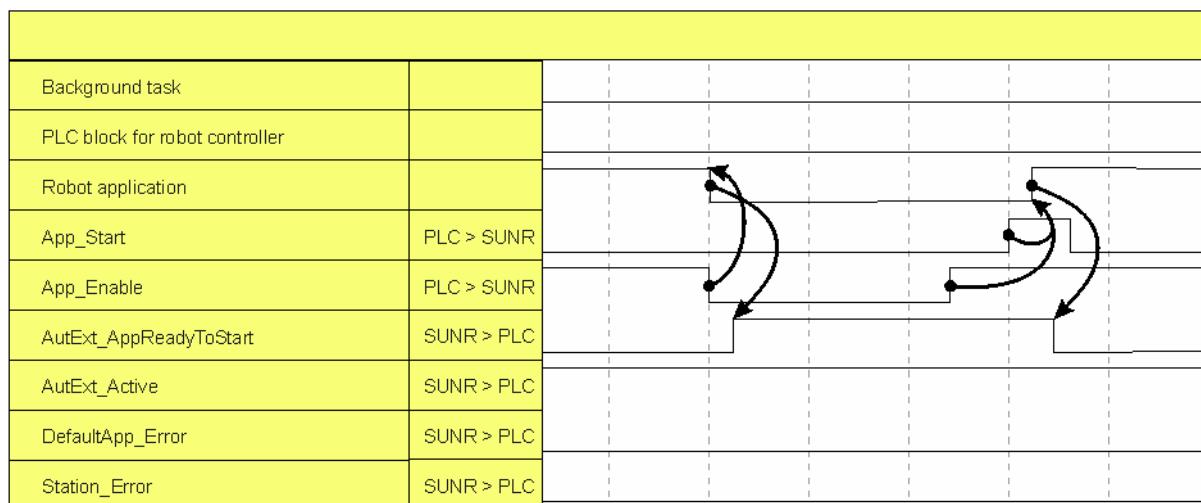


Fig. 12-2: Restart after user stop

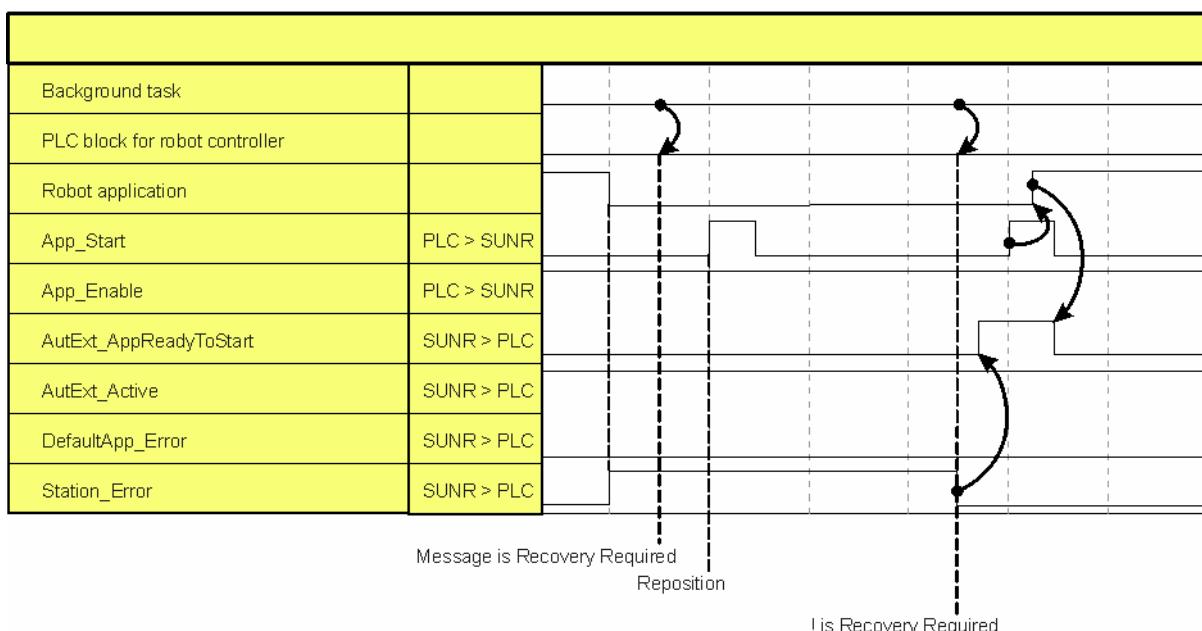


Fig. 12-3: Restart after external EMERGENCY STOP

12.7 Configuring the external controller in the project settings

Procedure

1. Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Change project settings** from the context menu. The **Properties for [Sunrise Project]** window opens.
2. Select **Sunrise > External control** in the directory in the left area of the window.
3. Make the settings for external control of the project in the right-hand area of the window.
 - Set the check mark at **Project is controlled externally**.
 - In the **Default application** area, select the default application.
 - Under **Input interface:**, select the interface for the external communication.
 - Configure the input/output parameters for the interface.
 (>>> [12.7.1 "Input/output parameters of the I/O interface" Page 227](#))
 (>>> [12.7.2 "Input/output parameters of the UDP interface" Page 227](#))
4. Click on **OK** to save the settings and close the window.



The selected default application is indicated by the following icon in the **Package Explorer** view:

If the default application is renamed, the icon is no longer displayed and the application must be selected as the default application once again.
 (>>> [5.4.6 "Setting the robot application as the default application" Page 67](#))

Description

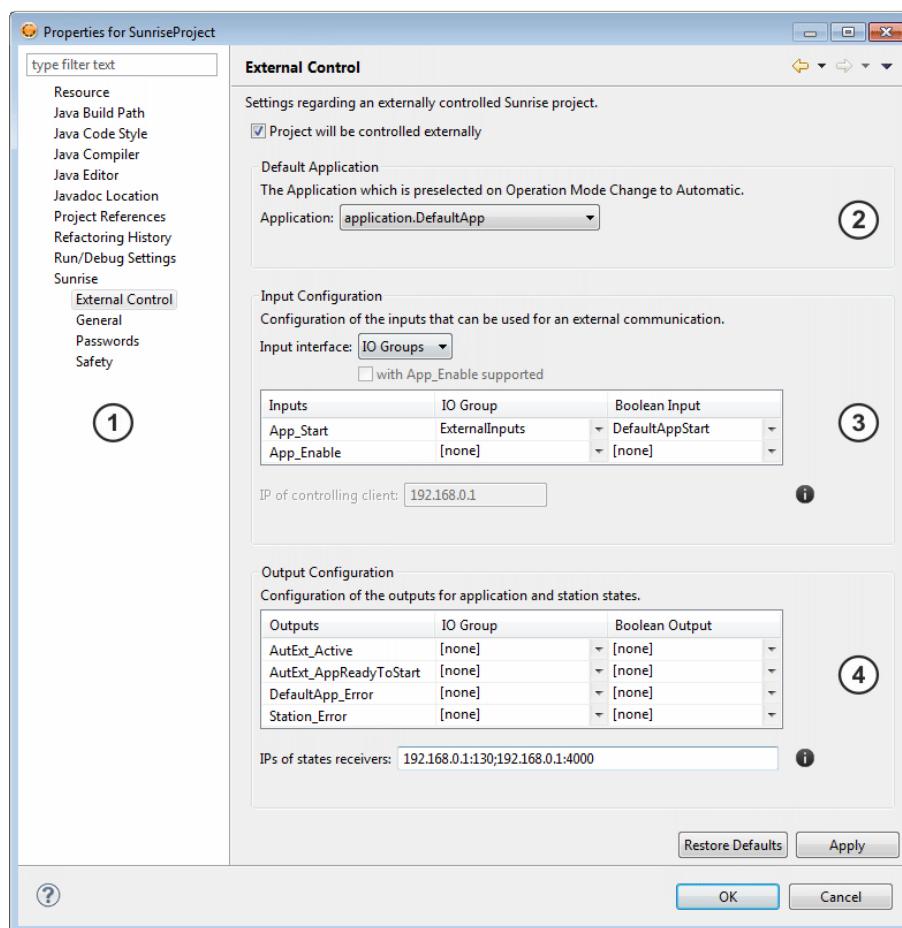


Fig. 12-4: External control

Item	Description
1	Directory of the project settings
2	“Default application” area All robot applications of the project are available for selection as the default application.
3	“Input configuration” area The interface for the external communication is selected here: <ul style="list-style-type: none"> • IO Groups: I/O interface • UDP: UDP interface The configurable input parameters depend on the specific interface.
4	“Output configuration” area The configurable output parameters are not dependent on the interface selected for the inputs. The values of the outputs can also be requested via UDP, for example, if the I/O interface has been configured for the inputs.

The following buttons are available:

Button	Description
Restoring default values	Resets the window to the default settings. All user settings will be lost.
Apply	Saves the user settings. The window remains open.
OK	Saves the user settings and closes the window.
Cancel	Closes the window without saving.

12.7.1 Input/output parameters of the I/O interface

If the I/O interface is used, mapped inputs/outputs of an I/O group must be assigned to the required input/output signals.

The input **App_Start** is absolutely vital for external control of a project. The input **App_Enable** and the signal outputs can optionally be configured.

Column	Description
I/O group	All I/O groups of the I/O configuration of the project are available.
Boolean input	All inputs of the I/O group selected in the I/O group column are available.
Boolean output	All outputs of the I/O group selected in the I/O group column are available.

12.7.2 Input/output parameters of the UDP interface

Parameter	Description
with App_Enable supported	Use of the input signal App_Enable <ul style="list-style-type: none"> Check box not active (default): App_Enable is not evaluated. Check box active: App_Enable is evaluated.
IP of controlling client:	IP address of the client configured for external control of the project
IPs of state receivers:	List of clients to receive status information (optional) For each client, the IP address must be specified in the following format together with the corresponding port: <ul style="list-style-type: none"> <i>IP_address_1:Port_1;IP_address_2:Port_2;...</i> Note: It is advisable to specify the IP address and port of the controlling client in order to inform it of changes of state.

12.8 Formatting of the UDP data packets

Form and length of the UDP data packets for the data exchange are predefined:

- UTF-8 coding
- Data arrays are separated by a semicolon.

12.8.1 Status messages of the robot controller

Description

In the case of the UDP interface, application and station statuses are transferred from the robot controller to an external controller by means of so-called status messages.

In the following cases, the robot controller sends status messages to the clients that are configured as recipients of status messages in the project settings:

- Following receipt of the control message from an external client
- Following the change in state of an output signal

The data packet sent by the robot controller consists of the following data arrays:

Array no.	Description
1	<p>Time stamp Type: Integer (long); unit: ms</p> <p>The time stamp is the current system time of the robot controller when the status message is sent. Corresponds to the time in milliseconds elapsed since midnight on 1.1.1970.</p>
2	<p>Data packet counter (packets sent to the client)</p> <p>When the robot controller sends a new status message, the counter is incremented by 1. The client can use the counter to determine the order in which the status messages were sent.</p>
3	<p>Data packet counter (valid packets received by the client)</p> <p>When the robot controller signals to the client that the received packet is valid, the counter is incremented by 1. The client can use the counter to determine the controller message to which the robot controller is responding.</p> <p>The client can request the counter for restoration of a cancelled connection and then use the requested <i>value+1</i> as the counter in its next controller message.</p>
4	<p>Error ID</p> <p>The ID signals to the client whether the received controller message was valid or defective.</p> <p>(>>> "Error codes" Page 229)</p>
5	<p>Current status of the output signal AutExt_Active</p> <ul style="list-style-type: none"> • TRUE: AUT mode is active and the project on the robot controller can be controlled externally via UDP. • FALSE: AUT mode is not active or the project on the robot controller cannot be controlled externally via UDP.
6	<p>Current status of the output signal AutExt_AppReadyToStart</p> <ul style="list-style-type: none"> • TRUE: The default application is ready to start. • FALSE: The default application cannot be started.
7	<p>Current status of the output signal DefaultApp_Error</p> <ul style="list-style-type: none"> • TRUE: An error occurred during execution of the default application. • FALSE: The default application has not signaled an error.

Array no.	Description
8	<p>Current status of the output signal Station_Error</p> <ul style="list-style-type: none"> • TRUE: The station has signaled an error. • FALSE: The station is running without errors.
9	<p>Current state of the default application</p> <ul style="list-style-type: none"> • IDLE: The application is selected. • RUNNING: The application is executed. • MOTIONPAUSED: The application is paused. • REPOSITIONING: The robot is repositioned. The application is still paused because the robot has left the path. • ERROR: An error occurred while the application was running. • STARTING: The application is being initialized to switch to the RUNNING state. • STOPPING: The application is being reset to the start of the program. The application is then in the IDLE state.
10	<p>Current status of the input signal App_Start</p> <ul style="list-style-type: none"> • TRUE, FALSE <p>Status defined by the last valid controller message. The client can request the current status of the signal for restoration of a cancelled connection.</p>
11	<p>Current status of the input signal App_Enable</p> <ul style="list-style-type: none"> • TRUE, FALSE <p>Status defined by the last valid controller message; FALSE if no controller message has been received in the last 100 ms. The client can request the current status of the signal for restoration of a cancelled connection.</p>

Example

```
1449066055468;7;2;1;true;false;false;false;RUNNING;false;false
```

Error codes

ID	Description
1	<p>No error – internally triggered change of state</p> <p>The change of state of an output signal was not triggered by a message from the controlling client, but by an internal event on the robot controller.</p>
0	<p>No error – valid message received</p> <p>The most recently received message is valid and is being processed.</p>
-1	<p>Incorrect client IP address</p> <p>The IP address of the client that sent the message does not match the IP address of the client configured for external control.</p>

ID	Description
-2	Incorrect message structure The message could not be decoded, e.g. because it contains too many or too few data arrays or because the data arrays are not separated by semicolons.
-3	Incorrect data packet counter The data packet counter of the current message was not incremented by 1 (relative to the most recently received message).
-4	Incorrect time stamp The time stamp must be an integer.
-5	Incorrect signal name The signal name must be App_Start , App_Enable or Get_State .
-6	Incorrect signal value The signal value must be TRUE or FALSE.
-7	Timeout error After App_Enable was set to TRUE, no further valid message was received for 100 ms. The application is paused.



If more than one fault occurs simultaneously, the fault with the highest priority is transferred. A fault with the ID -3, for example, has a higher priority than a fault with the ID -4.

12.8.2 Controller messages of the external client

Precondition

When a controller message is sent, the following target address and port must always be specified:

- IP address of the robot controller (see **Configuration** tab in the station configuration)
- Port 30300 (fixed port of the robot controller)

Description

With the UDP interface, input signals are set via so-called controller messages that the external controller must send to the robot controller. This client data packet must contain the following data arrays:

Array no.	Description
1	Time stamp Type: Integer (long); unit: ms The time stamp should be the current system time of the client when the controller message is sent.
2	Data packet counter (packets sent to the robot controller) When the client sends a new controller message, the counter must be incremented by 1.

Array no.	Description
4	<p>Name of the input signal</p> <ul style="list-style-type: none"> App_Start The default application can be started or resumed by means of a change of state from FALSE to TRUE as long as the output signals AutExt_Active and AutExt_AppReadyToStart are TRUE. App_Enable If App_Enable is activated in the project settings, the signal must be TRUE in order to be able to start or resume the default application. (>>> "App_Enable" Page 231) Get_State With this signal (TRUE or FALSE), the client can request a status message from the robot controller.
5	<p>Value of the sent input signal</p> <ul style="list-style-type: none"> TRUE, FALSE

Example

```
1449066055468;1;App_Start;true
```

App_Enable

If the input signal **App_Enable** is evaluated, the following points must be taken into consideration when sending controller messages:

- The application can only be started by the input signal **App_Start** if the robot controller has received a message with ...App_Enable;true in the last 100 ms.
- The input signal **App_Enable** functions like a heartbeat signal. The application is executed as long as the robot controller receives a controller message, e.g. ...App_Enable;true, at least every 100 ms. If no message is received, the application is paused.



When sending the controller messages, the client must take the network delay into account.

- If the external client sets the input signal **App_Enable** from TRUE to FALSE within the 100 ms, this also pauses the application.

12.9 External control via UDP – Start-up example

The example shows how a robot application can be started from a PC via UDP and what start-up steps and programming are required for this.

The input signal **App_Enable** is not used in this example. This example can thus not be used to pause an application and does not claim to be comprehensive.

12.9.1 Starting up the external controller

The following steps are required for starting up the external controller:

1. Connect the PC to the robot controller via the Ethernet interface KLI.
2. Assign a fixed network IP address to the PC, e.g. 192.168.0.10.



It is recommended not to use dynamic IP address assignment via DCHP.

3. Carry out the required project settings in Sunrise.Workbench.
 - Select the application to be started as the default application.
 - Select the UDP interface.
 - Enter the network IP address of the PC as the IP of the controlling client.
 - Enter the IP address and port via which the PC receives the status messages from the robot controller (here: port 30333).

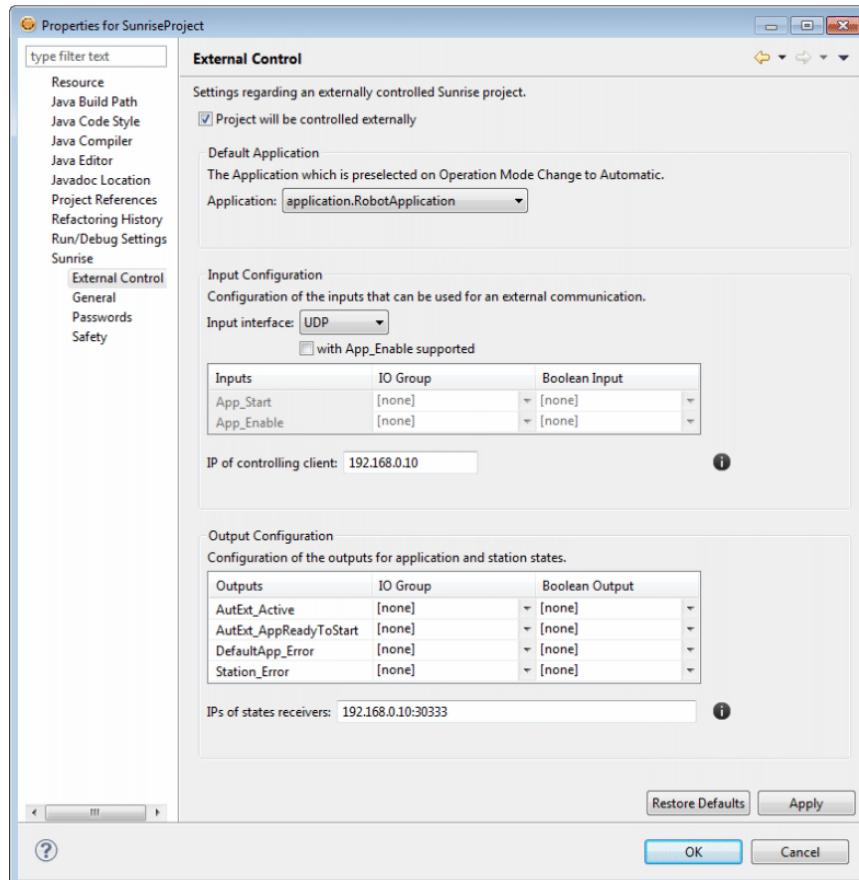


Fig. 12-5: Project settings in Sunrise.Workbench

4. Synchronize the project to the robot controller.
5. Select AUT mode.

Once the robot is ready to move, all status indicators on the smartHMI are green and the application can be started via the UDP interface.

12.9.2 Programming the external controller

On the PC used for external control, there must be a program that can generate and send UDP data packets.



If a firewall is used, it must be ensured that it does not block the incoming and outgoing UDP data packets.

Precondition

- The correct target address and port have been assigned to the data packets that are to be sent:
 - IP address of the robot controller (see **Configuration** tab in the station configuration)
 - Port 30300 (fixed port of the robot controller)

Description

Following a reboot of the robot controller, the robot application can be started with a controller message with **App_Start**:

```
1457449078435;1;App_Start;true
```



The first number in the packet is the time stamp that must be used to document when the packet was sent. Here, and in the following code examples, this number must always be replaced with a current time stamp in milliseconds. (When using Java, such a number can be generated, for example, with `java.lang.System.currentTimeMillis()`.)



Following a reboot of the robot controller, the value 1 must always be transferred for the data packet counter. For each subsequent data packet, the counter must be incremented by 1.

If the value to be transferred for the counter is not known, a socket on the PC must be opened that can receive UDP messages at port 30333.

Get_State can then be used to request the current counter value:

```
1457450539457;1;Get_State;true
```



In Sunrise.Workbench, port 30333 has been defined in the project settings as the port via which the PC receives the status messages from the robot controller. If a different port is to be used, it can be entered in the project settings.

If the socket is now checked for received messages, a status message should now be present as the answer from the robot controller, e.g.:

```
1457450539459;4;1337;-3;true;true;false;false;IDLE;false;true
```

The received message shows that the current value of the data packet counter is 1337. The counter value 1338 must therefore be transferred in the next data packet.

In order to restart a robot application, the state of the signal from **App_Start** must change from FALSE to TRUE. For this purpose, the following packets are sent:

```
1457450539511;1338;App_Start;false
```

```
1457450539511;1339;App_Start;true
```



It is advisable to check the socket for received messages after every data packet that is sent to the robot controller. In this way, it is easy to check whether an error occurred during processing of a message.

(>>> ["Error codes" Page 229](#))

Java program

```
1 import java.net.*;
2 class UdpSample
3 {
4     public static void main(String args[]) throws Exception
5     {
6         DatagramSocket mySocket = new DatagramSocket(30333);
7
8         // robot address (please adjust IP!)
9         InetSocketAddress robotAddress =
10            new InetSocketAddress("192.168.0.2", 30300);
11
12        // get robot state
13        byte[] msg = String.format("%d;1;Get_State;true",
14            System.currentTimeMillis()).getBytes("UTF-8");
15        mySocket.send(new DatagramPacket(msg, msg.length, robotAddress));
16
17        // receive answer state message
18        byte[] receiveData = new byte[508];
19        DatagramPacket receivePacket = new DatagramPacket(receiveData,
20            receiveData.length);
21        mySocket.receive(receivePacket);
22
23        // extract counter
24        String[] stateMessage = new String(receivePacket.getData()).split(";");
25        long counter = Long.parseLong(stateMessage[2]);
26
27        // start application by sending a rising edge
28        // (false->true) for App_Start
29        msg = String.format("%d;%d;App_Start;false", System.currentTimeMillis(),
30            ++counter).getBytes("UTF-8");
31        mySocket.send(new DatagramPacket(msg, msg.length, robotAddress));
32        msg = String.format("%d;%d;App_Start;true", System.currentTimeMillis(),
33            ++counter).getBytes("UTF-8");
34        mySocket.send(new DatagramPacket(msg, msg.length, robotAddress));
35    }
36 }
```

12.10 Configuring the signal outputs for a project that is not externally controlled

Description

The predefined output signals for the external controller can also be used to signal application and station statuses in projects that are not externally controlled.

The application statuses always refer to the default application selected in the project settings.



The selected default application is indicated by the following icon in the **Package Explorer** view:



If the default application is renamed, the icon is no longer displayed and the application must be selected as the default application once again.

(>>> [5.4.6 "Setting the robot application as the default application"](#)
[Page 67](#))

Precondition

- In the case of communication via the I/O system of the robot controller: The I/O configuration of the project contains the outputs configured and mapped in WorkVisual.
(>>> [12.5 "External controller output signals" Page 223](#))

Procedure

- Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Change project settings** from the context menu.
The **Properties for [Sunrise Project]** window opens.
- Select **Sunrise > General** in the directory in the left area of the window.
- Make the general settings for the project in the right-hand area of the window.
 - If application statuses are to be signaled: In the **Default application** area, select the desired default application.
 - In the **Output configuration** area, configure the output parameters required by the communications interface.
(>>> [12.10.1 "Output parameters of the I/O interface" Page 235](#))
(>>> [12.10.2 "Output parameters of the UDP interface" Page 235](#))
- Click on **OK** to save the settings and close the window.

12.10.1 Output parameters of the I/O interface

If the I/O interface is used, mapped outputs of an I/O group must be assigned to the required output signals.

Column	Description
I/O group	All I/O groups of the I/O configuration of the project are available.
Boolean output	All outputs of the I/O group selected in the I/O group column are available.

12.10.2 Output parameters of the UDP interface

Parameter	Description
IPs of state receivers:	List of clients to receive status information For each client, the IP address must be specified in the following format together with the corresponding port: <ul style="list-style-type: none"> • <i>IP_address_1:Port_1;IP_address_2:Port_2;...</i>

13 Safety configuration

13.1 Overview of safety configuration

The safety configuration defines the safety-oriented functions in order to integrate the robot safely into the system. Safety-oriented functions serve to protect human operators when they work with the robot.

The safety configuration is an integral feature of a Sunrise project and is managed in tabular form. The individual safety functions are grouped in KUKA Sunrise.Workbench on an application-specific basis. The safety configuration is then transferred with the project to the controller and activated there.



WARNING

Serious damage and injury or death can result from incorrect safety configuration. If a new or changed safety configuration is activated, the safety maintenance technician must conduct tests to ensure that the configured safety parameters have been correctly applied and that the safety functions of the configuration are fully functional (safety acceptance).



Configuration of the safety functions, activation and deactivation of the safety functions and safety acceptance may only be carried out by a trained safety maintenance technician. The safety maintenance technician is responsible for ensuring that the safety configuration is only activated on those robots for which it is intended.

The safety configuration is not checked for plausibility by KUKA Sunrise.Workbench.



Take substitute measures for minimizing risk in the case of incomplete start-up

Start-up is incomplete, for example, if not all safety functions have yet been implemented, or if a function test of the safety functions has not yet been carried out. The absence of substitute measures for minimizing risk can result in death, serious injury or damage to property.

- Additional substitute measures for minimizing risk must be taken and documented, e.g. installing a safety fence, attaching a warning sign, locking the main switch.



The system integrator must verify that the safety configuration sufficiently reduces risks during collaborative operation (HRC).

It is advisable to perform this verification taking into consideration the information and instructions for operating collaborative robots in ISO/TS 15066.



States with various safety settings are defined in the safety configuration as part of the ESM mechanism (Event-Driven Safety Monitoring). It is possible to switch between these in the application. Since switching between these states is carried out by means of non-safety-oriented signals, all configured states must be consistent. This means that each state must ensure a sufficient degree of safety, regardless of the time or place of activation (i.e. regardless of the current process step).

13.2 Software classification

The development processes of the software meet the requirements on software according to IEC 62304. The classification of the software is derived from the LBR Med risk management process.

The software comprises 2 components:

- Safety controller:
Class C software
- Motion controller (non-safe part of Sunrise.OS):
Class B software

The motion controller is considered as SOUP (Software of Unknown Provenance).

The manufacturer's intended application of the medical device is not known. For this reason, the worst case is assumed for classification of the software:

- An error in the motion controller can lead to death or serious injuries; the behavior of the motion controller cannot be guaranteed in the event of an error. This means that the motion controller is to be assigned to class C.
- An error in the safety controller can lead to death or serious injuries. This means that the safety controller is to be assigned to class C. The safety controller was developed in accordance with processes that meet the requirements of IEC 62304. The safety controller thus meets the requirements of class C and the behavior can be guaranteed in the event of an error.
- The safety controller can safely monitor errors arising from the motion controller. A precondition for this is a correct safety controller configured by the safety maintenance technician of the medical device manufacturer in accordance with the application-specific risk management.

The reaction in the event of an error can be safe stopping of the robot or the setting of a safe output to which the higher-level safety controller of the medical device manufacturer responds.

The safety controller is the measure for reducing the risk so that the software class of the motion controller can be reduced to class B.

In the software, the software classes are assigned to the individual components as follows:

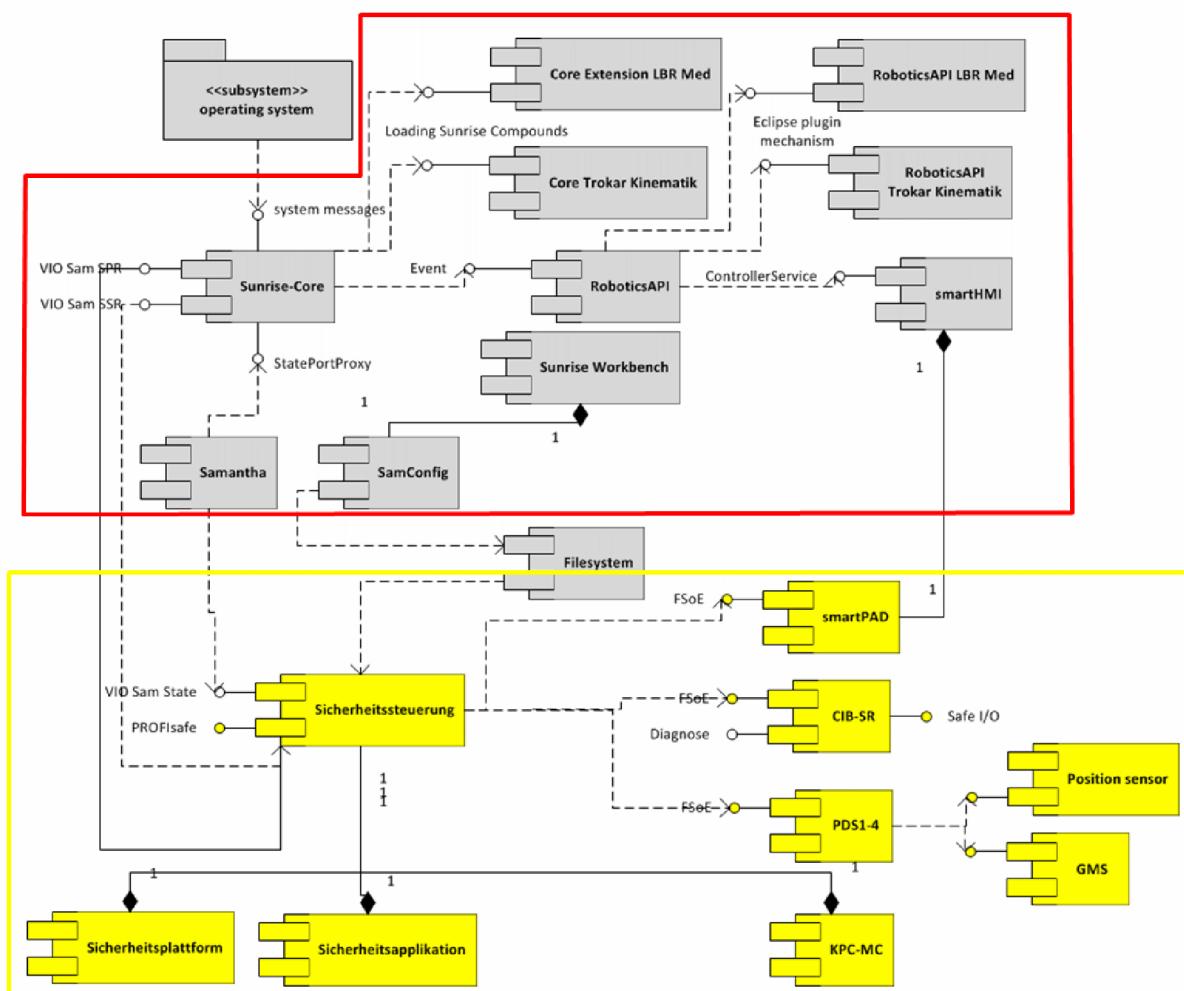


Fig. 13-1: Software classification

1 Red: motion controller, class B

2 Yellow: safety controller, class C

KUKA Deutschland GmbH cannot guarantee that all user-specific applications implemented reduce the risk sufficiently. The risk arising from the implementation of user-specific applications must be assessed by the medical device manufacturer in the context of his risk management process.

(>>> 13.2.2 "Effects of software classification" Page 241)

13.2.1 Requirements on SOUP components

In accordance with IEC 62304, there are particular requirements on the SOUP components of the software. (>>> 13.2 "Software classification" Page 238)

13.2.1.1 Development process

The software was created in accordance with the KUKA development process. The development is based on the version Sunrise.OS 1.15 which, with the exception of the safety controller, is considered as a SOUP component for Sunrise.OS Med and is referred to hereinafter as the motion controller. Only one configuration of Sunrise.OS 1.15 is used in

the product. Modifications to the configuration or an update for Sunrise.OS Med are not planned.

The functional capability of all software components is ensured by means of the component test. The integration of all components into the overall system is checked by means of a release test based on a test plan.

Software risk management is implemented at system level by means of a dedicated risk management process. At component level, component FMEAs are performed. Potential risks arising from the motion controller are also taken into consideration. A final assessment of the risks in conjunction with the user-specific applications must be carried out by the medical device manufacturer.

13.2.1.2 Software requirements

Sunrise.OS Med, including the motion controller SOUP component, is operated exclusively with the robot and the corresponding robot controller. The requirements on the motion controller are verified by means of integration and system tests. The interfaces to the safety controller are tested by means of unit and component tests.

13.2.1.3 Retrofitting SOUP components

For Sunrise.OS Med, retrofits in the form of updates are limited to the elimination of critical bugs. Known errors will be tracked and, depending on the severity, be communicated to the medical product manufacturer. Modifications going beyond this, particularly the integration of a new Sunrise.OS version, require a new CB Report and a new release. Changes to functions and a list of unresolved risks from the risk management process will be made available to the medical product manufacturer.

13.2.1.4 SOUP components

Sunrise.OS Med contains the following SOUP components:

The Sunrise.OS Med System Software consists of the motion controller (SOUP) and the safety controller. The safety controller additionally uses the following SOUP components:

- SafeOS:
SafeOS is the safe variant of ProConOS.
PLC runtime environment: SafeOS executes safe IEC 61131-compliant PLC code. The PLC code is created in the safe development environment SAFEPROG (safe variant of MULTIPROG). Function blocks that have been implemented in software class C can be integrated into the IEC 61131 code. Both SafeOS and SAFEPROG are purchased parts from KW-Software and were developed in accordance with IEC 61508.
- CIP Safety (Common Industrial Protocol):
Protocol stack for communication with safe EtherNet/IP devices of the overall system (e.g. Rockwell) from Molex
- PROFIsafe:
PROFIsafe protocol stack from KW-Software for communication with the safe PROFINET field bus (e.g. SIEMENS PLC) of the overall system
- VxWorks:
Real-time operating system from WindRiver

13.2.2 Effects of software classification

The figure (>>> [Fig. 13-1](#)) shows which software class acc. to IEC 62304 the corresponding component meets.

Since KUKA is not familiar with the user-specific applications, the system integrator must carry out an assessment in his risk management process and take appropriate measures to ensure that the software of the overall system adequately reduces the risk.

The system integrator must carry out an assessment in his risk management process and take appropriate measures that take into consideration the fact that the behavior of the motion controller (class B) cannot be guaranteed in the event of a fault. Only the behavior of the safety controller (class C) can be guaranteed.

If the risk management process of the system integrator indicates that the behavior of the motion controller (class B) could lead to injuries, serious injuries or death, the system integrator must configure safe, application-specific monitoring. The monitoring must be implemented on the safety controller (class C).

If no direct reduction of the risk can be achieved using the safety controller, additional external measures must be implemented on the system side. The external measures must communicate with the safety controller in order to achieve single-failure safety of the overall system.



Further information about configuring the safety functions is contained in the "Safety configuration" chapter of the operating and programming instructions of the system software for system integrators. (>>> [13 "Safety configuration" Page 237](#))



The basis for an overall system's safety configuration is always a risk management process. The examples shown below do not claim to be comprehensive.

13.2.2.1 Example: Tool changing

In an application, the robot is to move to a specific position for a tool change. The motion command is carried out using the motion controller component. The motion controller (class B) does not guarantee that the path will be executed as planned in the event of a fault. The safety controller does not offer monitoring of the path, but does, for example, provide workspace monitoring. When the workspace limits are reached, the monitoring triggers a preconfigured safe reaction (e.g. stops the robot, switches a safe output). If the hazard is not sufficiently reduced, or if the risk management indicates that the entire path is to be safely monitored, this monitoring must be carried out by an external safety controller (e.g. using a tracking camera). The connected tracking camera communicates with the safety controller via a safe input. The safety controller then triggers the preconfigured safe reaction.

13.2.2.2 Example: Holding the position

In an application, the robot is to hold a specific position in order to position a drilling jig for a drilling operation. The robot position is held using the motion controller component. The motion controller cannot guarantee the holding of the robot position.

The safety controller provides the safe monitoring AMF *Standstill monitoring of all axes*. This monitoring function checks that all axes of the kinematic system are moving at a velocity no greater than $\pm 1^\circ/\text{s}$ and that all axes deviate by less than $\pm 0.1^\circ$ from the position when the AMF is activated.

ted. In the event of a fault, a previously configured safe reaction is triggered, e.g. stopping the robot and applying the brakes or switching a safe output.

13.3 Single fault safety

In the event of a fault, the single fault safety required by IEC 60601 can only be guaranteed for the behavior of the safety controller.

(>>> [13.2 "Software classification" Page 238](#))

A preconfigured reaction is safely triggered, e.g. a Stop 0 or the switching of a safe output. The brakes of the LBR Med are considered as a safety measure and are only applied during operation in the event of a fault. In all other situations, the robot remains under servo control, even if it is not moving or the application has been terminated.

The safety controller monitors the system for single faults relating to the safety-oriented monitoring functions of the system.

(>>> [13.9 "Atomic Monitoring Functions" Page 248](#))

For the robot, this means that unexpected behavior of the motion controller is to be considered as a single fault. The monitoring by the safety controller constitutes the risk-reducing measure. The safety maintenance technician must implement safe monitoring of the expected behavior by means of a suitable safety configuration, dependent on the specific application and risk situation.

If the safety controller does not provide a suitable monitoring function for the expected behavior, the safety maintenance technician must implement a monitoring function that can trigger a reaction of the safety controller via a safety-oriented external signal. (>>> [13.2.2.1 "Example: Tool changing" Page 241](#))

If the safety controller detects a single fault, a preconfigured safe reaction is triggered.

The monitoring function covers the following single faults:

1. Internal fault in the safety controller (e.g. cross comparison error):
The overall system is switched to the safe state. The robot is stopped and the safety-oriented outputs assume the LOW state.
2. If the AMFs of a row of the PSM table are violated and there is no internal fault of the safety application active, either an external signal is set or a stop is commanded, depending on the safety configuration.
(>>> [13.9 "Atomic Monitoring Functions" Page 248](#))
3. An external safety controller signals a fault via a safe input of the safety controller.

As the reaction to violated AMFs of a PSM row, the safety controller must be configured as follows:

Configuration of PSM rows on the basis of the safety-oriented monitoring functions listed in (>>> [13.9 "Atomic Monitoring Functions" Page 248](#)) and setting of a safe output to which a higher-level controller can respond and/or a configuration of a safe stop of the robot for which application of the brakes is commanded.

A fundamental performance feature is that a defined state (safe state) is assumed in the event of a fault in order to rule out the possibility of unforeseen robot motions in the event of a fault.



The safety controller can only detect single faults on the basis of monitoring functions (2) and (3) if the safety configuration has been correctly configured by the safety maintenance technician in accordance with the requirements of the application. The safety maintenance technician must critically consider this in his risk management process. If an internal fault occurs (1), the robot is switched to the safe state. The safety maintenance technician must ensure that no unacceptable risks can arise from an internal fault in the application.

13.4 Safety concept

Overview

The safety configuration must implement all safety functions which are required to operate the robot. A safety function monitors the entire system on the basis of specific criteria. These are described by individual monitoring functions, so-called AMFs (Atomic Monitoring Functions). To configure a safety function, several AMFs can be linked to form complex safety monitoring functions. In addition, the safety function defines a suitable reaction which is triggered in case of error.

Example: In a specific area of the robot's workspace, the velocity at the TCP must not exceed 500 mm/s ("Workspace monitoring" and "Velocity monitoring" monitoring functions). Otherwise, the robot must stop immediately (reaction in case of error).

PSM and ESM

The Sunrise safety concept provides 2 different monitoring mechanisms:

- Permanent safety-oriented monitoring

The safety functions of the PSM mechanism (Permanent Safety Monitoring) are always active. It is only possible to deactivate individual safety functions by changing the safety configuration.

The PSM mechanism is used to constantly monitor the system. It implements basic safety settings which are independent of the process step being carried out. These include, for example, EMERGENCY STOP functions, the enabling switch on the smartPAD, the definition of a cell area or safety functions that depend on the operating mode.

- Event-dependent safety-oriented monitoring

The ESM mechanism (Event-driven Safety Monitoring) defines safe states. It is possible to switch between these in the application. A safe ESM state contains the safety functions required in the corresponding process step.

Since switching is carried out by means of non-safety-oriented signals, the defined state must ensure a sufficient degree of safety, regardless of the time or place of activation.

The ESM mechanism allows specific safety functions to be adapted for specific processes. This is of particular importance for human-robot collaboration applications, as these often require various safety settings depending on the situation. The required parameters, such as permissible velocity, collision values or spatial limits, can be individually defined for each process step using an ESM state.

AMF

The smallest unit of a safety monitoring function is called an Atomic Monitoring Function (AMF).

Each AMF supplies an elementary, safety-relevant item of information, e.g. whether a safety-oriented input is set or whether the Automatic operating mode is selected.

Atomic Monitoring Functions can have 2 different states and are LOW-active. This means that if a monitoring function is violated, the state switches from "1" to "0".

- State "0": The AMF is violated.
- State "1": The AMF is not violated.

The AMF *smartPAD Emergency Stop* is violated, for example, if the EMERGENCY STOP device on the operator panel is pressed.

Safety function

A safe ESM state is defined with up to 20 safety functions. The safety functions of the ESM mechanism use exactly one AMF. If this AMF is violated, the safety function and thus the entire ESM state is considered to be violated.

For safety functions of the PSM mechanism, up to 3 AMFs are logically linked to one another. This allows complex safety monitoring functions to be implemented. If all AMFs of a safety function of the PSM mechanism are violated, the entire safety function is considered to be violated.

13.5 Safety-oriented reactions

A suitable reaction is defined for each safety function. This reaction must take place in the case of an error and put the system into a safe state.

The following reactions can be configured:

- Safety stop 0 is triggered.



It is advisable to only configure a safety stop 0 if it is necessary to immediately switch off the drives and apply the brakes as a reaction.

- Safety stop 1 is triggered.
- Safety stop 1 (path-maintaining) is triggered.

This is the recommended stop reaction. It has the lowest impact on the process, as an application can be resumed without the need to re-position the robot.



CAUTION

In crushing situations, safety stop 1 and safety stop 1 (path-maintaining) can result in higher crushing forces due to the controlled stop on a planned braking path. It is therefore advisable to use safety stop 0 for safety monitoring functions which recognize crushing situations (e.g. the AMF *Collision detection*, *TCP force monitoring*).

- Brake is triggered.

The robot is braked on the path as long as the safety monitoring is violated. The braking process is monitored by the safety controller. Safety stop 1 is executed in the event of a fault.

Unlike the safety stop, the braking process does not lead to a complete standstill or deactivation of the drives. The velocity reduction and safety-oriented monitoring of the braking process are only carried out until the safety monitoring is no longer violated.

Conditions for use of the "Brake" safety reaction:

- Only available with the KUKA Sunrise.EnhancedVelocityController option

- Only available for safety functions of the PSM mechanism
 - Only compatible with safety functions that contain Cartesian velocity monitoring
 - Not compatible with safety functions that contain an extended AMF
 - Only compatible with position-controlled spline motions
- (>>> [17.2 "Brake" safety reaction" Page 550](#))
- Safety-oriented output is set to "0" (LOW level).
- Setting a safety-oriented output is a safety reaction that is only available for safety functions of the PSM mechanism.

The reactions can be used for any number of safety functions. A reaction is triggered once one of the safety functions using this reaction is violated. This makes it possible, for example, to inform a higher-level controller via a safe output when specific errors occur.

With the PSM mechanism, it is possible to trigger several different reactions when a specific combination of AMFs is violated. For example, a safety stop can be triggered as well as a safe output. To do so, 2 safety functions must be configured with identical AMF combinations.

If 2 safety functions differ only in the type of stop configured, a violation triggers the stronger stop reaction. In other words, it triggers the stop reaction which causes an earlier safety-oriented disconnection of the drives. If several safety functions use the same output signal as a reaction, this signal is set to "0" once one of the safety functions is violated.

13.5.1 Time response of safety-oriented outputs

All the safety-oriented outputs use LOW as a safe state.

If a safety function which uses a safety output as a reaction is violated, this output is immediately set to LOW.

If the violation state is cancelled, the output is only set to HIGH again when the following conditions have been met:

- The safety function is not violated for at least 24 ms. The reaction to cancellation of the violation state is always delayed.
- If an Ethernet safety interface is used:
The output has the LOW level for at least 500 ms beforehand. If the LOW level has not yet been present for this time, the level change to HIGH waits until the 500 ms has elapsed.
- If the discrete safety interface is used:
The output has the LOW level for at least 200 ms beforehand. If the LOW level has not yet been present for this time, the level change to HIGH waits until the 200 ms has elapsed.



When using safety functions with a safe output as a reaction, it must be noted that connection errors (i.e. communication errors) at safe inputs or outputs are automatically acknowledged by the safety controller when the connection is restored. Accordingly, the level of the safe output can switch from LOW to HIGH once the connection is restored.
For this reason, the safety maintenance technician must ensure that peripheral devices do not automatically restart.

13.6 Safety interfaces

Various safety interfaces are available for exchanging safety-oriented signals between a higher-level controller and a robot controller. The safety-oriented inputs of these interfaces can be used to connect safety devices,

e.g. external EMERGENCY STOP devices or safety gates, and to evaluate the corresponding input signals. The safety-oriented outputs of these interfaces can be used to signal the violation of safety functions.

- Ethernet safety interfaces (only slave function available)
 - PROFINET/PROFIsafe
 - EtherCAT/FSoE
- Discrete safety interfaces
 - CIB_SR/X11



Field buses such as PROFINET and EtherCAT can be configured in WorkVisual. Further information about the concrete configuration of the field buses is contained in the corresponding field bus documentation.



Further information on interface X11 can be found in the Instructions for Use of the KUKA Sunrise Cabinet Med robot controller.

13.7 Permanent Safety Monitoring

Description

The safety functions of the PSM mechanism (Permanent Safety Monitoring) are permanently active and use the criteria defined by these functions to ensure that the overall system is constantly monitored.

For a safety function of the PSM mechanism, up to 3 AMFs (Atomic Monitoring Functions) can be linked to one another. The entire safety function is only considered violated if all of these AMFs are violated. The safety function also defines a reaction. This is triggered if the entire safety function is violated.

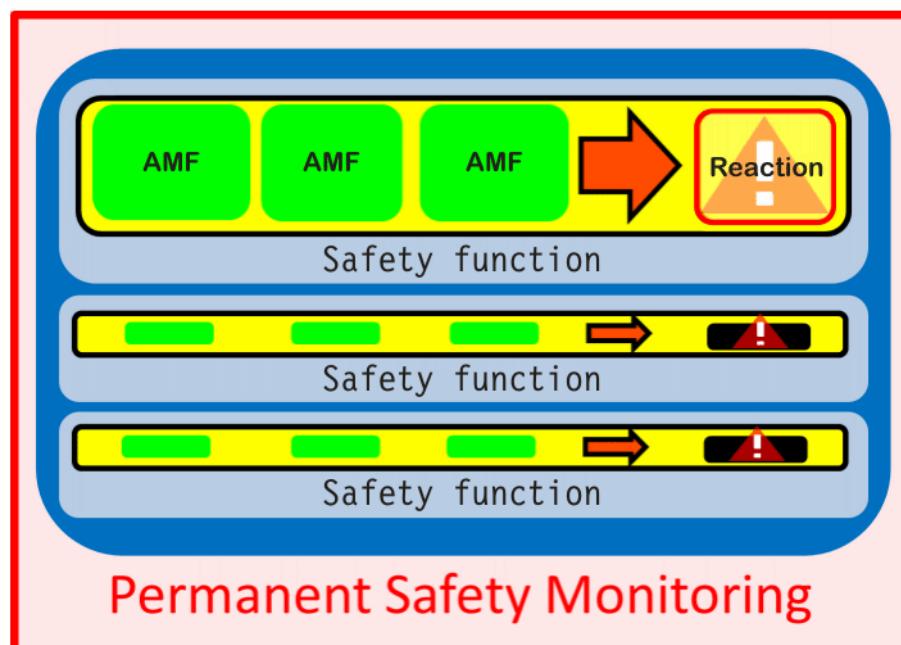


Fig. 13-2: Safety functions of the PSM mechanism

Categories

For diagnosis in case of error, a category is assigned to each safety function of the PSM mechanism. Depending on the category, errors are displayed on the smartPAD and saved in the LOG file. For this reason, it is advisable to select these carefully.

The following categories are available:

Category	Recommended use
<i>None</i>	For safety functions which cannot be assigned a category
<i>Output</i>	For safety functions which use setting an output as a reaction In this category, no diagnostic information is provided in case of violation.
<i>Enabling device</i>	For safety functions which evaluate an enabling switch In this category, no diagnostic information is provided in case of violation because enabling is a normal operating state and not an error state.
<i>Local EMERGENCY STOP</i>	For safety functions which evaluate an EMERGENCY STOP triggered by the EMERGENCY STOP device on the smartPAD
<i>External EMERGENCY STOP</i>	For safety functions which evaluate an EMERGENCY STOP triggered by an external EMERGENCY STOP device
<i>Operator safety</i>	For safety functions which evaluate the signal for operator safety
<i>Safe operational stop</i>	For safety functions which monitor robot standstill
<i>Collision detection</i>	For safety functions which are used for collision detection or force monitoring
<i>Safety stop</i>	For safety functions which use a safety stop as a reaction and cannot be assigned to another category. Example: external safety stop
<i>Velocity monitoring</i>	For safety functions which are used for monitoring an axis-specific or Cartesian velocity
<i>Workspace monitoring</i>	For safety functions which are used for monitoring an axis-specific or Cartesian space

13.8 Event-driven Safety Monitoring

The ESM mechanism (Event-driven Safety Monitoring) makes it possible to switch between different safe ESM states depending on the situation.

Up to 10 safe states can be defined. Switching between states can be carried out in the robot application or in a background application.

(>>> [13.12.4.8 "Switching between ESM states" Page 270](#))

A safe ESM state is defined with up to 20 safety functions which must ensure a sufficient degree of safety in every situation. An ESM state becomes active when the program switches to this ESM state. As long as the ESM state is active, all corresponding safety functions are monitored in addition to the permanently active safety functions.

Use of the ESM mechanism is optional. The ESM mechanism is deactivated if no ESM state is defined in the safety configuration.

When using the ESM mechanism, exactly one safe state is always active. It is not possible to switch it off in the application.

The safety functions of an ESM state each contain a single AMF which is assigned to a suitable stop reaction.

Once a safety function of the active ESM state is violated, a stop is triggered. The type of stop reaction will be the strongest of all the violated safety functions in all ESM states (either active or inactive). In other words, it triggers the stop reaction which causes the earliest safety-oriented disconnection of the drives.

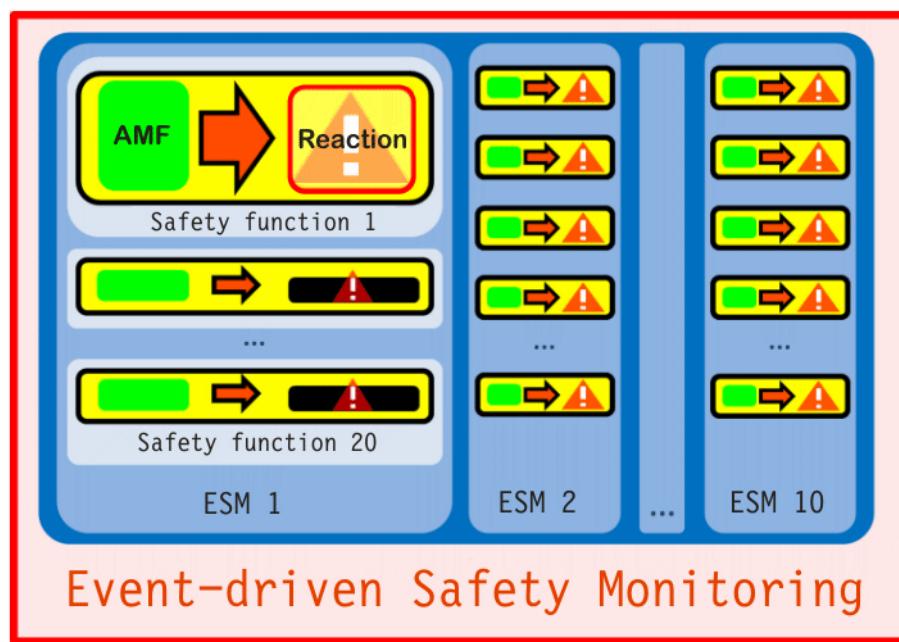


Fig. 13-3: Safety functions of the ESM mechanism

13.9 Atomic Monitoring Functions

The smallest unit of a safety function is designated as the Atomic Monitoring Function (AMF). This can be, for example, evaluating the enabling switch on the smartPAD or monitoring the velocity of an axis.

Categories

Atomic Monitoring Functions are divided into 3 categories:

- Standard AMFs
- Parameterizable AMFs
- Extended AMFs

Overview

KUKA Sunrise contains a basic package of AMFs. These include, for example, all standard AMFs. The following safety options are also available and can be used to install further AMFs:

- KUKA Sunrise.SafeOperation (SOP)
- KUKA Sunrise.HRC: safety option for HRC applications

AMF	Basic	SOP	HRC
<i>Axis range monitoring</i>	✓	✓	✓
<i>Automatic mode</i>	✓	✓	✓
<i>Test mode</i>	✓	✓	✓
<i>High-velocity mode</i>	✓	✓	✓
<i>Reduced-velocity mode</i>	✓	✓	✓
<i>Input signal</i>	✓	✓	✓
<i>Motion enable</i>	✓	✓	✓

AMF	Basic	SOP	HRC
<i>smartPAD Emergency Stop</i>	✓	✓	✓
<i>Position referencing</i>	✓	✓	✓
<i>Time delay</i>	✓	✓	✓
<i>smartPAD enabling switch inactive</i>	✓	✓	✓
<i>smartPAD enabling switch panic active</i>	✓	✓	✓
<i>Axis velocity monitoring</i>	✗	✓	✓
<i>Cartesian workspace monitoring</i>	✗	✓	✓
<i>Cartesian velocity monitoring</i>	✗	✓	✓
<i>Cartesian protected space monitoring</i>	✗	✓	✓
<i>Standstill monitoring of all axes</i>	✗	✓	✓
<i>Tool-related velocity component</i>	✗	✓	✓
<i>Tool orientation</i>	✗	✓	✓
<i>Axis torque monitoring</i>	✗	✗	✓
<i>Base-related TCP force component</i>	✗	✗	✓
<i>Collision detection</i>	✗	✗	✓
<i>Torque referencing</i>	✗	✗	✓
<i>TCP force monitoring</i>	✗	✗	✓
<i>Hand guiding device enabling active</i>	✗	✗	✓
<i>Hand guiding device enabling inactive</i>	✗	✗	✓

13.9.1 Standard AMFs

Description

Standard AMFs provide information about system components or system states, e.g. the safety equipment on the smartPAD or the active operating mode. Standard AMFs can be used in any number of safety functions.

Overview

AMFs for evaluating the safety equipment on the smartPAD:

AMF	Task
<i>smartPAD Emergency Stop</i>	Monitors the EMERGENCY STOP device on the smartPAD
<i>smartPAD enabling switch inactive</i>	Checks whether the enabling signal has not been issued on the smartPAD.
<i>smartPAD enabling switch panic active</i>	Checks whether an enabling switch on the smartPAD has been pressed down fully (panic position).

(>>> [13.14.1 "Evaluating the safety equipment on the KUKA smartPAD"](#)
[Page 275](#))

AMFs for evaluating the operating mode:

AMF	Task
<i>Test mode</i>	Checks whether a test operating mode is active (T1, T2, CRR)
<i>Automatic mode</i>	Checks whether the Automatic operating mode is active (AUT)
<i>Reduced-velocity mode</i>	Checks whether an operating mode with reduced velocity is active (T1, CRR) Note: In the case of a mobile platform, the velocity is not reduced in T1 and CRR mode.
<i>High-velocity mode</i>	Checks whether an operating mode with programmed velocity is active (T2, AUT)

AMFs for evaluating the motion enable:

AMF	Task
<i>Motion enable</i>	Monitors the motion enable signal Motion enable is withdrawn if a safety stop is active. Note: The “Brake” safety reaction does not lead to withdrawal of motion enable.

(>>> [13.14.3 "Evaluating the motion enable" Page 276](#))

13.9.2 Parameterizable AMFs

Description

In contrast to standard AMFs, parameterizable AMFs additionally have one or more parameters. These can be configured depending on the values at which the AMF is to be considered violated (e.g. monitoring limits).

Up to 100 instances are available for each parameterizable AMF. In this way, differently parameterized versions of the AMF can be configured and used. The instance of an AMF may be used multiple times in the table in which the safety functions are configured.

Overview

AMF for evaluating safety-oriented inputs:

AMF	Task
<i>Input signal</i>	Monitors a safety-oriented input (>>> 13.14.4 "Monitoring safe inputs" Page 276)

AMFs for evaluating the enabling signal on the hand guiding device:

AMF	Task
<i>Hand guiding device enabling inactive</i>	Checks whether the enabling signal has not been issued on the hand guiding device.
<i>Hand guiding device enabling active</i>	Checks whether the enabling signal has been issued on the hand guiding device. The AMF is used to activate further monitoring functions during manual guidance with an enabling device.

(>>> [13.14.5 "Manual guidance with enabling device and velocity monitoring" Page 277](#))

AMFs for evaluating the referencing status:

AMF	Task
<i>Position referencing</i>	Monitors the referencing status of the position sensors of the axes of a kinematic system (>>> 13.14.6 "Evaluating the position referencing" Page 280)
<i>Torque referencing</i>	Monitors the referencing status of the axis torque sensors of a kinematic system (>>> 13.14.7 "Evaluating the torque referencing" Page 281)

AMFs for velocity monitoring:

AMF	Task
<i>Axis velocity monitoring</i>	Monitors the velocity of an axis (>>> 13.14.8.1 "Defining axis-specific velocity monitoring" Page 282)
<i>Cartesian velocity monitoring</i>	Monitors the Cartesian translational velocity at defined points of a kinematic system (>>> 13.14.8.2 "Defining Cartesian velocity monitoring" Page 283)
<i>Tool-related velocity component</i>	Monitors the Cartesian translational velocity in a specific defined direction (up to 6 monitoring points can be configured). (>>> 13.14.8.3 "Direction-specific monitoring of Cartesian velocity" Page 285)

AMFs for space monitoring:

AMF	Task
<i>Cartesian workspace monitoring</i>	Checks whether a part of the structure of a kinematic system being monitored is located outside of its permissible workspace (>>> 13.14.9.1 "Defining Cartesian workspaces" Page 290)
<i>Cartesian protected space monitoring</i>	Checks whether a part of the structure of a kinematic system being monitored is located within a non-permissible protected space (>>> 13.14.9.2 "Defining Cartesian protected spaces" Page 292)
<i>Axis range monitoring</i>	Monitors the position of one of the axes of a kinematic system (>>> 13.14.9.3 "Defining axis-specific monitoring spaces" Page 295)

AMF for monitoring the tool orientation:

AMF	Task
<i>Tool orientation</i>	Checks whether the orientation of the tool of a kinematic system is outside a permissible range (>>> 13.14.10 "Monitoring the tool orientation" Page 296)

AMFs for monitoring forces and torques (HRC):

AMF	Task
<i>Axis torque monitoring</i>	Monitors the measured torque of an axis (>>> 13.14.13.1 "Axis torque monitoring" Page 300)
<i>Collision detection</i>	Monitors the external axis torques of all axes of a kinematic system (>>> 13.14.13.2 "Collision detection" Page 301)
<i>TCP force monitoring</i>	Monitors the external force acting on the tool or robot flange of a kinematic system (>>> 13.14.13.3 "TCP force monitoring" Page 303)
<i>Base-related TCP force component</i>	Monitors a component of the external force acting on the tool or robot flange of a kinematic system relative to a base coordinate system. (>>> 13.14.13.4 "Direction-specific monitoring of the external force on the TCP" Page 304)

13.9.3 Extended AMFs

Description

An extended AMF differs from a standard AMF and a parameterizable AMF in that monitoring parameters are only defined during operation. The parameters are set at the time of activation. For the AMF *Standstill monitoring of all axes*, for example, the axis angles are set as reference angles for monitoring at the time of activation.

An extended AMF is activated if all other AMFs used by the safety function are violated. As long as at least one of the other AMFs is not violated, the extended AMF is not active and not evaluated.



Extended AMFs are only evaluated one cycle after they are activated. This can result in an extension of the reaction time by up to 12 ms.

Up to 100 instances are available for each extended AMF.



If the same instance of an Extended AMF is used in multiple rows of a PSM table, it is activated and remains activated as long as it is activated by at least one of these rows. It is advisable to use the instance of an extended AMF only once in the safety configuration.



Extended AMFs are not available for the safety functions of the ESM mechanism.

Overview

AMF for standstill monitoring:

AMF	Task
<i>Standstill monitoring of all axes</i>	Monitors the standstill of all axes of a kinematic system. (>>> 13.14.11 "Standstill monitoring (safe operational stop)" Page 299)

AMF for switching a delay:

AMF	Task
Time delay	Delays the triggering of the reaction of a safety function for a defined time. (>>> 13.14.12 "Activation delay for safety function" Page 300)

13.9.4 Availability of the AMFs depending on the kinematic system

Description

Some AMFs are kinematic-specific, i.e. the kinematic system to be monitored can be selected during configuration of these AMFs. (Parameter *Monitored kinematic system* with the values *First kinematic system ... Fourth kinematic system*)

The kinematic system to be monitored must be specified as follows in the kinematic-specific AMFs:

- *First kinematic system*: Robot
- *Second kinematic system*: Mobile platform
- *Third kinematic system*: No function
- *Fourth kinematic system*: No function

Overview

Not all kinematic-specific AMFs are available for monitoring a KMP, as the required sensor information is not available. If an AMF cannot be used, it is always violated.

AMF	LBR	KMP
Position referencing	✓	✗
Torque referencing	✓	✗
Axis velocity monitoring	✓	✓
Cartesian velocity monitoring	✓	✓
Tool-related velocity component	✓	✓
Cartesian workspace monitoring	✓	✗
Cartesian protected space monitoring	✓	✗
Axis range monitoring	✓	✗
Tool orientation	✓	✗
Axis torque monitoring	✓	✗
Collision detection	✓	✗
TCP force monitoring	✓	✗
Base-related TCP force component	✓	✗
Standstill monitoring of all axes	✓	✗

13.10 Worst-case reaction times of the safety functions in the case of a single fault

The reaction time describes the time between the following events:

- Time at which the event occurs that is to trigger a safety reaction, e.g. violation of a monitored axis range or setting of an EMERGENCY STOP input
- Time at which the safety reaction is initiated, e.g. stop reaction is initiated or an output has been deactivated

The reaction time thus contains fault detection times and delays before initiation of the safety reaction. The worst-case reaction time in the case of a single fault considers the presence of an individual fault and is thus greater than the reaction time typically expected for the safety function. The reaction time does not include the time between initiation of a stop reaction and the kinematic system coming to a standstill.

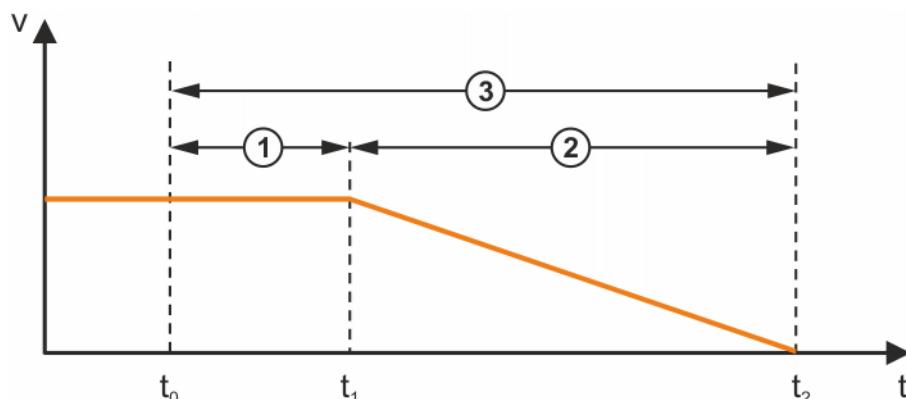


Fig. 13-4: Reaction time of a safety function

- 1 Reaction time
- 2 Braking time
- 3 Stopping time = Reaction time + Braking time
- v Velocity
- t Time
- t_0 Time at which the triggering event occurs
- t_1 Time at which the safety reaction is initiated
- t_2 Time at which the kinematic system comes to a standstill

The reaction time of a safety function depends on the monitoring function (AMF) used, the linked reaction and the monitored kinematic system.

For the stop reactions, the reaction time for the safety stop 0 is specified in each case. For safety stop 1 and safety stop 1 (path-maintaining), the reaction time may be longer in the case of defective stopping with the drives. This fault is detected by monitoring the braking ramps. The reaction time thus depends on the actual motion up to triggering of the braking ramp monitoring. Deactivation of the motor power can be delayed by a maximum of 1 second for safety stop 1 and safety stop 1 (path-maintaining).

If multiple monitoring functions (AMFs) are combined in a PSM table row, the monitoring function with the longest reaction time determines the reaction time of the safety function.

13.10.1 Worst-case reaction times of the LBR Med monitoring functions

Axis range monitoring

Reaction	Reaction time
Stop 0	27 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Input signal CIB_SR

Reaction	Reaction time
Stop 0	174 ms
CIB_SR output	328 ms
PROFIsafe output	143 ms + PROFIsafe master watchdog time
FSoE output	143 ms + FSoE master watchdog time

Input signal PROFIsafe

Reaction	Reaction time
Stop 0	67 ms + y*
CIB_SR output	245 ms + y*
PROFIsafe output	Watchdog time * [2 + Ceil(24 ms / watchdog time)]
FSoE output	36 ms + y* + FSoE master watchdog time

*: For PROFIsafe inputs, delay y must additionally be taken into consideration. This delay is dependent on the watchdog time of the PROFIsafe slave and is set by the PROFIsafe master:

$$y = 24 \text{ ms} * \text{Floor}(\text{slave watchdog time} / 12 \text{ ms})$$

Input signal FSoE

Reaction	Reaction time
Stop 0	67 ms + y**
CIB_SR output	245 ms + y**
PROFIsafe output	36 ms + y** + PROFIsafe master watchdog time
FSoE output	Watchdog time * [2 + Ceil(24 ms / watchdog time)]

**: For FSoE inputs, delay y must additionally be taken into consideration. This delay is dependent on the watchdog time of the FSoE slave and is set by the FSoE master:

$$y = 24 \text{ ms} * \text{Floor}(\text{slave watchdog time} / 12 \text{ ms})$$

Input signal media flange "Touch"

Reaction	Reaction time
Stop 0	174 ms
CIB_SR output	352 ms
PROFIsafe output	143 ms + PROFIsafe master watchdog time

Reaction	Reaction time
FSoE output	143 ms + FSoE master watchdog time

smartPAD Emergency Stop

Reaction	Reaction time
Stop 0	174 ms
CIB_SR output	352 ms
PROFIsafe output	143 ms + PROFIsafe master watchdog time
FSoE output	143 ms + FSoE master watchdog time

smartPAD enabling switch panic active

Reaction	Reaction time
Stop 0	174 ms
CIB_SR output	352 ms
PROFIsafe output	143 ms + PROFIsafe master watchdog time
FSoE output	143 ms + FSoE master watchdog time

smartPAD enabling switch inactive

Reaction	Reaction time
Stop 0	174 ms
CIB_SR output	352 ms
PROFIsafe output	143 ms + PROFIsafe master watchdog time
FSoE output	143 ms + FSoE master watchdog time

Axis velocity monitoring

Reaction	Reaction time
Stop 0	32 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Cartesian workspace monitoring

Reaction	Reaction time
Stop 0	27 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Cartesian velocity monitoring

Reaction	Reaction time
Stop 0	32 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Cartesian protected space monitoring

Reaction	Reaction time
<i>Stop 0</i>	27 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Standstill monitoring of all axes

Reaction	Reaction time
<i>Stop 0</i>	27 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Tool-related velocity component

Reaction	Reaction time
<i>Stop 0</i>	32 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Tool orientation

Reaction	Reaction time
<i>Stop 0</i>	27 ms
CIB_SR output	258 ms
PROFIsafe output	49 ms + PROFIsafe master watchdog time
FSoE output	49 ms + FSoE master watchdog time

Axis torque monitoring

Reaction	Reaction time
<i>Stop 0</i>	27 ms
CIB_SR output	294 ms
PROFIsafe output	85 ms + PROFIsafe master watchdog time
FSoE output	85 ms + FSoE master watchdog time

Base-related TCP force component

Reaction	Reaction time
<i>Stop 0</i>	27 ms + x***
CIB_SR output	258 ms + x***
PROFIsafe output	49 ms + PROFIsafe master watchdog time + x***
FSoE output	49 ms + FSoE master watchdog time + x***

***: With this monitoring function, an additional detection time x must be taken into account for collision detection, as the collision forces are not

measured directly. Detection of the actual collision forces is carried out approximately with a delay of a PT₁ element with the time constant T=1/30 s.

Collision detection

Reaction	Reaction time
Stop 0	27 ms + x***
CIB_SR output	258 ms + x***
PROFIsafe output	49 ms + PROFIsafe master watchdog time + x***
FSoE output	49 ms + FSoE master watchdog time + x***

***: With this monitoring function, an additional detection time x must be taken into account for collision detection, as the collision forces are not measured directly. Detection of the actual collision forces is carried out approximately with a delay of a PT₁ element with the time constant T=1/30 s.

TCP force monitoring

Reaction	Reaction time
Stop 0	27 ms + x***
CIB_SR output	258 ms + x***
PROFIsafe output	49 ms + PROFIsafe master watchdog time + x***
FSoE output	49 ms + FSoE master watchdog time + x***

***: With this monitoring function, an additional detection time x must be taken into account for collision detection, as the collision forces are not measured directly. Detection of the actual collision forces is carried out approximately with a delay of a PT₁ element with the time constant T=1/30 s.

Hand guiding device enabling active

The reaction time depends on the input used to connect the enabling device on the hand guiding device to the robot controller. The reaction time corresponds to the reaction time of the corresponding AMF *Input signal*.

Hand guiding device enabling inactive

The reaction time depends on the input used to connect the enabling device on the hand guiding device to the robot controller. The reaction time corresponds to the reaction time of the corresponding AMF *Input signal*.

13.11 Deactivation of safety functions via an input

Description

A safety-oriented input can be configured in the safety-oriented project settings to allow the deactivation of safety functions. A safety stop triggered by one of the following AMFs can be briefly cancelled:

- *Axis range monitoring*
- *Cartesian workspace monitoring*
- *Cartesian protected space monitoring*
- *Tool orientation*

- *Tool-related velocity component*
- *Standstill monitoring of all axes*
- *Position referencing*
- *Torque referencing*
- *Axis torque monitoring*
- *Collision detection*
- *TCP force monitoring*
- *Base-related TCP force component*

Use

Deactivation of safety functions may be used, for example, for freeing persons in a crushing situation.

- To cancel a safety stop triggered by one of the defined AMFs, the configured input must be set to HIGH.
- As long as the input is HIGH, the robot can be moved for a maximum of 5 seconds. Every further safety stop triggered by one of the defined AMFs in this time does not become active.
- After this time, the input must be reset and set again.

Velocity monitoring

While the safety functions are deactivated, all axis-specific velocity monitoring functions and the Cartesian velocity monitoring function remain active.

For all kinematic systems, safety-oriented monitoring of the Cartesian velocity of 250 mm/s of the robot and tools is additionally active. This additional Cartesian velocity monitoring is active irrespective of the operating mode.

Procedure

1. Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Change project settings** from the context menu.
The **Properties for [Sunrise Project]** window opens.
2. Select **Sunrise > Safety** in the directory in the left area of the window.
3. Make the following settings in the right-hand part of the window:
 - Set the check mark at **Allow muting via input**.
 - Select the input that is to allow the deactivation of safety functions.
The inputs of the discrete safety interface and of the Ethernet safety interface can be used as long as they are configured in WorkVisual.

(>>> [13.6 "Safety interfaces" Page 245](#))



The enabling device of the hand guiding device can be used as an input for deactivating safety functions. In this case, it must be taken into consideration in the risk assessment that every time the enabling switch on the hand guiding device is used, a safety stop that is active at the time of the enabling can be cancelled if it was triggered by one of the defined safety functions.

4. Click on **OK** to save the settings and close the window.

13.12 Safety configuration (SafetyConfiguration.sconf file)

The safety configuration is an integral feature of a Sunrise project. It is managed in tabular form.

When creating a new Sunrise project, a standard safety configuration is automatically generated (SafetyConfiguration.sconf file).

The standard safety configuration contains permanently active safety functions predefined by KUKA.



Further information on the standard safety configuration can be found in the "Safety" chapter.

Further safety functions and safe ESM states can be configured. Safety-oriented tools can also be mapped.

After the safety configuration has been transferred to the robot controller, it must be activated and safety acceptance must be carried out.

13.12.1 Overview of safety configuration and start-up

Step	Description
1	Open the safety configuration. (>>> 13.12.2 "Opening the safety configuration" Page 261)
2	Edit the safety functions in the <i>Customer PSM</i> table or create new safety functions. (>>> 13.12.3 "Configuring the safety functions of the PSM mechanism" Page 264)
3	If necessary, adapt in the <i>KUKA PSM</i> table the parameters of the parameterizable AMFs used, e.g. for the Cartesian velocity monitoring during manual guidance. (>>> 13.14.5.3 "Velocity monitoring during manual guidance" Page 280)
4	Configure event-dependent monitoring functions if required. To do so, create safe ESM states and corresponding safety functions. Existing ESM states can be changed by adapting safety functions which are already configured or by adding new ones. (>>> 13.12.4 "Configuring the safe states of the ESM mechanism" Page 267)
5	If necessary, map safety-oriented tools. (>>> 13.12.5 "Mapping safety-oriented tools" Page 271)
6	Save safety configuration.
7	When using the ESM mechanism Program the necessary switch between the safe states in the robot application and/or background applications. (>>> 13.12.4.8 "Switching between ESM states" Page 270)

Step	Description
8	<p>When using position-based AMFs (>>> "Position-based AMFs" Page 321)</p> <p>Create the application for the position and torque referencing of the LBR or an application of your own for the reference run.</p> <p>(>>> 13.16.1 "Position referencing" Page 312)</p>
9	<p>When using axis torque-based AMFs (>>> "Axis torque-based AMFs" Page 321)</p> <p>Create the application for the position and torque referencing of the LBR and integrate the safety-oriented tool into the application. Further adaptations in the application may be necessary.</p> <p>(>>> 13.16.2 "Torque referencing" Page 313)</p>
10	<p>Transfer the safety configuration to the robot controller.</p> <ul style="list-style-type: none"> • By installation of the System Software or by project synchronization
11	Reboot the robot controller to apply the safety configuration.
12	<p>Activate the safety configuration on the robot controller</p> <p>(>>> 13.13.1 "Activating the safety configuration" Page 274)</p>
13	<p>When using position-based AMFs (>>> "Position-based AMFs" Page 321)</p> <p>Carry out position referencing.</p>
14	<p>When using axis torque-based AMFs (>>> "Axis torque-based AMFs" Page 321)</p> <p>Carry out torque referencing.</p>
15	<p>Carry out safety acceptance.</p> <p>(>>> 13.17 "Safety acceptance overview" Page 317)</p>

13.12.2 Opening the safety configuration

Procedure

- In the Sunrise project, double-click on the file **SafetyConfiguration.sconf**.

Description

The safety configuration contains several tables.

• KUKA PSM table

The table contains the safety functions prescribed by KUKA. These cannot be deactivated or deleted. The reactions are permanently configured. The parameters of the parameterizable AMFs used can be changed.

The table documents the system behavior and, in conjunction with the *Customer PSM* table, provides a full description of the permanently active safety functions.

- *Customer PSM* table

The user-specific safety functions are configured in this table. It contains the safety functions preconfigured by KUKA. These can be deactivated, changed or deleted.

- Tables for ESM states (optional)

A table is created for each ESM state. It contains the safety functions of the state. The standard configuration does not contain any preconfigured ESM states.

- *Tool selection table* table

Safety-oriented tools can be mapped in this table. Each kinematic system can be assigned a maximum of one fixed tool that is always active and one or more tools that can be activated via an input.

13.12.2.1 Evaluating the safety configuration

When the safety configuration is evaluated, the *Customer PSM* and *KUKA PSM* tables are always checked simultaneously. It is possible for the two tables to contain identical safety functions with different reactions. If different stop reactions are configured, a violation triggers the stronger stop reaction. In other words, it triggers the stop reaction which causes an earlier safety-oriented disconnection of the drives.

If the ESM mechanism is used, all safety functions of the currently active ESM state are additionally monitored.

13.12.2.2 Overview of the graphical user interface for the safety configuration

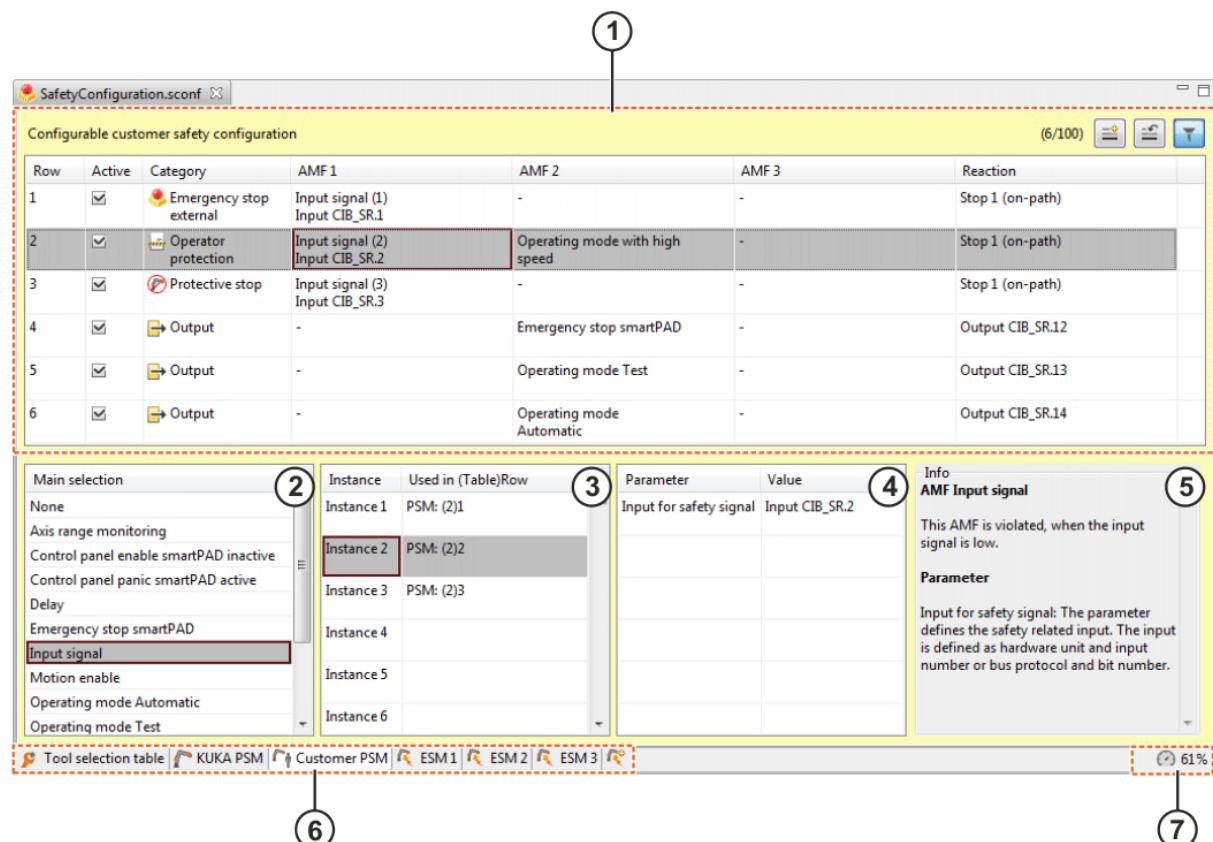


Fig. 13-5: Graphical user interface for safety configuration

The description of the user interface elements refers to the configuration of safety functions. The tool selection table is described separately.

Item	Description
1	Table selected Contains the configured safety functions of the selected PSM table or of the selected ESM state.
2	Selection table With respect to the cell selected in the highlighted table row, the category, AMF or reaction of a safety function can be selected here.
3	Instance table This area displays the instances of the AMF marked in the selection table as well as the table rows in which they are used.
4	Parameter table The parameter values of the AMF instance selected in the instance table are displayed here. The values can be changed.
5	Information display Information about the selected category, AMF or reaction
6	List of tables In this area, the desired tables can be selected and new ESM states can be added.
7	Computing time utilization of the safety controller Indicates the percentage of the computing time used for the open safety configuration, including all changes that have not been saved.

List of tables

The list of tables in the lower area of the Editor is used to select the table to be displayed and edited.

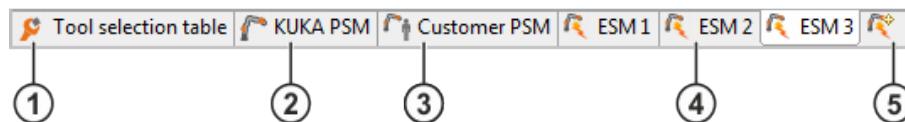


Fig. 13-6: List of tables

Item	Description
1	<i>“Tool selection table” tab</i> Opens the <i>Tool selection table</i> table. Safety-oriented tools can be mapped.
2	<i>“KUKA PSM” tab</i> Opens the <i>KUKA PSM</i> table. The parameters of the parameterizable AMFs used can be changed.
3	<i>“Customer PSM” tab</i> Opens the <i>Customer PSM</i> table. Safety functions can be modified and created.
4	Tab for an ESM state Opens the ESM state. The ESM state can be edited.

Item	Description
5	Add new ESM state button Adds a new ESM state. The new state is automatically opened and can be edited.

13.12.3 Configuring the safety functions of the PSM mechanism

The PSM mechanism defines safety monitoring functions which are permanently active.

The safety functions are displayed in tabular form. Each row in the table contains a safety function.

In the PSM table **Customer PSM**, new safety functions are added and existing settings are adapted. This means that the category, the Atomic Monitoring Functions (AMFs) used, the parameterization of the AMF instances and the reaction can be changed. Individual safety functions can be activated or deactivated.

13.12.3.1 Opening the *Customer PSM* table

Procedure

1. Open the safety configuration.
2. Select the *Customer PSM* tab from the list of tables. The table is displayed and can be edited.

The diagram shows the 'Configurable customer safety configuration' table with the following numbered elements:

- (1) Active column header: A column where safety functions can be activated or deactivated.
- (2) Category column header: A column defining the category of each safety function.
- (3) AMF 1 column header: A column for the first Atomic Monitoring Function instance.
- (4) Reaction column header: A column defining the reaction to an error.
- (5) AMF 2 column header: A column for the second Atomic Monitoring Function instance.
- (6) AMF 3 column header: A column for the third Atomic Monitoring Function instance.
- (7) Row 1: A row for an 'External EMERGENCY STOP' function. It includes a checkmark in the Active column, a yellow icon in the Category column, and a detailed description in the AMF 1 column: 'Input signal (1)', 'Input CIB_SR.1'. The Reaction column shows 'Stop 1 (path-maintaining)'.

Fig. 13-7: PSM table *Customer PSM*

Item	Description
1	Active column Defines whether the safety function is active. Deactivated safety functions are not monitored. <ul style="list-style-type: none"> • Check box active: safety function is active. • Check box not active: safety function is deactivated.
2	Category column Defines the category of the safety function. In the event of an error, the category is shown on the smartHMI as the cause of error.

Item	Description
3	Columns <i>AMF 1, AMF 2, AMF 3</i> Define the individual AMFs of the safety function. Up to 3 AMFs can be used. The safety function is violated if all of the AMFs used are violated.
4	<i>Reaction column</i> Defines the reaction of the safety function. It is triggered if the safety function is violated.
5	Number of safety functions currently configured A total of 100 rows are available for configuring the user-specific safety monitoring functions.
6	Buttons for editing the table
7	Selected row The row containing the currently selected safety function is highlighted in gray.

The following buttons are available:

Button	Description
	Add row Adds a new row to the table (only possible when the non-configured blank rows are hidden). The new row has the standard configuration and is activated automatically.
	Reset row Resets the configuration of the selected row to the standard configuration. The safety function is deactivated.
	Show empty rows/Hide empty rows All empty rows which are not configured are deactivated and preset with the standard configuration. <ul style="list-style-type: none">• <i>Category</i>: None• <i>AMF 1, AMF 2, AMF 3</i>: None• <i>Reaction</i>: Stop 1 The empty rows can be shown or hidden. The empty rows are hidden by default.

13.12.3.2 Creating safety functions for the PSM mechanism

Precondition

- The *Customer PSM* table is open.

Procedure

Non-configured empty rows are displayed:

1. Select an empty row in the table.
2. Set the category, the AMFs used and the reaction of the safety function in the corresponding columns.
3. Set the check mark in the *Active* column if the row is to be activated.

Non-configured empty rows are not displayed:

1. Click on **Add row**. A preconfigured row is added to the table. The row is automatically activated (check mark in the *Active* column).
2. Set the category, the AMFs used and the reaction of the safety function in the corresponding columns.

13.12.3.3 Deleting safety functions of the PSM mechanism

Precondition

- The *Customer PSM* table is open.

Procedure

1. In the table, select the row with the safety function to be deleted.
2. Click on **Reset row**. The safety function is deactivated and is given the standard configuration (None, AMF, Reaction: Stop 1).

13.12.3.4 Editing existing safety functions of the PSM mechanism

Precondition

- The *Customer PSM* table is open.

Procedure

Changing the category:

1. Select the *Category* column in the desired row. The available categories are displayed in the Main Selection table.
2. Select the desired category from the Main selection table. The category is applied to the safety function.

Changing the AMF used:

1. Select the *AMF 1*, *AMF 2* or *AMF 3* column in the desired row. The available AMFs are displayed in the Main selection table.
2. Select the desired AMF from the Main selection table. The AMF is applied to the safety function.
3. For multiply instanced AMFs: select the desired instance from the Main selection table. The instance is applied to the safety function.
4. For parameterizable AMFs: in the parameter table, set the parameter of the AMF in the *Value* column and insert the settings with the Enter key.

Changing a reaction:

1. Select the *Reaction* column in the desired row. The available reactions are displayed in the Main selection table.
2. Select the desired reaction from the Main selection table. The reaction is applied to the safety function.
3. If the **Output** reaction has been selected: in the Parameter table, select the output bit whose signal is to be set to LOW if a safety function is violated. Accept the setting with the Enter key.

Activating/deactivating a safety function:

- Click on the *Active* column in the desired row. The check mark is set / removed.



Once the safety configurations are transferred to the robot controller and activated, only the activated safety functions are available.

13.12.4 Configuring the safe states of the ESM mechanism

Using the ESM mechanism, various safety settings are defined by configurable safe states. Up to 10 safe states can be created. The states are numbered sequentially from 1 to 10 and can therefore be identified unambiguously.

A safe state is defined in a table with up to 20 safety functions. These safety functions define the safety settings which must be valid for the state.

A safe state is represented in a table. Each row in the table contains a safety function.

Use of the ESM mechanism is optional. The ESM mechanism is activated if at least one ESM state is configured. If no ESM states are configured, the mechanism is deactivated.

If the ESM mechanism is active, exactly one safe state is valid. The safety functions of this state are monitored in addition to the permanently active safety functions. Depending on the situation, it is possible to switch between the configured safe states. Switching can be carried out in the robot application or in a background application.

(>>> [13.12.4.8 "Switching between ESM states" Page 270](#))

An ESM state is active until it is commanded to switch to another ESM state.

The configured ESM state with the lowest number is automatically active when the controller is booted.

13.12.4.1 Adding a new ESM state

Description

Up to 10 safe states can be created for the ESM mechanism. If this number is reached, the tab for adding new states is hidden.

Procedure

1. Open the safety configuration.
2. Select the **Add new ESM state** button in the table directory. A new ESM state is created.

The new ESM state is given the lowest state number which has not yet been assigned. It has an active safety function with a standard configuration. A new tab for the state is added to the list of tables.

The table for the state is automatically opened and can be edited.

13.12.4.2 Opening a table for an ESM state

Precondition

- The ESM mechanism is activated.

Procedure

1. Open the safety configuration.
2. Select the tab for the desired ESM state from the list of tables. The table for the ESM state is automatically displayed and can be edited.

Row	Active	AMF	Reaction
1	<input checked="" type="checkbox"/>	Collision detection (I) Maximum external torque : 15 Nm	Stop 1
2	<input checked="" type="checkbox"/>	Cartesian velocity monitoring (I) Maximum velocity : 250 mm/s	Stop 1

Fig. 13-8: Table for an ESM state

Item	Description
1	<p>Active column Defines whether the safety function is active. Deactivated safety functions are not monitored.</p> <ul style="list-style-type: none"> Check box active: safety function is active. Check box not active: safety function is deactivated. <p>The safety function in the first row of the table is always active. It cannot be deactivated (indicated by the lock icon).</p>
2	<p>AMF column Defines the AMF of the safety function. Only one AMF is used for safety functions of ESM states. If this AMF is violated, the safety function and thus the entire state is violated.</p>
3	<p>Reaction column Defines the reaction of the safety function. It is triggered if the safety function is violated.</p>
4	<p>Number of safety functions currently configured A total of 20 rows are available for configuring the safety monitoring functions of an ESM state.</p>
5	Buttons for editing the table
6	<p>Selected row The row containing the currently selected safety function is highlighted in gray.</p>

Buttons

The following buttons are available:

Button	Description
	<p>Delete state Deletes the entire state. The delete operation must be confirmed via a dialog.</p>
	<p>Add row Adds a new row to the table (only possible when the non-configured blank rows are hidden). The new row has the standard configuration and is activated automatically.</p>

Button	Description
	<p>Reset row</p> <p>Resets the configuration of the selected row to the standard configuration. The safety function is deactivated (exception: the first row of the table is always active).</p>
	<p>Show empty rows/Hide empty rows</p> <p>All empty rows which are not configured are deactivated and preset with the standard configuration.</p> <ul style="list-style-type: none"> • <i>AMF:</i> None • <i>Reaction:</i> Stop 1 <p>The empty rows can be shown or hidden. The empty rows are hidden by default.</p>

13.12.4.3 Deleting an ESM state

Procedure

1. Open the safety configuration.
2. Select the tab for the ESM state to be deleted from the list of tables.
3. Click on **Delete state**.
4. Reply to the request for confirmation with **Yes**. The state is deleted.

Once the safety configuration is saved and closed, an ESM state is automatically removed if it has the following settings:

- All rows have the standard configuration (AMF: None, Reaction: Stop 1)
- The first row is activated and all other rows are deactivated.

13.12.4.4 Creating a safety function for the ESM state

Precondition

- The table for the desired ESM state is open.

Procedure

Non-configured empty rows are displayed:

1. Select an empty row in the table.
2. Set the AMF used and the reaction of the safety function in the corresponding columns.
3. Set the check mark in the *Active* column if the row is to be activated.

Non-configured empty rows are not displayed:

1. Click on **Add row**. A preconfigured row is added to the table. The row is automatically activated (check mark in the *Active* column).
2. Set the AMF used and the reaction of the safety function in the corresponding columns.

13.12.4.5 Deleting a safety function of an ESM state

Precondition

- The table for the desired ESM state is open.

Procedure

1. In the table, select the row with the safety function to be deleted.
2. Click on **Reset row**. The safety function is deactivated and is given the standard configuration (None, AMF, Reaction: Stop 1).

13.12.4.6 Editing an existing safety function of an ESM state

Precondition

- The table for the desired ESM state is open.

Procedure

Changing the AMF used:

1. Select the *AMF* column in the desired row. The available AMFs are displayed in the Main selection table.
2. Select the desired AMF from the Main selection table. The AMF is applied to the safety function.
3. For multiply instanced AMFs: select the desired instance from the Main selection table. The instance is applied to the safety function.
4. For parameterizable AMFs: in the parameter table, set the parameter of the AMF in the **Value** column and insert the settings with the Enter key.

Changing a reaction:

1. Select the *Reaction* column in the desired row. The available reactions are displayed in the Main selection table.
2. Select the desired reaction from the Main selection table. The reaction is applied to the safety function.

Activating/deactivating a safety function:

- Click on the *Active* column in the desired row. The check mark is set / removed.

13.12.4.7 Deactivating the ESM mechanism

Description

Use of the ESM mechanism is optional. It can be deactivated.

Procedure

- Delete all ESM states.

13.12.4.8 Switching between ESM states

Description

The `setESMState(...)` method can be used to activate an ESM state and switch between the different ESM states. The method belongs to the LBR class and can be used in both a robot application and a background application.

Syntax

```
robot.setESMState(state);
```

Explanation of the syntax

Element	Description
<code>robot</code>	Type: LBR Name of the robot for which the ESM state is activated
<code>state</code>	Type: String Number of the ESM state which is activated • 1 ... 10 If a non-configured ESM state is specified, the robot stops with a safety stop 1.

Example

In an application, the LBR is to be guided by hand. For this purpose, a suitable start position is addressed. In order to address the start position, ESM state 3 must be activated. ESM state 3 ensures sensitive collision detection and monitors the Cartesian velocity.

Manual guidance is to begin once the start point has been reached. ESM state 8 must be activated for manual guidance. ESM state 8 requires enabling on the hand guiding device but permits a higher Cartesian velocity than ESM state 3.

```
@Inject
private LBR robot;
// ...

public void run() {
// ...
robot.setESMState("3");
robot.move(lin(getFrame("Start")).setCartVelocity(300));

robot.setESMState("8");
robot.move(handGuiding());
// ...
}
```

13.12.5 Mapping safety-oriented tools

Description

Each kinematic system can be assigned a maximum of one fixed safety-oriented tool that is always active and one or more safety-oriented tools that can be activated via an input.

- Assignment of a fixed tool (always active)

A fixed tool is coupled to the flange of the configured kinematic system and cannot be uncoupled or changed. The fixed tool can be a machining tool, a tool for picking up workpieces or a tool that can pick up other tools, e.g. a tool changer.

The assignment of multiple fixed tools to a kinematic system is not allowed. In this case, all tool-dependent monitoring functions of this kinematic system enter the safe state.

- Assignment of tools that can be activated (via an input)

The tool is activated when the configured input signal is HIGH.

If a fixed tool is configured for this kinematic system, the activatable tool is coupled to the pickup frame of the fixed tool (standard frame

for motions). If no fixed tool is configured for a kinematic system, it is coupled to the flange of the kinematic system.

If an activatable tool is configured for a kinematic system, exactly one activatable tool must always be active for this kinematic system. This means that exactly one of the input signals configured for this kinematic system must be HIGH.

If multiple activatable tools are active simultaneously, or if none of the activatable tools is active, all tool-dependent monitoring functions of this kinematic system enter the safe state. For this reason, the tool **No tool** must be activated if the activatable tool is uncoupled.

Procedure

1. Open the safety configuration.
2. Select the *Tool selection table* tab from the list of tables. Map the tools as desired.
3. Save the safety configuration.

Overview

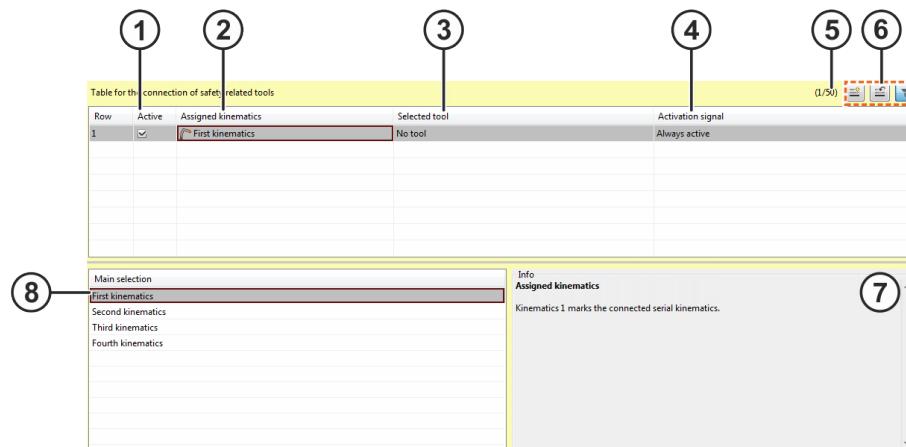


Fig. 13-9: *Tool selection table* table

Item	Description
1	State of the mapped tool <ul style="list-style-type: none"> • Check box active: the tool is always active or activatable. • Check box not active: the tool is deactivated.
2	Kinematic system to which the tool is assigned <ul style="list-style-type: none"> • <i>First kinematic system</i>: robot • <i>Second kinematic system</i>: mobile platform • <i>Third kinematic system</i>: no function • <i>Fourth kinematic system</i>: no function
3	Tool assigned to the kinematic system <ul style="list-style-type: none"> • No tool: no tool is assigned to the kinematic system. • All safety-oriented tools defined in the object templates are available for selection.

Item	Description
4	<p>Activation of the tool</p> <ul style="list-style-type: none"> <i>Always active</i>: the tool is always active. A maximum of 1 fixed tool can be assigned to each kinematic system. The tool can be activated via a safe input. The safe inputs of the Ethernet safety interface used are available.
5	<p>Number of tools currently mapped</p> <p>A total of 50 rows are available for mapping.</p>
6	Buttons for editing the table
7	<p>Information display</p> <p>Information about the selected parameter</p>
8	<p>Selection table</p> <p>The table contains the values available for the parameter selected in the configuration line.</p>

The following buttons are available:

Button	Description
	<p>Add row</p> <p>Adds a new row to the table (only possible when the non-configured blank rows are hidden). The new row has the standard configuration and is activated automatically.</p>
	<p>Reset row</p> <p>Resets the configuration of the selected row to the standard configuration. The mapped tool is activated.</p>
	<p>Show empty rows/Hide empty rows</p> <p>All empty rows which are not configured are deactivated and preset with the standard configuration.</p> <ul style="list-style-type: none"> <i>Assigned kinematic system: First kinematic system</i> <i>Selected tool: No tool</i> <i>Activation signal: Always active</i> <p>The empty rows can be shown or hidden. The empty rows are hidden by default.</p>

13.13 Activating the safety configuration

The safety configuration on the robot controller must be activated. If no safety configuration is active, the robot cannot be moved.

When it is activated, the safety configuration is assigned a unique ID (= checksum of the safety configuration) and displayed under **Safety config ID**. With this ID, the safety maintenance technician can clearly identify the safety configuration activated on the robot controller.

A new safety configuration can be transferred to the robot controller by installing the system software or by way of project synchronization. After a reboot of the robot controller, the old safety configuration is no longer active and the new safety configuration is not yet active:

- The new safety configuration must be activated.
- If the new safety configuration is not to be activated, the last active safety configuration can be restored.



After activating the safety configuration, the safety maintenance technician must carry out a safety acceptance procedure in order to verify the safety configuration. (>> [13.17 "Safety acceptance overview" Page 317](#))



If the activated safety configuration contains deactivated rows, i.e. if safety functions in the PSM table or in an ESM state are deactivated, a warning message is displayed on the **Safety** tile. Before using the safety configuration, it is advisable to check whether the deactivation of the safety functions is desirable and permissible.

13.13.1 Activating the safety configuration

Precondition

- User group “Safety maintenance”

Procedure

1. Select the **Safety > Activation** tile at the Station level. The **Activation** view opens.
2. Press the **Activate** button.

13.13.2 Restoring the safety configuration

Description

If a new safety configuration is transferred to the robot controller, but is not to be activated, the most recently active safety configuration can be restored.

Precondition

- User group “Safety maintenance”

Procedure

1. Select the **Safety > Activation** tile at the Station level. The **Activation** view opens.
2. Press the **Reset** button.

13.13.3 Deactivating the safety configuration

Description

An active safety configuration can be deactivated again. Following successful deactivation, the robot can no longer be moved. A corresponding message is displayed under the **Safety** tile.

Precondition

- User group “Safety maintenance”

Procedure

1. Select the **Safety > Activation** tile at the Station level. The **Activation** view opens.
2. Press the **Deactivate** button.
3. Check whether the safety configuration has been deactivated successfully.

13.14 Using and parameterizing the AMFs

Up to 100 instances are available for each parameterizable AMF. As the processing power of the safety controller is limited, this quantity cannot be used to the full in practice.

- Each instance of the AMF used in the safety configuration requires part of the available processing power. The processing time required by an AMF instance depends, for example, on the number of parameters and the complexity of the corresponding calculations.
- How often an AMF instance is used in the safety configuration, how many lines are used in the *Customer PSM* table and how many ESM states are used are not relevant for the processing power.

Response if the processing power of the safety controller is exceeded:

- The required processing time of a safety configuration is calculated automatically on saving the safety configuration. If it is too great, a warning is displayed. It is nonetheless saved.
- The transfer of an excessively large safety configuration to the robot controller is prevented. Project synchronization and installation of the system software are canceled in this case with a corresponding error message.

13.14.1 Evaluating the safety equipment on the KUKA smartPAD

The smartPAD has an EMERGENCY STOP device and an enabling device. The corresponding safety-oriented functions are preconfigured in the *KUKA PSM* table and cannot be changed.

Further safety functions evaluated by the safety equipment on the smartPAD can be configured. The following standard AMFs are available for this purpose:

AMF	Description
<i>smartPAD Emergency Stop</i>	The AMF is violated if the EMERGENCY STOP device on the smartPAD is pressed.
<i>smartPAD enabling switch inactive</i>	The AMF is violated if no enabling signal is issued on the smartPAD (no enabling switch is pressed on the smartPAD or an enabling switch is fully pressed).
<i>smartPAD enabling switch panic active</i>	The AMF is violated if an enabling switch on the smartPAD is fully pressed (panic position).

13.14.2 Evaluating the operating mode

The set operating mode has a powerful effect on the behavior of the robot and determines which safety precautions are required.

The following standard AMFs are available for configuring a safety function to evaluate the set operating mode:

AMF	Description
<i>Test mode</i>	The AMF is violated if a test operating mode is active (T1, T2, CRR).
<i>Automatic mode</i>	The AMF is violated if the active operating mode is an automatic mode (AUT).
<i>Reduced-velocity mode</i>	The AMF is violated if an operating mode is active whose velocity is reduced to a maximum of 250 mm/s (T1, CRR). Note: In the case of a mobile platform, the velocity is not reduced in T1 and CRR mode.
<i>High-velocity mode</i>	The AMF is violated if an operating mode is active in which the robot is moved at a programmed velocity (T2, AUT).

13.14.3 Evaluating the motion enable

Description

The robot cannot be moved without the motion enable. The motion enable can be cancelled for various reasons, e.g. if enabling is not issued in Test mode or if the EMERGENCY STOP is pressed on the smartPAD.

The AMF for motion enable functions like a group signal for all configured stop conditions. In particular, it can be used for switching off peripheral devices. For safety functions which receive the evaluation of the motion enable, a safe output should therefore be configured as the reaction. If a safety stop is set as the reaction, the robot cannot be moved.

AMF	Description
<i>Motion enable</i>	The AMF is violated if the motion enable is not issued due to a stop request. Note: This AMF is only suitable for use with an output as a reaction.

Example

Switching off a tool (category: *Output*)

A tool (e.g. a laser) which is connected to an output is to be switched off when the motion enable is cancelled. It is only to be switched off if the operator safety is violated.

AMF1	AMF2	AMF3	Reaction
<i>Input signal</i> (operator safety)	-	<i>Motion enable</i>	<i>Output</i> (tool)

13.14.4 Monitoring safe inputs

Description

The inputs of the discrete safety interface and of the Ethernet safety interface can be used as safe inputs as long as they are configured in WorkVisual.

(>>> [13.6 "Safety interfaces" Page 245](#))

Safety equipment can be connected to the safe inputs, e.g. external EMERGENCY STOP devices or safety gates. The AMF *Input signal* is used to evaluate the associated input signal.

AMF	Description
<i>Input signal</i>	The AMF is violated if the monitored input is low (state "0").



If a robot with a media flange Touch is used, the safety-oriented inputs at which enabling and panic switches for the media flange are connected can be used in the AMFs.

Parameter	Description
<i>Input for safety signal</i>	Safety-oriented input to be monitored

Example 1

Operator safety (category: *Operator safety*)

A safety gate is connected to a safety-oriented input. If the safety gate is opened in Automatic or T2 mode, a safety stop 1 (path-maintaining) is to be triggered.

AMF1	AMF2	AMF3	Reaction
<i>Input signal</i>	<i>High-velocity mode</i>	-	<i>Stop 1 (path-main-taining)</i>

Example 2

External E-STOP (category: *External EMERGENCY STOP*)

An external EMERGENCY STOP device is connected to a safety-oriented input. If the external EMERGENCY STOP device is pressed, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
<i>Input signal</i>	-	-	<i>Stop 1 (path-main-taining)</i>

13.14.5 Manual guidance with enabling device and velocity monitoring

An application of human-robot collaboration involves manually guiding the robot, e.g. to teach points on a path. This requires a hand guiding device with a safety-oriented enabling device.

For manual guidance, safety-oriented velocity monitoring with a maximum permissible velocity of 250 mm/s is preconfigured. The maximum permissible velocity can be adapted.

The maximum permissible velocity during manual guidance must be defined in a risk assessment.

(>>> [13.14.5.3 "Velocity monitoring during manual guidance" Page 280](#))



CAUTION

If the robot is manually guided, an EMERGENCY STOP device must be installed. It must always be within reach of the operator.

13.14.5.1 Monitoring of enabling switches on hand guiding devices

Description

The AMF *Hand guiding device enabling inactive* serves to evaluate 3-step enabling devices. Up to 3 enabling switches and 3 panic switches can be configured. 3-step enabling devices with only one output which process the panic signal internally can also be evaluated.

The AMF fulfils the following normative requirements and measures against predictable misuse:

- If the enabling switch has been fully pressed down, the signal will not be issued if the switch is released to the center position.
- The signal is cancelled in case of a stop request. To issue the signal again, the enabling switch must be released and pressed again.
- The signal is only issued 100 ms after the enabling switch has been pressed.

The following applies if several enabling switches are used:

- If all 3 enabling switches of an enabling device are held simultaneously in the center position, a safety stop 1 is triggered.
- It is possible to hold 2 enabling switches of an enabling device 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 the enabling switches are held simultaneously in the center position for longer than 15 seconds, this triggers a safety stop 1.
- If the enabling switches of different enabling devices are pressed simultaneously, e.g. an enabling switch on the smartPAD and an enabling switch on the hand guiding device, a safety stop 1 (path-maintaining) is triggered.

AMF	Description
<i>Hand guiding device enabling inactive</i>	<p>The AMF is violated in the following cases:</p> <ul style="list-style-type: none"> • All safe inputs to which an enabling switch is connected have the signal level LOW (state "0") • At least one of the safe inputs to which a panic switch is connected has the signal level LOW (state "0")

 If a robot with a media flange Touch is used, the safety-oriented inputs at which enabling and panic switches for the media flange are connected can be used in the AMFs.

Parameter	Description
<i>Enabling switch 1 used</i>	Indicates whether the enabling switch is connected to a safe input
<i>Enabling switch 2 used</i>	
<i>Enabling switch 3 used</i>	<ul style="list-style-type: none"> • true: an input is connected • false: no input is connected <p>Default: false</p>
<i>Enabling switch 1 input signal</i>	Safe input to which the enabling switch is connected
<i>Enabling switch 2 input signal</i>	
<i>Enabling switch 3 input signal</i>	<p>The inputs of the discrete safety interface or the safe inputs of the Ethernet safety interface can be used as safe inputs as long as they are configured in WorkVisual.</p> <p>(>>> 13.6 "Safety interfaces" Page 245)</p>
<i>Panic switch 1 used</i>	Indicates whether the panic switch is connected to a safe input
<i>Panic switch 2 used</i>	
<i>Panic switch 3 used</i>	<ul style="list-style-type: none"> • true: an input is connected • false: no input is connected <p>Default: false</p>

Parameter	Description
Panic switch 1 input signal	Safe input to which the panic switch is connected
Panic switch 2 input signal	The inputs of the discrete safety interface or the safe inputs of the Ethernet safety interface can be used as safe inputs as long as they are configured in WorkVisual.
Panic switch 3 input signal	(>>> 13.6 "Safety interfaces" Page 245)

Example

Manual guidance with signal (category: *Enabling device*)

A robot equipped with a hand guiding device is to be manually guided in a defined area in order to teach the points on a path. The area for manual guidance is defined by a Cartesian protected space. In this protected space, a Cartesian velocity of 250 mm/s must only be exceeded if the enabling signal is issued via the hand guiding device. If no enabling signal is issued, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
<i>Hand guiding device enabling inactive</i>	<i>Cartesian protected space monitoring</i>	<i>Cartesian velocity monitoring</i>	<i>Stop 1 (path-maintaining)</i>

13.14.5.2 Monitoring functions during manual guidance

Description

The standard AMF *Hand guiding device enabling active* makes it possible to implement safety functions that activate other monitoring functions during manual guidance with the enabling device, e.g. Cartesian velocity monitoring.

AMF	Description
<i>Hand guiding device enabling active</i>	<p>This AMF is violated if the enabling signal for manual guidance is issued.</p> <p>The AMF <i>Hand guiding device enabling active</i> represents the inverse state of the AMF <i>Hand guiding device enabling inactive</i>:</p> <ul style="list-style-type: none"> • The AMF <i>Hand guiding device enabling active</i> is violated if the AMF <i>Hand guiding device enabling inactive</i> is not violated. • The AMF <i>Hand guiding device enabling active</i> is not violated as long as the AMF <i>Hand guiding device enabling inactive</i> is violated. <p>The AMF <i>Hand guiding device enabling active</i> takes into account the enabling device configured for the AMF <i>Hand guiding device enabling inactive</i>.</p>

Example

Space and velocity monitoring during manual guidance with enabling device (category: *Workspace monitoring, Velocity monitoring*)

During manual guidance of a robot with an enabling device, the robot must not leave a defined workspace. Furthermore, the robot is to move with a maximum velocity during manual guidance of 600 mm/s. If the workspace is left while enabling is active, or if the velocity limit is exceeded, a safety stop 1 (path-maintaining) is to be executed.

AMF1	AMF2	AMF3	Reaction
<i>Hand guiding device enabling active</i>	<i>Cartesian workspace monitoring</i>	-	<i>Stop 1 (path-maintaining)</i>
<i>Hand guiding device enabling active</i>	<i>Cartesian velocity monitoring</i>	-	<i>Stop 1 (path-maintaining)</i>

13.14.5.3 Velocity monitoring during manual guidance

For manual guidance of the robot, a maximum permissible velocity must be defined that may not be exceeded during manual guidance. The value for this velocity must be defined in a risk assessment.

A safety function for safe velocity monitoring during manual guidance is configured in row 3 of the *KUKA PSM* table. The safety function takes into account the enabling device configured for the *Hand guiding device enabling inactive* AMF. Once the enabling signal has been issued, a safety stop 1 (path-maintaining) is carried out if the velocity limit is exceeded.

The instance of the *Cartesian velocity monitoring* AMF that is used has the following preset parameter values:

- *Monitored kinematic system: First kinematic system*
- *Monitored structure: Robot and tool*
- *Maximum velocity: 250 mm/s*

The parameter values can be modified. (>>> [13.14.8.2 "Defining Cartesian velocity monitoring" Page 283](#))

13.14.6 Evaluating the position referencing

Description

Position referencing checks whether the saved zero position of the motor of an axis (= saved mastering position) corresponds to the actual mechanical zero position.

The safety integrity of the safety functions based upon this is limited until the position referencing test has been performed. This includes, for example, safely monitored Cartesian and axis-specific robot positions, safely monitored Cartesian velocities, TCP force monitoring and collision detection.

The AMF *Position referencing* can be used to check whether the position values of all axes are referenced.

AMF	Description
<i>Position referencing</i>	The AMF is violated in the following cases: <ul style="list-style-type: none"> • The position of at least one axis of the monitored kinematic system is not referenced. • The position referencing of at least one axis has failed.

Parameter	Description
<i>Monitored kinematic system</i>	Kinematic system to be monitored <ul style="list-style-type: none"> • <i>First kinematic system: Robot</i> • <i>Second kinematic system: No function</i> • <i>Third kinematic system: No function</i> • <i>Fourth kinematic system: No function</i>

Example

Monitoring the position referencing status (category: *Safety stop*)

A robot with non-referenced axes may only be moved at a reduced velocity of maximum 250 mm/s. The reduced velocity is intended to prevent hazards arising as a result of position referencing not having been performed or having failed.

To ensure this, the referencing status of all axes is monitored in the operating modes with high velocity (T2 and AUT). As soon as the position of at least one axis is not successfully referenced, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
<i>High-velocity mode</i>	-	<i>Position referencing</i>	<i>Stop 1 (path-main-taining)</i>

13.14.7 Evaluating the torque referencing

Description

The referencing test of the joint torque sensors checks whether the expected external torque, which can be calculated for an axis based on the robot model and the given load data, corresponds to the value determined on the basis of the measured value of the joint torque sensor. If the difference between these values exceeds a certain tolerance value, the referencing of the torque sensors has failed.

The safety integrity of the safety functions based upon this is limited until the torque referencing test has been performed successfully. This includes, for example, axis torque and TCP force monitoring as well as collision detection.

The AMF *Torque referencing* can be used to check whether the joint torque sensors of all axes are referenced.

AMF	Description
<i>Torque referencing</i>	The AMF is violated in the following cases: <ul style="list-style-type: none"> The joint torque sensor of at least one axis of the monitored kinematic system is not referenced. The referencing of at least one joint torque sensor has failed.

Parameter	Description
<i>Monitored kinematic system</i>	Kinematic system to be monitored <ul style="list-style-type: none"> <i>First kinematic system:</i> Robot <i>Second kinematic system:</i> No function <i>Third kinematic system:</i> No function <i>Fourth kinematic system:</i> No function

Example

Monitoring the torque referencing status (category: *Safety stop*)

A safe collision detection is configured for a station. If a torque of more than 20 Nm is detected in at least one axis of the robot, a safety stop 0 is triggered. Since the safety integrity of this function is only ensured for successfully referenced joint torque sensors, the referencing status of the sensors must be monitored simultaneously. As soon as at least one joint torque sensor has not been referenced or referencing has failed, a safety

stop 1 (path-maintaining) is to be triggered in high-velocity operating modes (T2 and AUT).

AMF1	AMF2	AMF3	Reaction
<i>Collision detection</i>	-	-	<i>Stop 0</i>
<i>High-velocity mode</i>	-	<i>Torque referencing</i>	<i>Stop 1 (path-maintaining)</i>

13.14.8 Velocity monitoring functions

A moving kinematic system always presents a danger to persons in its vicinity. In order to protect persons, it may be necessary to impose a defined maximum velocity, for example to give persons time to move out of the way of the robot. This means that the velocity must be monitored continuously.

Velocity monitoring functions are available for the robot and for mobile platforms. The axis velocities and the Cartesian velocity of a kinematic system can be monitored.

13.14.8.1 Defining axis-specific velocity monitoring

The AMF *Axis velocity monitoring* is used to define an axis-specific velocity monitoring function.

AMF	Description
<i>Axis velocity monitoring</i>	The AMF is violated if the absolute velocity of the monitored axis of the monitored kinematic system exceeds the configured limit.
Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: Mobile platform • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Monitored axis</i>	<p>Axis of the kinematic system to be monitored</p> <ul style="list-style-type: none"> • Axis1 ... Axis16 <p>Axis1 ... Axis7 are used for an LBR.</p> <p>In the case of a mobile platform, the axes are assigned as follows:</p> <ul style="list-style-type: none"> • Axis1: front left drive • Axis2: front right drive • Axis3: rear left drive • Axis4: rear right drive
<i>Maximum velocity [°/s]</i>	<p>Maximum permissible velocity at which the monitored axis may move in the positive and negative direction of rotation</p> <ul style="list-style-type: none"> • 1 ... 500 °/s

13.14.8.2 Defining Cartesian velocity monitoring

Description

The *Cartesian velocity monitoring* AMF is used to define a Cartesian velocity monitoring function.

- In the case of a robot, the translational Cartesian velocity can be monitored at all axis center points as well as at the robot flange. If a safety-oriented tool is active on the robot controller, the system can additionally or alternatively monitor the velocity at the center points of the spheres which are used to configure the safety-oriented tool.

(>>> [9.3.10 "Configuring a safety-oriented tool" Page 176](#))



The system does not monitor the entire structure of the robot and tool against the violation of a velocity limit, but only defined points of the robot structure and the center points of the monitoring spheres of the tools. In particular with protruding tools and workpieces, the monitoring spheres of the safety-oriented tool must be planned and configured in such a way as to assure the safety integrity of the velocity monitoring.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the Cartesian velocity of the monitoring spheres relative to the carrier kinematic system is monitored and not the absolute velocity of the monitoring spheres in space.

- In the case of a mobile platform, the translational Cartesian velocity can be monitored at 4 defined points near the corner points of the platform. If a safety-oriented tool is active on the robot controller, the system can additionally or alternatively monitor the velocity at the center points of the spheres which are used to configure the safety-oriented tool.
- Only velocity components within the plane of the platform are taken into consideration.

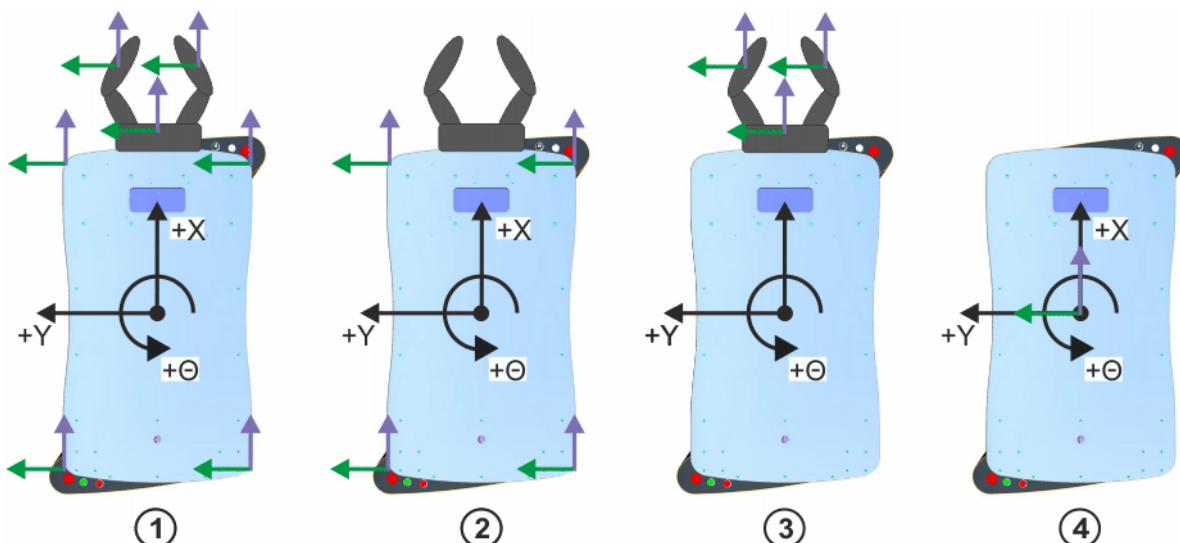


Fig. 13-10: Velocity monitoring for platforms (monitored structure)

- 1 Monitored structure: Robot and tool
- 2 Monitored structure: Robot
- 3 Monitored structure: Tool
- 4 Monitored structure: Tool (no safety-oriented tool active)

AMF	Description
<i>Cartesian velocity monitoring</i>	<p>The AMF is violated if the Cartesian translational velocity at at least one point of the monitored kinematic system exceeds the defined limit.</p> <p>The AMF is additionally violated in the following cases:</p> <ul style="list-style-type: none"> • An axis is not mastered. • The referencing of a mastered axis has failed. <p>Note: If an AMF is violated due to loss of mastering, the robot can only be moved and mastered again by switching to CRR mode.</p>
Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: Mobile platform • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Monitored structure</i>	<p>Structure to be monitored</p> <p>Kinematic system to be monitored is a robot</p> <ul style="list-style-type: none"> • <i>Robot and tool</i>: The center points of the axes on the robot and the center points of the spheres used to configure the active safety-oriented tool are monitored (default). • <i>Robot</i>: The center points of the axes on the robot are monitored. • <i>Tool</i>: The center points of the spheres used to configure the active safety-oriented tool are monitored. <p>Note: If no safety-oriented tool is active and the tool is selected as the structure to be monitored, the center point of the robot flange is monitored.</p> <p>(>>> <i>"Spheres on the robot" Page 289</i>)</p>
	<p>Kinematic system to be monitored is a mobile platform</p> <ul style="list-style-type: none"> • <i>Robot and tool</i>: The 4 corner points of the platform and the center points of the spheres used to configure the active safety-oriented tool are monitored (default). • <i>Robot</i>: The 4 corner points of the platform are monitored. • <i>Tool</i>: The center points of the spheres used to configure the active safety-oriented tool are monitored. <p>Note: If no safety-oriented tool is active and the tool is selected as the structure to be monitored, the frame at the center point of the platform is monitored.</p>
<i>Maximum velocity [mm/s]</i>	<p>Maximum permissible Cartesian velocity which must not be exceeded at any monitored point</p> <ul style="list-style-type: none"> • 1 ... 10000 mm/s

Example

Category: *Velocity monitoring*

If a Cartesian workspace is violated, the Cartesian velocity of the robot must not exceed 300 mm/s. If this velocity is exceeded, a safety stop 1 is triggered.

AMF1	AMF2	AMF3	Reaction
<i>Cartesian workspace monitoring</i>	-	<i>Cartesian velocity monitoring</i>	<i>Stop 1</i>

13.14.8.3 Direction-specific monitoring of Cartesian velocity

Description

The *Tool-related velocity component* AMF is used to check whether the Cartesian translational velocity in a specific direction is below the configurable limit value.

The AMF can be used, for example, to ensure that the velocity in the working direction of a sharp-pointed tool is not too high. The AMF can also be used to monitor the motion direction.

The AMF monitors the translational Cartesian velocity at up to 6 points of the last active safety-oriented tool of the kinematic chain. These monitoring points are safety-oriented frames of the safety-oriented tool and can be defined as such in the frame properties:

- **Point for tool-related velocity**
 - If a safety-oriented frame is defined as a point for the tool-related velocity, the origin of this frame is used as the monitored point.
 - If no safety-oriented frame is defined as a point for the tool-related velocity, the origin of the pickup frame of the active safety-oriented tool is used as the monitored point:
 - In the case of a fixed safety-oriented tool, the pickup frame is the flange coordinate system. The velocity is monitored at the origin of the flange coordinate system.
 - In the case of a safety-oriented tool that is coupled to the fixed tool, the pickup frame is the standard frame for motions of the fixed tool. The velocity is monitored at the origin of the standard frame for motions.

Additionally, a safety-oriented frame defining the orientation for the tool-related velocity at the monitored points can be freely selected in the properties of the safety-oriented tool:

- **Orientation for tool-related velocity**
 - If no safety-oriented frame is defined as an orientation frame for the tool-related velocity, the pickup frame of the active safety-oriented tool is used as the orientation for the tool-related velocity:
 - In the case of a fixed safety-oriented tool, the pickup frame is the flange coordinate system. The orientation of the flange coordinate system determines the monitoring direction.
 - In the case of a safety-oriented tool that is coupled to the fixed tool, the pickup frame is the standard frame for motions of the fixed tool. The orientation of the standard frame for motions determines the monitoring direction.

(>>> [9.3.10 "Configuring a safety-oriented tool" Page 176](#))

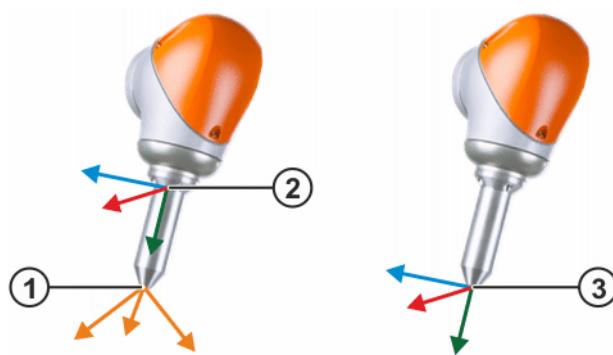


Fig. 13-11: Orientation at the point for tool-related velocity

- 1 Point for the tool-related velocity
- 2 Orientation for the tool-related velocity
- 3 Orientation at the point for the tool-related velocity (combination of 1 and 2)

If the monitored kinematic system is a mobile platform, it is assumed that the safety-oriented tool is fastened at the center point of the platform.

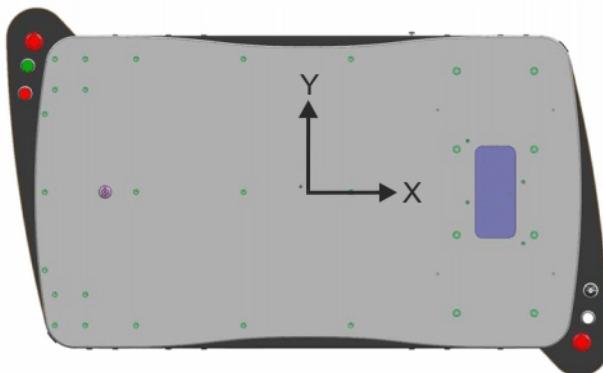


Fig. 13-12: Point for tool-related velocity

If no safety-oriented tool is active, the following frame is used as the point and orientation for the tool-related velocity, depending on the monitored kinematic system:

- For a robot: frame at the center point of the robot flange
- For a mobile platform: frame at the center point of the platform

The component of the velocity vectors of the configured points in a specific direction of the configured orientation frame is monitored. During configuration of the AMF, this direction is specified as a component of the velocity vectors in the coordinate system of the orientation frame. Any of the total of 6 components of the coordinate system, i.e. the X, Y and Z components, each in the positive and negative direction, can be selected.

Furthermore, the maximum velocity that the selected component of the velocity vector must not exceed is also defined.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the velocity of the monitored point relative to the carrier kinematic system is monitored and not the absolute velocity in space.

AMF	Description
<i>Tool-related velocity component</i>	<p>The AMF is violated if the configured component of the velocity vector in the coordinate system of the orientation frame of the monitored kinematic system exceeds the maximum defined value at one or more of the monitored points.</p> <p>In the case of an LBR, the AMF is additionally violated in the following cases:</p> <ul style="list-style-type: none"> • An axis is not mastered. • The referencing of a mastered axis has failed.
Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: Mobile platform • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Maximum velocity [mm/s]</i>	<p>Maximum Cartesian velocity for the monitored component of the velocity vector</p> <ul style="list-style-type: none"> • 1 ... 10000 mm/s <p>Note: When selecting the maximum velocity, it must be noted that, particularly in the case of highly dynamic motions, low velocities against the commanded direction of motion may occur due to overshoot. For this reason, it is recommended that the maximum velocity should not be set too low.</p>
<i>Component of the velocity vector</i>	<p>Monitored component of the velocity vector (direction of monitoring)</p> <ul style="list-style-type: none"> • <i>X positive or X negative</i> • <i>Y positive or Y negative</i> • <i>Z positive or Z negative</i>

Example

A sharp-pointed tool may be moved in its working direction at a maximum of 25 mm/s. In order to be able to monitor the velocity in the working direction, a frame is created at the tool tip and the tool and the frame are configured as safety-oriented. The safety-oriented frame at the tool tip is defined in the frame properties as a point for the tool-related velocity and in the tool properties as an orientation frame.

(>>> [9.3.10 "Configuring a safety-oriented tool" Page 176](#))

An instance of the *Tool-related velocity component* AMF is configured in such a way that the positive Z component of the velocity vector in the coordinate system of the tool tip may not exceed a value of 25 mm/s. For this, the following parameters are set for the instance used:

- *Monitored kinematic system*: First kinematic system
- *Maximum velocity [mm/s]*: 25
- *Component of the velocity vector*: Z positive

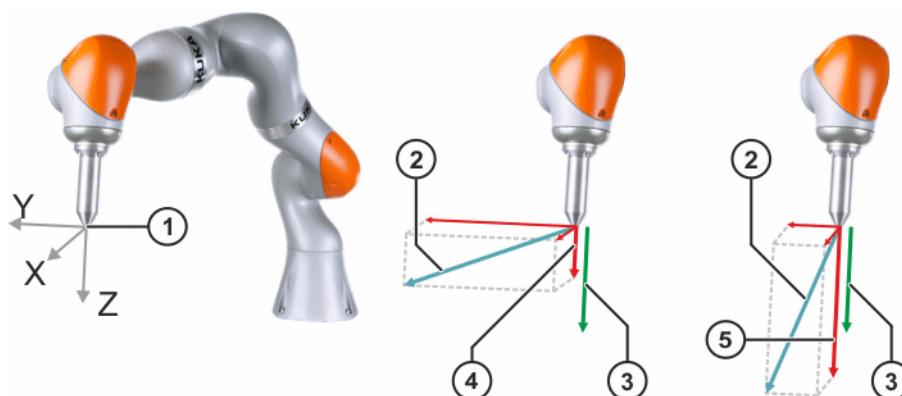


Fig. 13-13: Velocity monitoring in the tool direction

Item	Description
1	Reference frame for the tool-related velocity component
2	Velocity vector of the translational Cartesian velocity
3	Maximum permissible velocity for the positive Z component of the velocity vector
4	Positive Z component of the velocity vector The velocity is below the maximum permissible velocity; the AMF is not violated.
5	Positive Z component of the velocity vector The velocity is above the maximum permissible velocity; the AMF is violated.

The safety function configured with the AMF monitors the positive Z component of the velocity vector. If the maximum velocity of 25 mm/s is exceeded by the monitored component in Automatic mode, a safety stop 1 (path-maintaining) is to be executed.

Category: *Velocity monitoring*

AMF1	AMF2	AMF3	Reaction
Automatic mode	Tool-related velocity component (1) First kinematic system	-	Stop 1 (path-maintaining)

13.14.9 Monitoring spaces

Description

The robot environment can be divided into areas in which it must remain for execution of the application, and areas which it must not enter or may only enter under certain conditions. The system must then continuously monitor whether the robot is within or outside of such a monitoring space. A monitoring space can be defined as a Cartesian cuboid or by means of individual axis ranges.

A Cartesian monitoring space can be configured as a workspace in which the robot must remain, or as a protected space which it must not enter.

Via the link to other safety monitoring functions, it is possible to define further conditions which must be met when a monitoring space is violated. For example, a monitoring space can be activated by a safe input or applicable in Automatic mode only.



If the robot has violated a monitoring space and been stopped by the safety controller, the robot can be moved out of the violated area in CRR mode.
 (>>> [6.9 "CRR mode – controlled robot retraction" Page 96](#))

Spheres on the robot

Spheres are modeled around selected points on the robot, enclosing and moving with the robot. These spheres are predefined and are monitored, as standard, against the limits of activated Cartesian monitoring spaces.

The centers and radii of the monitored spheres are defined in the machine data of the robot. A sphere is defined for each robot axis, for the robot base and for the robot flange.

The dimensions of the monitored spheres vary according to robot type and the media flange used:

- r = sphere radius
- z, y = sphere center point relative to the robot base coordinate system

Variant 1: LBR Med 7 R800 with media flange Inside electrical Med

	Base	A1	A2	A3	A4	A5	A6	A7	Flange
r [mm]	135	90	125	90	125	90	80	85	65
z [mm]	50	90	340	538	740	935	1140	1130	1220
y [mm]							-30		

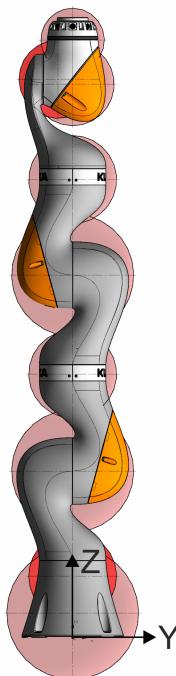


Fig. 13-14: Example: Variant 1

Variant 2: LBR Med 14 R820 with media flange Inside electrical Med

	Base	A1	A2	A3	A4	A5	A6	A7	Flange
r [mm]	150	100	140	90	131	90	80	85	65
z [mm]	50	160	360	580	780	980	1180	1170	1260
y [mm]							-30		

Spheres on tool

If a safety-oriented tool is active on the robot controller, the system not only monitors the spheres on the robot as standard, but also the spheres used to configure the safety-oriented tool.

(>>> [9.3.10 "Configuring a safety-oriented tool" Page 176](#))



The system does not monitor the entire structure of the robot and tool against the violation of a space, but rather only the monitoring spheres. In particular with protruding tools and workpieces, the monitoring spheres of the safety-oriented tool must be planned and configured in such a way as to assure the safety integrity of workspaces and protected spaces.

Selecting monitoring spheres

It is not necessary or appropriate to include all robot and tool spheres in the Cartesian workspace monitoring of every application.

Example: If the entry of a tool into a protected space is programmed to activate further monitoring functions, only the tool spheres must be monitored.

The structure to be monitored can be selected when configuring Cartesian monitoring spaces:

- Robot and tool (default)
- Only tool
- Only robot

Stopping distance

If the robot is stopped by a monitoring function, it requires a certain stopping distance before coming to a standstill.

The stopping distance depends primarily on the following factors:

- Robot type
- Velocity of the robot
- Position of the robot axes
- Payload



Including stopping distances in the risk assessment

The stopping distance when a safety function is triggered varies according to the specific robot type. Failure to take this into consideration when parameterizing the safety functions may result in death, severe injuries or damage to property.

- The system integrator must include the stopping distances in the risk assessment and parameterize the safety functions accordingly.



Further information about the stopping distances and stopping times can be found in the Instructions for Use of the relevant robot.

13.14.9.1 Defining Cartesian workspaces

Description

A Cartesian workspace is defined as a cuboid whose position and orientation in space are defined relative to the world coordinate system.

These monitoring spheres are monitored against the limits of activated Cartesian workspaces and must move within these workspaces.

The AMF *Cartesian workspace monitoring* is used to define a Cartesian workspace. The AMF is violated as soon as one of the monitored spheres is no longer completely within the defined workspace.

The AMF is additionally violated in the following cases:

- An axis is not mastered.
- The referencing of a mastered axis has failed.



If the monitored kinematic system is mounted on a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the position and orientation of the monitoring space are relative to the world coordinate system and are thus defined relative to the alignment of the base of the monitored kinematic system. For this reason, the monitoring space is also moved in the event of a change in position or inclination of the carrier kinematic system.

Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Monitored structure</i>	<p>Structure to be monitored</p> <ul style="list-style-type: none"> • <i>Robot and tool</i>: The spheres on the robot and the spheres used to configure the safety-oriented tool are monitored. (Default) • <i>Robot</i>: The spheres on the robot are monitored. • <i>Tool</i>: The spheres used to configure the safety-oriented tool are monitored. <p>Note: If no safety-oriented tool is configured and the tool is selected as the structure to be monitored, the sphere on the robot flange is monitored. (>>> <i>"Spheres on the robot"</i> Page 289)</p>

One corner of the cuboid is defined relative to the world coordinate system. This is the origin of the workspace and is defined by the following parameters:

Parameter	Description
<i>X, Y, Z [mm]</i>	<p>Offset of the origin of the workspace along the X, Y and Z axes of the world coordinate system</p> <ul style="list-style-type: none"> • -100,000 mm ... +100,000 mm
<i>A, B, C [°]</i>	<p>Orientation of the origin of the workspace about the axes of the world coordinate system, specified by the rotational angles A, B, C</p> <ul style="list-style-type: none"> • 0° ... 359°

Based on this defined origin, the size of the workspace is determined along the coordinate axes:

Parameter	Description
<i>Length [mm]</i>	Length of the workspace (= distance along the positive X axis of the origin) <ul style="list-style-type: none"> • 0 mm ... 100,000 mm
<i>Width [mm]</i>	Width of the workspace (= distance along the positive Y axis of the origin) <ul style="list-style-type: none"> • 0 mm ... 100,000 mm
<i>Height [mm]</i>	Height of the workspace (= distance along the positive Z axis of the origin) <ul style="list-style-type: none"> • 0 mm ... 100,000 mm



The violation of a Cartesian workspace is only rectified when all monitored spheres have returned to within the workspace limits with a minimum distance of 1 mm to these limits.

Example

The diagram shows an example of a Cartesian workspace. Its origin is offset in the negative X and Y directions with reference to the world coordinate system.

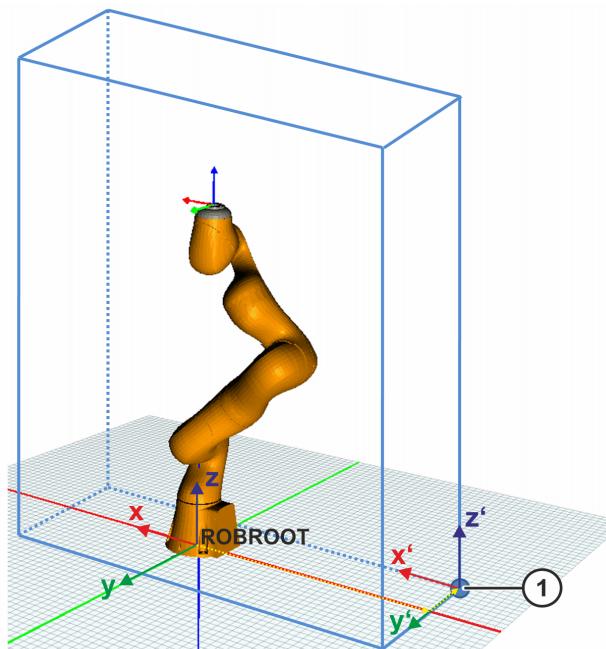


Fig. 13-15: Example of a Cartesian workspace

1 Origin of the workspace

13.14.9.2 Defining Cartesian protected spaces

Description

A Cartesian protected space is defined as a cuboid whose position and orientation in space are defined relative to the world coordinate system.

These monitoring spheres are monitored against the limits of activated protected spaces and must move outside of these protected spaces.

The AMF *Cartesian protected space monitoring* is used to define a Cartesian protected space. The AMF is violated as soon as one of the moni-

tored spheres is no longer completely outside of the defined protected space.

The AMF is additionally violated in the following cases:

- An axis is not mastered.
- The referencing of a mastered axis has failed.

If a very narrow protected space is configured, the robot may be able to move into and out of the protected space without the space violation being detected. Possible cause: Due to a very high tool velocity, the protected space is only violated during a very short interval.

Assuming that the following minimum values are configured:

- Radius of tool sphere: 25 mm
- Thickness of protected space: 0 mm

In this case, tool velocities of over 4 m/s are required for the robot to pass through the protected space without detection.

The following measures are recommended in order to prevent robots from passing through protected spaces undetected:

- Configure Cartesian velocity monitoring (do not allow a value greater than 4 m/s).
- OR: When configuring the protected space, select sufficient values for the length, width and height of the protected space.
- OR: When configuring the tool spheres, select sufficient values for the radius.



If the monitored kinematic system is mounted on a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the position and orientation of the monitoring space are relative to the world coordinate system and are thus defined relative to the alignment of the base of the monitored kinematic system. For this reason, the monitoring space is also moved in the event of a change in position or inclination of the carrier kinematic system.

Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Monitored structure</i>	<p>Structure to be monitored</p> <ul style="list-style-type: none"> • <i>Robot and tool</i>: The spheres on the robot and the spheres used to configure the safety-oriented tool are monitored. (Default) • <i>Robot</i>: The spheres on the robot are monitored. • <i>Tool</i>: The spheres used to configure the safety-oriented tool are monitored. <p>Note: If no safety-oriented tool is configured and the tool is selected as the structure to be monitored, the sphere on the robot flange is monitored. (>>> <i>"Spheres on the robot"</i> Page 289)</p>

One corner of the cuboid is defined relative to the world coordinate system. This is the origin of the protected space and is defined by the following parameters:

Parameter	Description
X, Y, Z [mm]	Offset of the origin of the protected space along the X, Y and Z axes of the world coordinate system • -100,000 mm ... +100,000 mm
A, B, C [°]	Orientation of the origin of the protected space about the axes of the world coordinate system, specified by the rotational angles A, B, C • 0° ... 359°

Based on this defined origin, the size of the protected space is determined along the coordinate axes:

Parameter	Description
Length [mm]	Length of the protected space (= distance along the positive X axis of the origin) • 0 mm ... 100,000 mm
Width [mm]	Width of the protected space (= distance along the positive Y axis of the origin) • 0 mm ... 100,000 mm
Height [mm]	Height of the protected space (= distance along the positive Z axis of the origin) • 0 mm ... 100,000 mm



The violation of a Cartesian protected space is only rectified when all monitored spheres have returned to outside the protected space limits with a minimum distance of 1 mm to these limits.

Example

The diagram shows an example of a Cartesian protected space. Its origin is offset in the negative X and positive Y directions with reference to the world coordinate system.

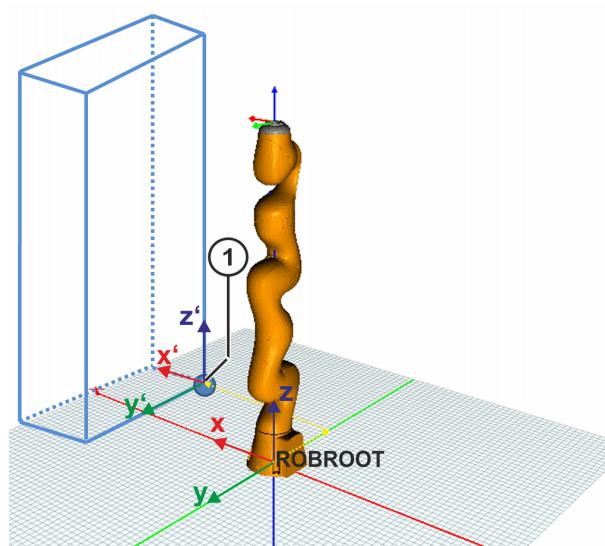


Fig. 13-16: Example of a Cartesian protected space

1 Origin of the protected space

13.14.9.3 Defining axis-specific monitoring spaces

Description

The axis limits can be defined individually and safely monitored for each axis. The axis angle must lie within the defined axis range.

The AMF *Axis range monitoring* is used to define an axis-specific monitoring space. The AMF is violated if an axis is not inside the defined axis range.

The AMF is additionally violated in the following cases:

- An axis is not mastered.
- The referencing of a mastered axis has failed.

Parameter	Description
<i>Monitored kinematic system</i>	Kinematic system to be monitored <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Monitored axis</i>	Axis to be monitored <ul style="list-style-type: none"> • Axis1 ... Axis16 Note: Axis1 ... Axis7 are used for an LBR.
<i>Lower limit [°]</i>	Lower limit of the allowed axis range in which the monitored axis may move <ul style="list-style-type: none"> • -180° ... +180°
<i>Upper limit [°]</i>	Upper limit of the allowed axis range in which the monitored axis may move <ul style="list-style-type: none"> • -180° ... +180°



The permissible axis range runs in the positive direction of rotation of the axis from the lower to the upper limit.
If the axis position at $\pm 180^\circ$ lies within the permissible angle range, the lower limit must be greater than the upper limit.

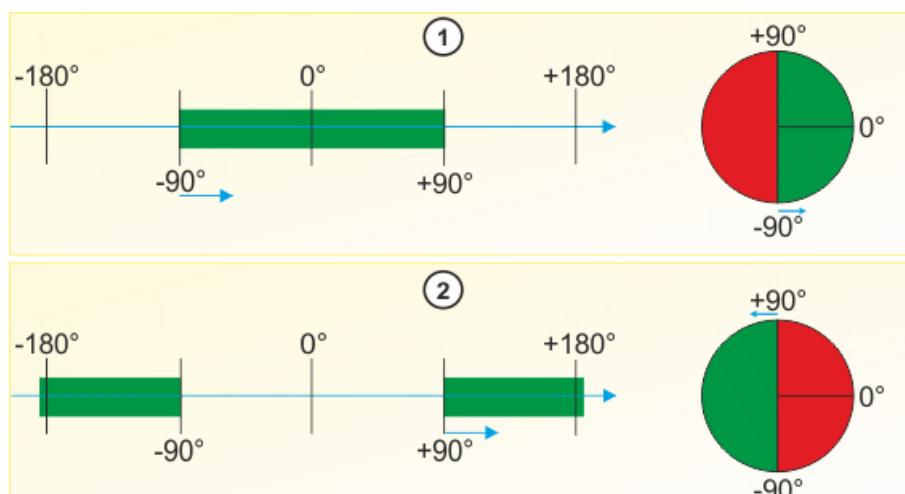


Fig. 13-17: Examples of axis-specific workspaces

- 1 Lower limit: -90°; upper limit: +90°
- 2 Lower limit: +90°; upper limit: -90°



For personnel protection, only the position of the axis is relevant. For this reason, the positions are converted to the axis range -180° ... +180°, even for axes which can rotate more than 360°.

Example

Axes A1, A2 and A4 are to be monitored so that the robot may only be moved in a limited space. The monitoring is activated by a safe input. The permitted range of each axis is defined by an upper and lower limit, and is shown in green in the corresponding chart in the PSM table.

As soon as one of the monitored axis ranges is violated, a safety stop 1 (path-maintaining) is triggered. For this purpose, an individual table row must be used for each axis.

Configurable customer safety configuration						(11/100)
Row	Active	Category	AMF 1	AMF 2	AMF 3	Reaction
8	<input checked="" type="checkbox"/>	Workspace monitoring	Input signal (4) Input for safety signal : Input CIB_SR4	Axis range monitoring (1) Monitored axis : Axis 1		Stop 1 (path-maintaining)
9	<input checked="" type="checkbox"/>	Workspace monitoring	Input signal (4) Input for safety signal : Input CIB_SR4	Axis range monitoring (2) Monitored axis : Axis 2		Stop 1 (path-maintaining)
10	<input checked="" type="checkbox"/>	Workspace monitoring	Input signal (4) Input for safety signal : Input CIB_SR4	Axis range monitoring (3) Monitored axis : Axis 4		Stop 1 (path-maintaining)

Fig. 13-18: PSM table – simultaneous monitoring of 3 axes

13.14.10 Monitoring the tool orientation

The AMF *Tool orientation* can be used to monitor the orientation of a safety-oriented tool. It checks whether a specific axis of the tool orientation frame is within a permissible direction range.

This function can for example be used to prevent dangerous parts of the mounted tool, e.g. sharp edges, from pointing towards humans in HRC applications.

The following tool orientations are monitored, depending on the tool configuration:

- As standard, the orientation of the Z axis of the tool orientation frame of the last active safety-oriented tool of the kinematic chain is monitored.
(>>> [9.3.10 "Configuring a safety-oriented tool" Page 176](#))
- If no tool orientation frame is defined, the Z axis of the pickup frame of the last active safety-oriented tool of the kinematic chain is monitored.
 - If a fixed safety-oriented tool is active, this corresponds to the Z axis of the flange coordinate system.
 - If a safety-oriented tool is active and coupled to the fixed tool, this corresponds to the Z axis of the pickup frame of the fixed tool.
- If no safety-oriented tool is active, the Z axis of the flange coordinate system is monitored.

The permissible range for the orientation angle is defined by a reference vector with a fixed orientation relative to the world coordinate system and a permissible deviation angle from this reference vector.

The reference vector is defined by the rotation of the unit vector of the Z axis of the world coordinate system about the 3 Euler angles A, B and C relative to the world coordinate system. A monitoring cone is extended

around the reference vector. The opening of the cone is defined by a configurable deviation angle. The deviation angle defines the permissible angle between the tool orientation and reference vector. The values of the angle of the reference vector and the deviation angle are defined in the parameterization of the AMF.

The monitoring sphere defines the permissible range for the tool orientation.

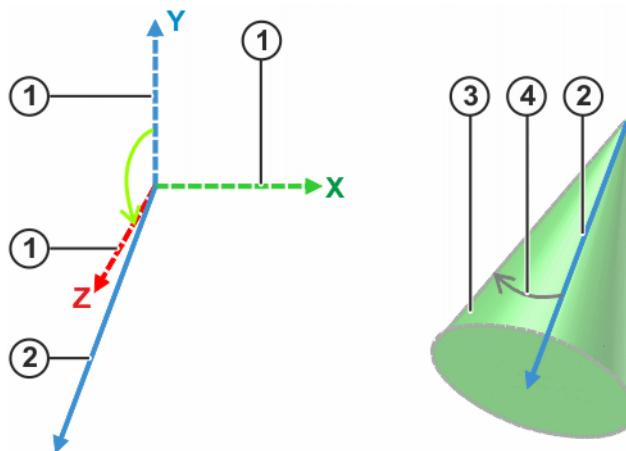


Fig. 13-19: Monitoring cone for tool orientation

Item	Description
1	Axes of the world coordinate system
2	Reference vector The reference vector defines a fixed orientation relative to the world coordinate system.
3	Monitoring cone Defines the permissible range for the tool orientation.
4	Deviation angle The deviation angle determines the opening of the monitoring cone.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform), it must be taken into consideration that the orientation of the reference vector is relative to the world coordinate system. This means that the reference orientation is defined relative to the alignment of the base of the monitored kinematic system. For this reason, the reference orientation is also moved in the event of a change in inclination of the carrier kinematic system (e.g. due to driving up a ramp).

AMF	Description
<i>Tool orientation</i>	<p>The AMF is violated if the angle between the reference vector and Z axis of the tool orientation frame is greater than the configured deviation angle.</p> <p>The AMF is additionally violated in the following cases:</p> <ul style="list-style-type: none"> • An axis is not mastered. • The position referencing of a mastered axis has failed.

Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> <i>First kinematic system:</i> Robot <i>Second kinematic system:</i> No function <i>Third kinematic system:</i> No function <i>Fourth kinematic system:</i> No function
<i>A [°]</i>	<p>Rotation of the reference vector about the Z axis of the world coordinate system</p> <ul style="list-style-type: none"> 0° ... 359°
<i>B [°]</i>	<p>Rotation of the reference vector about the Y axis of the world coordinate system</p> <ul style="list-style-type: none"> 0° ... 359°
<i>C [°]</i>	<p>Rotation of the reference vector about the X axis of the world coordinate system</p> <ul style="list-style-type: none"> 0° ... 359°
<i>Operating angle [°]</i>	<p>Workspace of the tool orientation</p> <p>Defines the maximum permissible deviation angle between the reference vector and the Z axis of the tool orientation frame.</p> <ul style="list-style-type: none"> 1° ... 179°

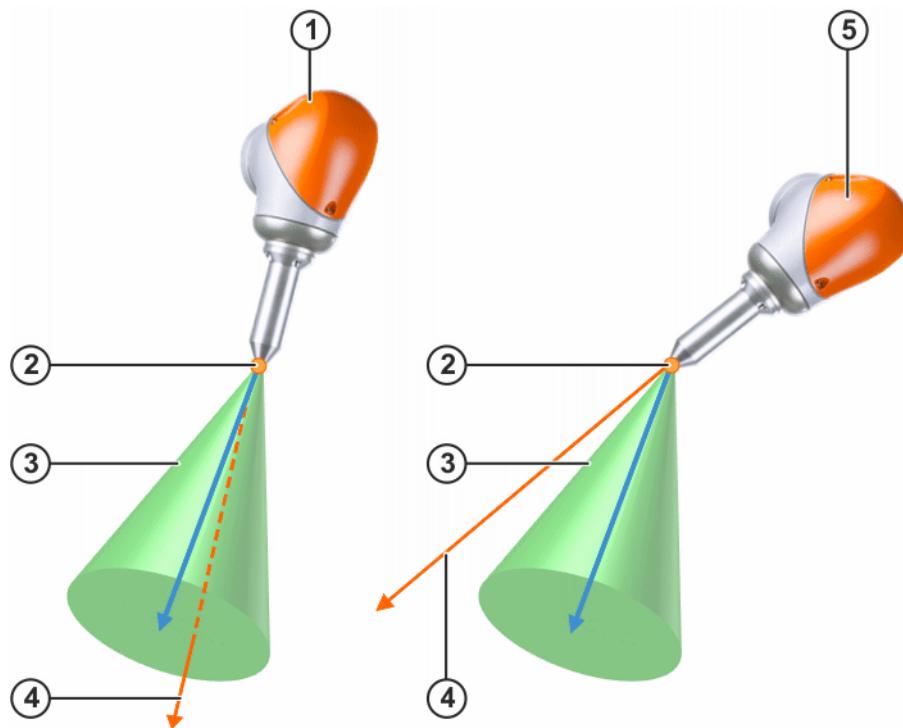


Fig. 13-20: Tool orientation (not violated and violated)

Item	Description
1	<p>Robot is not violating the <i>Tool orientation AMF</i>.</p> <p>The Z axis of the tool orientation frame is within the range defined by the monitoring cone.</p>
2	Origin of the tool orientation frame

Item	Description
3	Monitoring cone
4	Z axis of the tool orientation frame
5	Robot is violating the <i>Tool orientation</i> AMF. The Z axis of the tool orientation frame is outside of the range defined by the monitoring cone.

13.14.11 Standstill monitoring (safe operational stop)

Description

If, under certain conditions, the robot must not move but must remain under servo-control, the standstill of all axes must be safely monitored. The AMF *Standstill monitoring of all axes* is used for this purpose.

This AMF is an extended AMF, meaning that the monitoring only begins when all other AMFs of the safety function are violated.



Extended AMFs are not available for the safety functions of the ESM mechanism.

Standstill is defined as retaining the axis positions. At the start of standstill monitoring, the axis positions are saved and compared to the current joint values for as long as the monitoring is active.

Since standstill monitoring is monitored in a narrow tolerance range, monitoring can also be violated if the motion of the robot is caused by an outside force, e.g. if the robot is jolted.

AMF	Description
<i>Standstill monitoring of all axes</i>	The AMF is violated as soon as the joint value of an axis is outside of a tolerance range of +/- 0.1° of the value saved when standstill monitoring was activated, or if one of the axes moves at an absolute value of more than 1 °/s.
Parameter	Description
<i>Monitored kinematic system</i>	Kinematic system to be monitored <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function

Example

Category: *Safe operational stop*

If the robot is situated outside of its workspace, it must be assured that the robot is no longer moving as soon as persons are in its vicinity. The workspace is configured by means of a Cartesian workspace. There is a sensor connected to a safe input which detects persons at risk. If both the workspace and the input signal are violated, the standstill monitoring is activated.

AMF1	AMF2	AMF3	Reaction
<i>Input signal</i>	<i>Cartesian workspace monitoring</i>	<i>Standstill monitoring of all axes</i>	<i>Stop 1</i>

13.14.12 Activation delay for safety function

Description

The AMF *Time delay* can be used to delay the triggering of the reaction of a safety function for a defined time.

This AMF is an extended AMF, meaning that the delay time only starts running when all other AMFs of the safety function are violated.



Extended AMFs are not available for the safety functions of the ESM mechanism.

AMF	Description
<i>Time delay</i>	This AMF is violated if the set time has expired.



If the same instance of the AMF is used for several safety functions, the delay time begins running from the first activation.

Parameter	Description
<i>Delay time</i>	<p>Amount of time by which the triggering of the reaction of a safety function is delayed.</p> <ul style="list-style-type: none"> • 12 ms ... 24 h <p>The time can be entered in milliseconds (ms), seconds (s), minutes (min) and hours (h). Each delay is a multiple of 12 ms, meaning that it is rounded up to the next multiple of 12.</p>

Example

Category: *Safety stop*

A robot whose axes are not referenced is to be allowed to be moved in Automatic mode for a limited time. Once this time has elapsed, e.g. after 2 hours, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
<i>Automatic mode</i>	<i>Position referencing</i>	<i>Time delay</i>	<i>Stop 1 (path-maintaining)</i>

13.14.13 Monitoring of forces and torques

An LBR is fitted with position and joint torque sensors in all axes. These make it possible to measure and react to external forces and torques.



If the permissible forces or torques are exceeded continuously due to jamming, it is possible to move the robot free by changing to CRR mode.

13.14.13.1 Axis torque monitoring

Axis torque monitoring can limit and monitor the torques of individual axes.

The following points must be observed when using axis torque monitoring:

- Successful torque referencing is a precondition.

AMF	Description
<i>Axis torque monitoring</i>	The AMF is violated if the torque of the monitored axis exceeds or falls below the configured torque limit.



If the AMF is violated and a safety stop triggered, the interaction forces may continue to increase due to the stopping distances of the robot. For this reason, the AMF may only be used in collaborative operation at reduced velocity. For this, the AMF can be combined with the AMF *Cartesian velocity monitoring*, *Axis velocity monitoring* or *Tool-related velocity component*.

Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Monitored axis</i>	<p>Axis to be monitored</p> <ul style="list-style-type: none"> • Axis1 ... Axis16 <p>Note: Axis1 ... Axis7 are used for an LBR.</p>
<i>Minimum torque [Nm]</i>	<p>Minimum permissible torque for the given axis</p> <ul style="list-style-type: none"> • -500 ... 500 Nm
<i>Maximum torque [Nm]</i>	<p>Maximum permissible torque for the given axis</p> <ul style="list-style-type: none"> • -500 ... 500 Nm

13.14.13.2 Collision detection

Collision detection monitors the external axis torques against a definable limit value.

The external axis torque is that part of the torque of an axis which is generated from the forces and torques occurring as the robot interacts with its environment. The external axis torque is not measured directly but is rather calculated using the dynamic robot model. The accuracy of the calculated values depends on the dynamics of the robot motion and of the interaction forces of the robot with its environment.

The following points must be observed when using collision detection:

- Successful position and torque referencing are preconditions.
- The load data of safety-oriented tools are taken into consideration (if active).
- If a safety-oriented fixed tool is configured, it must also be mounted on the robot flange.
- The load data of workpieces that are picked up are only taken into consideration if the current workpiece is communicated to the safety controller.
(>>> [15.10.5 "Transferring workpiece load data to the safety controller"](#)
[Page 412](#))
- The weight of the heaviest workpiece whose mass is configured in the safety-oriented project settings is taken into consideration.

 The AMF *Collision detection* does not automatically take into consideration possible errors in the workpiece load data. When configuring the collision detection, it is therefore necessary to set the lowest possible values for the maximum permissible external torque. In this way, significant deviations in the load data are interpreted as a collision and cause a violation of the AMF.

 Workpieces that have been picked up must not come loose unintentionally and fall down while the monitoring is active. The user must ensure this when using the AMF.

 If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured that the carrier kinematic system does not move while the AMF is being used. As long as the robot base of the monitored kinematic system is being accelerated, the safety integrity of the AMF is not assured.

 If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured, during use of the AMF, that the mounting direction of the monitored kinematic system does not differ from the configured mounting direction (e.g. due to tilting of the mobile platform). Otherwise, the safety integrity of the AMF is not assured.

AMF	Description
<i>Collision detection</i>	This AMF is violated if the external torque of at least one axis exceeds the configured limit value.

 If the AMF is violated and a safety stop triggered, the interaction forces may continue to increase due to the stopping distances of the robot. For this reason, the AMF may only be used in collaborative operation at reduced velocity. For this, the AMF can be combined with the AMF *Cartesian velocity monitoring*, *Axis velocity monitoring* or *Tool-related velocity component*.

 External forces on the robot or tool with short distances to the robot axes can only cause slight external torques in the robot axes under certain circumstances. If the AMFs are used, this can pose a safety risk, particularly in potential crushing situations during collaborative operation. Critical crushing situations can arise on the robot itself, between the robot and the surroundings or between the tool and the surroundings. It is therefore advisable to avoid potentially critical incidents of crushing by using suitable equipment for the robot cell and/or by using one of the following AMFs: *Cartesian workspace monitoring*, *Cartesian protected space monitoring*, *Axis range monitoring* or *Tool orientation*.

Parameter	Description
<i>Monitored kinematic system</i>	Kinematic system to be monitored <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Maximum external torque [Nm]</i>	Maximum permissible external torque <ul style="list-style-type: none"> • 0 ... 30 Nm

13.14.13.3 TCP force monitoring

Description

In TCP force monitoring, the external force acting on the tool or robot flange is monitored against a definable limit value.

The external force on the TCP is not measured directly but is rather calculated using the dynamic robot model. The accuracy of the calculated external force depends on the dynamics of the robot motion and of the actual force, among other things.

The following points must be observed when using TCP force monitoring:

- Successful position and torque referencing are preconditions.
 - The load data of safety-oriented tools are taken into consideration (if active).
 - If a safety-oriented fixed tool is configured, it must also be mounted on the robot flange.
 - The load data of workpieces that are picked up are only taken into consideration if the current workpiece is communicated to the safety controller.
- (>>> [15.10.5 "Transferring workpiece load data to the safety controller"](#)
[Page 412](#))
- The weight of the heaviest workpiece whose mass is configured in the safety-oriented project settings is taken into consideration.



The AMF *TCP force monitoring* automatically takes into consideration possible errors in the workpiece load data. For this reason, when configuring the monitoring, it is necessary to set a value for the maximum permissible external force at the TCP which is greater than the weight of the heaviest workpiece configured in the safety-oriented project settings.



Workpieces that have been picked up must not come loose unintentionally and fall down while the monitoring is active. The user must ensure this when using the AMF.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured that the carrier kinematic system does not move while the AMF is being used. As long as the robot base of the monitored kinematic system is being accelerated, the safety integrity of the AMF is not assured.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured, during use of the AMF, that the mounting direction of the monitored kinematic system does not differ from the configured mounting direction (e.g. due to tilting of the mobile platform). Otherwise, the safety integrity of the AMF is not assured.

AMF	Description
<i>TCP force monitoring</i>	This AMF is violated if the external force acting on the tool or robot flange exceeds the configured limit value.



If the AMF is violated and a safety stop triggered, the interaction forces may continue to increase due to the stopping distances of the robot. For this reason, the AMF may only be used in collaborative operation at reduced velocity. For this, the AMF can be combined with the AMF *Cartesian velocity monitoring*, *Axis velocity monitoring* or *Tool-related velocity component*.



External forces on the robot with short distances to the robot axes can only cause slight external torques in the robot axes under certain circumstances. If the AMFs are used, this can pose a safety risk, particularly in potential crushing situations during collaborative operation. Critical crushing situations can arise on the robot itself or between the robot and the surroundings. It is therefore advisable to avoid potentially critical incidents of crushing by using suitable equipment for the robot cell and/or by using one of the following AMFs: *Cartesian workspace monitoring*, *Cartesian protected space monitoring*, *Axis range monitoring* or *Tool orientation*.

Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Maximum TCP force [N]</i>	<p>Maximum permissible external force on the TCP</p> <ul style="list-style-type: none"> • 50 ... 1,000 N

Accuracy of force detection

- The accuracy of TCP force detection is dependent on the robot pose. The safety controller recognizes non-permissible poses and sets the AMF *TCP force monitoring* to "violated" with a corresponding error message.
Non-permissible poses are those in which it is possible for TCP forces to have a short distance to all robot axes. This applies to singularity poses and poses near singularities.
(>>> [14.11 "Singularities" Page 363](#))
- External forces on the robot reduce the accuracy of TCP force detection. In many cases, the safety controller can automatically detect the external forces acting on the robot. The AMF *TCP force monitoring* is violated in this case.



It is not possible to guarantee that the safety controller will always automatically detect external forces acting on the robot. The user must ensure that the external forces act exclusively on the TCP in order to assure the safety integrity of the AMF *TCP force monitoring*.

13.14.13.4 Direction-specific monitoring of the external force on the TCP

Description

The *Base-related TCP force component* AMF is used to monitor the external force acting in a specific direction on the tool or on the robot flange relative to a base coordinate system against a definable limit value.

As standard, the world coordinate system is used as the base coordinate system. No other base coordinate system can currently be defined for this monitoring function.

The AMF monitors the force along the component of a reference coordinate system. The orientation of the reference coordinate system corresponds as standard to the orientation of the base coordinate system. The orientation of the reference coordinate system relative to the base coordinate system can be modified in the AMF.

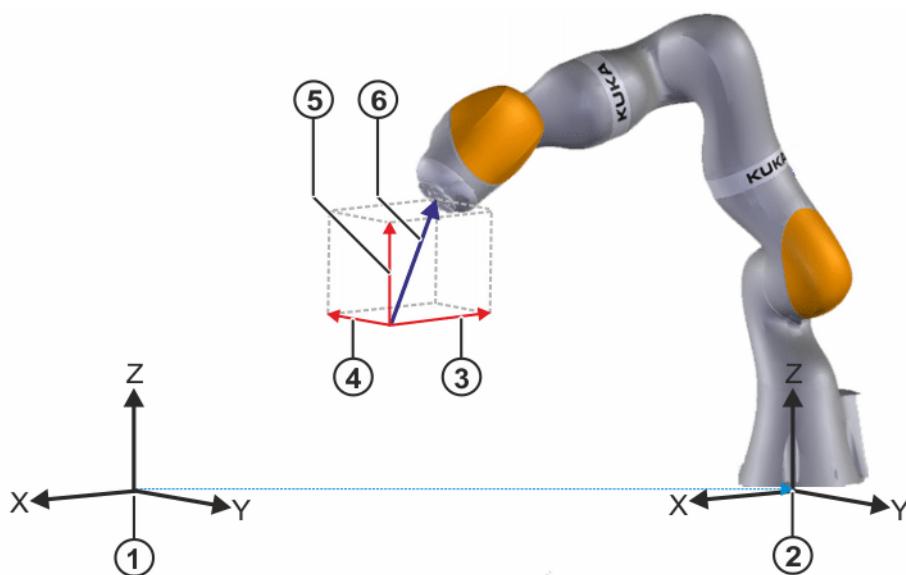


Fig. 13-21: Base-related TCP force monitoring

- 1 World coordinate system (= base coordinate system for the monitoring function)
- 2 World coordinate system is located as standard in the base of the robot.
- 3 Negative X component of the TCP force vector
- 4 Negative Y component of the TCP force vector
- 5 Positive Z component of the TCP force vector
- 6 TCP force vector

The external force on the TCP is not measured directly but is rather calculated using the dynamic robot model. The accuracy of the calculated external force depends on the dynamics of the robot motion and of the actual force, among other things.

The following points must be observed when monitoring base-related TCP force components:

- Successful position and torque referencing are preconditions.
- The load data of safety-oriented tools are taken into consideration (if active).
- If a safety-oriented fixed tool is configured, it must also be mounted on the robot flange.
- The load data of the heaviest safety-oriented workpiece are taken into consideration (if configured).
- The load data of workpieces that are picked up are only taken into consideration if the current workpiece is communicated to the safety controller.
(>>> [15.10.5 "Transferring workpiece load data to the safety controller"](#)
[Page 412](#))
- The weight of the heaviest workpiece whose mass is configured in the safety-oriented project settings is taken into consideration.



The *Base-related TCP force component* AMF automatically takes into consideration possible errors in the workpiece load data. For this reason, when configuring the monitoring, it is necessary to set a value for the maximum permissible external force at the TCP which is greater than the weight component of the mass of the heaviest workpiece in the monitored direction.

The monitoring of individual force components has advantages over TCP force monitoring:

- The monitoring function can be used in a larger workspace.
- Workpieces have no influence on horizontal monitoring functions.

Possible errors in the workpiece load data are not automatically taken into consideration. This results in additional external forces in the vertical direction. In the case of a monitoring function in the horizontal direction, these errors have no effect.



The AMF Base-related TCP force component may only be used if the direction in which hazardous forces can arise is known. At the same time, it must be ensured that no hazardous forces can arise in the non-monitored directions. If this is not the case, either the *AMF TCP force monitoring* must be used, or the other directions must also be monitored using the *AMF Base-related TCP force component*.



Workpieces that have been picked up must not come loose unintentionally and fall down while the monitoring is active. The user must ensure this when using the AMF.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured that the carrier kinematic system does not move while the AMF is being used. As long as the robot base of the monitored kinematic system is being accelerated, the safety integrity of the AMF is not assured.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured, during use of the AMF, that the mounting direction of the monitored kinematic system does not differ from the configured mounting direction (e.g. due to tilting of the mobile platform). Otherwise, the safety integrity of the AMF is not assured.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the reference coordinate system for the configured force component is defined relative to the robot base. The monitored direction of the force component moves with the carrier kinematic system.

AMF	Description
<i>Base-related TCP force component</i>	The AMF is violated if the external force acting along the monitored component of the TCP force vector exceeds the configured limit value.



If the AMF is violated and a safety stop triggered, the interaction forces may continue to increase due to the stopping distances of the robot. For this reason, the AMF may only be used in collaborative operation at reduced velocity. For this, the AMF can be combined with the *AMF Cartesian velocity monitoring*, *Axis velocity monitoring* or *Tool-related velocity component*.



External forces on the robot with short distances to the robot axes can only cause slight external torques in the robot axes under certain circumstances. If the AMFs are used, this can pose a safety risk, particularly in potential crushing situations during collaborative operation. Critical crushing situations can arise on the robot itself or between the robot and the surroundings.

It is therefore advisable to avoid potentially critical incidents of crushing by using suitable equipment for the robot cell and/or by using one of the following AMFs: *Cartesian workspace monitoring*, *Cartesian protected space monitoring*, *Axis range monitoring* or *Tool orientation*.

Parameter	Description
<i>Monitored kinematic system</i>	<p>Kinematic system to be monitored</p> <ul style="list-style-type: none"> • <i>First kinematic system</i>: Robot • <i>Second kinematic system</i>: No function • <i>Third kinematic system</i>: No function • <i>Fourth kinematic system</i>: No function
<i>Maximum TCP force [N]</i>	<p>Maximum external force acting along the monitored component of the TCP force vector</p> <ul style="list-style-type: none"> • 50 ... 1,000 N
<i>A [°]</i>	<p>Rotation of the TCP force vector about the Z axis of the base coordinate system</p> <ul style="list-style-type: none"> • 0° ... 359°
<i>B [°]</i>	<p>Rotation of the TCP force vector about the Y axis of the base coordinate system</p> <ul style="list-style-type: none"> • 0° ... 359°
<i>C [°]</i>	<p>Rotation of the TCP force vector about the X axis of the base coordinate system</p> <ul style="list-style-type: none"> • 0° ... 359°
<i>Component of the TCP force vector</i>	<p>Component of the TCP force vector that is monitored (direction of monitoring)</p> <ul style="list-style-type: none"> • <i>X positive or X negative</i> • <i>Y positive or Y negative</i> • <i>Z positive or Z negative</i>

Accuracy of force detection

- The accuracy of TCP force detection is also dependent on the robot pose. The safety controller recognizes non-permissible poses and sets the AMF *Base-related TCP force component* to "violated" with a corresponding error message.

Non-permissible poses are those in which it is possible for TCP forces to have a short distance to all robot axes. This applies to singularity poses and poses near singularities.

(>> [14.11 "Singularities" Page 363](#))

Depending on the direction of the monitored force component, the AMF *Base-related TCP force component* can be used closer to singularities than the AMF *TCP force monitoring*. This can result in a larger workspace.

- External forces on the robot reduce the accuracy of TCP force detection. In many cases, the safety controller can automatically detect the

external forces acting on the robot. The AMF *Base-related TCP force component* is violated in this case.



It is not possible to guarantee that the safety controller will always automatically detect external forces acting on the robot. The user must ensure that the external forces act exclusively on the TCP in order to assure the safety integrity of the AMF *Base-related TCP force component*.

Example

A workpiece is to be set down on a table. In order to be able to detect possible high crushing forces between the workpiece and the setdown surface, the force acting on the workpiece in the positive Z direction must be monitored. If the force in this direction exceeds a value of 50 N, a safety stop 1 (path-maintaining) is to be triggered.

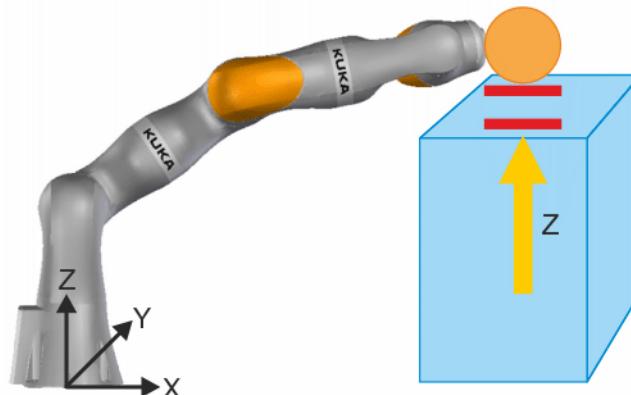


Fig. 13-22: Base-related TCP force monitoring in Z positive

Category: *Collision detection*

AMF1	AMF2	AMF3	Reaction
<i>Base-related TCP force component</i>	-	-	Stop 1 (path-maintaining)

The AMF *Base-related TCP force component* has the following parameters:

- *Monitored kinematic system: First kinematic system*
- *Maximum TCP force: 50 N*
- *Component of the TCP force vector: Z positive*
- *A = B = C: 0°*

13.15 Example of a safety configuration



The sole purpose of this example is to illustrate the safety configuration with KUKA Sunrise.Workbench.



The basis for a medical product's safety configuration is always a dedicated risk management process. The safety configuration displayed below serves only as an example and does not claim to be comprehensive.

13.15.1 Task

An LBR is used in an application in which it collaborates with a human. The tool installed on the robot is set as safety-oriented.

The operator places a workpiece in a workpiece pick-up position at regular intervals. One task of the robot is to check that the workpiece is present. It proceeds as follows: from a start position, it moves through the area accessible to humans (collaboration space) with a transfer motion. The purpose of this transfer motion is to achieve a pre-position of 20 cm above the waiting workpiece. It then moves toward the workpiece.

This lowering motion is parametrized with a force break condition (process limit value: 20 N). After reaching the process limit value, the robot uses its current position to determine whether the workpiece is present or not. The workpiece is not present if the robot can move all the way to the workpiece pick-up position. After it has finished checking, the robot moves back to the pre-position and out of the collaboration space.

13.15.2 Requirements

The following safety functions are required as part of the risk assessment for the above-described process:

1. It must be possible to stop the robot by pressing an external EMERGENCY STOP switch within reach of the operator.
2. The robot must not leave a defined workspace. The collaboration space is part of the workspace.
3. A transfer motion between the start position and pre-position can cause unintentional collisions with the operator. However, the space is designed in such a way that humans cannot be crushed. To reduce the risk of a collision, the maximum robot velocity for this space must be limited to 500 mm/s.
4. Collisions must be safely detected during the transfer motion and must cause the robot to come to a standstill. This is the case if, due to the collision, a torque of 15 Nm is exceeded in at least one axis.
5. During the motion between the pre-position and the workpiece pick-up position, there is a risk of the hand and arm of the operator being crushed. In order to ensure that the operator can respond appropriately and that the braking distances are sufficiently short, the robot velocity must not exceed 100 mm/s for this motion.
6. Furthermore, the robot must be brought to a standstill during the motion between the pre-position and the workpiece pick-up position if forces of more than 50 N arise. Force values of 20 N or more cause the lowering motion in the process to be aborted and are thus sufficiently below the latter limit.

13.15.3 Suggested solution for the task

In order for the requirements to be implemented, permanent and switchable safety monitoring functions must be configured:

- Permanent monitoring of the external EMERGENCY STOP device and workspace
- ESM state for the transfer motion between the start position and the pre-position
- ESM state for the motion between the pre-position and the workpiece pick-up position

To ensure a stable and smooth process sequence, the application must be designed in such a way that the defined limit values for the safety functions (velocity and workspace) are maintained.

The robot application implemented in the process described is not mentioned here.

Permanent safety monitoring

The EMERGENCY STOP function must be active throughout operation and the robot must not leave the workspace. Corresponding safety functions are configured in the Customer PSM table.

Configurable customer safety configuration						
Row	Active	Category	AMF 1	AMF 2	AMF 3	Reaction
1	<input checked="" type="checkbox"/>	External EMERGENCYSTOP	Input signal (1) Input for safety signal : InputCIB_SR.1	-	-	Stop 1 (path-maintaining)
2	<input checked="" type="checkbox"/>	Workspace monitoring	Cartesian workspace monitoring(1)	-	-	Stop 1 (path-maintaining)

Fig. 13-23: Permanent safety monitoring

Line	Description
1	External EMERGENCY STOP Implements requirement 1 An external EMERGENCY STOP is connected to a safe input. If the operator actuates the EMERGENCY STOP, a safety stop 1 (path-maintaining) is executed.
2	Cartesian workspace monitoring 2 Implements requirement 2 The workspace is represented by a safely monitored Cartesian workspace. If the robot leaves the configured space, a safety stop 1 (path-maintaining) is executed.

ESM state for transfer motion

An ESM state is defined for the transfer motion through the collaboration space between the start and pre-position. This is activated in the application before the transfer motion begins.

Velocity monitoring and collision detection must be active during the transfer motion in order to sufficiently reduce the danger of a collision between human and robot.

In order to avoid crushing at all times, an additional protected space is defined. This brings the robot to a standstill as soon as the distance between the robot or tool and the workpiece pick-up position becomes less than 15 cm.

State of the event-driven safety monitoring (ESM)			
Row	Active	AMF	Reaction
1	<input checked="" type="checkbox"/>	Cartesian velocity monitoring (1) Maximum velocity : 500 mm/s	Stop 1 (path-maintaining)
2	<input checked="" type="checkbox"/>	Collision detection (1) Maximum external torque : 15 Nm	Stop 1 (path-maintaining)
3	<input checked="" type="checkbox"/>	Cartesian protected space monitoring (1)	Stop 1 (path-maintaining)

Fig. 13-24: ESM state for transfer motion

Line	Description
1	Cartesian velocity monitoring Implements requirement 3 If a Cartesian velocity exceeds 500 mm/s, a safety stop 1 (path-maintaining) is executed.
2	Collision detection Implements requirement 4 If a collision causes an external torque of more than 15 Nm in at least one robot axis, a safety stop 1 (path-maintaining) is executed.
3	Protected space monitoring Implements the safety of the state regardless of the time and place of activation The safely monitored protected space encompasses the space above the workpiece pick-up position. As soon as the robot or the safely monitored tool enters this space, a safety stop 1 (path-maintaining) is executed.

ESM state for workpiece pick-up position

A specific ESM state is defined for the motions between the pre-position and the workpiece pick-up position. This is activated in the application before the lowering motion begins.

Velocity monitoring and force monitoring must be active during the motion in order to sufficiently reduce the danger of crushing the operator's hand or lower arm.

The state must ensure a sufficient degree of safety, regardless of the time or place of activation. The low permissible velocity and the active force monitoring mean that no further measures are necessary.

State of the event-driven safety monitoring (ESM)			
Row	Active	AMF	Reaction
1	<input checked="" type="checkbox"/>	Cartesian velocity (2) Maximum velocity : 100 mm/s	Stop 1 (on-path)
2	<input checked="" type="checkbox"/>	Tcp-force monitoring (1) Maximum Tcp-force : 50 N	Stop 0

Fig. 13-25: ESM state for workpiece pick-up position

Line	Description
1	Cartesian velocity monitoring Implements requirement 5 If a Cartesian velocity exceeds 100 mm/s, a safety stop 1 (path-maintaining) is executed.
2	Force monitoring Implements requirement 6 If a contact situation causes a force of more than 50 N to be exerted at the TCP, a stop 0 is executed.

13.16 Position and torque referencing

13.16.1 Position referencing

Description

Position referencing checks whether the saved zero position of the motor of an axis (= saved mastering position) corresponds to the actual mechanical zero position.

Referencing is carried out continuously by the system when an axis moves at less than 30 °/s. Referencing is successful when the mastering sensor detects the mechanical zero position of the axis in a narrow range around the saved zero position of the motor.

Referencing fails in the following cases:

- The mastering sensor does not detect the mechanical zero position of the axis in the range around the saved zero position of the motor.
- The mastering sensor detects the mechanical zero position of the axis at an unexpected point.

For robots without mastering sensor, the axis positions can only be referenced via an external system. The interface for external position referencing must be configured.

(>>> [13.16.4 "External position referencing" Page 316](#))

The safety integrity of the safety functions based upon this is limited until the position referencing test has been performed. This includes, for example, safely monitored Cartesian and axis-specific robot positions, safely monitored Cartesian velocities, TCP force monitoring and collision detection.

If position referencing fails on at least one axis, all AMFs based on safe axis positions are violated. (>>> ["Position-based AMFs" Page 321](#))

Requirements

The position of an axis is not referenced after the following events:

- Robot controller is rebooted.
- The axis is remastered.
- Torque referencing of the axis fails.
- The maximum torque of the joint torque sensor of the axis has been exceeded.



If the maximum torque of a joint torque sensor is exceeded for longer than 3 seconds, the brake of the corresponding axis is opened automatically for a few milliseconds. The opening of the brakes is repeated every 3 seconds until the maximum torque is no longer exceeded.

These events do not lead to a violation of the safe position-based safety functions. The robot can be moved, but the safety integrity of the safety functions is no longer assured.

The safety functions based on safe positions are only violated after these events if the position referencing of an axis fails. Referencing must be successfully carried out before safety-critical applications can be executed.

The position referencing status can be used as an AMF in the safety configuration. (>>> [13.14.6 "Evaluating the position referencing" Page 280](#))

Precondition

The position of an axis is referenced when the axis is moved over the saved zero position of the motor and the mastering sensor detects the zero position of the axis in a range of 0° +/- 0.5°.

Preconditions for this:

- The velocity at which the axis is moved over the zero position must be < 30 °/s.
- At the very least, a defined axis-specific range before and after the zero position must be passed through. The motion direction is not relevant.

The axis-specific range of motion is robot-specific:

Robot variant	A1	A2	A3	A4	A5	A6	A7
LBR Med 7 R800	±10.5°	±10.5°	±10.5°	±10.5°	±10.5°	±14°	±14°
LBR Med 14 R820	±9.5°	±9.5°	±10.5°	±10.5°	±10.5°	±14°	±14°

Execution

Position referencing of all axes is continuously performed by the system when the above conditions are met. Position referencing can be carried out in a targeted manner the following ways:

- Automatically while the program is running, when an axis moves over the zero position at less than 30 °/s.
- Jogging each axis individually over the zero position.
- Executing the application prepared by KUKA. The axes are moved over the zero position from the vertical stretch position.

An application for position and torque referencing of the LBR Med is available from Sunrise.Workbench. Position and torque referencing can be carried out simultaneously with this application.

(>>> [13.16.3 "Creating an application for position and torque referencing" Page 315](#))



If it is not possible to reference from the vertical stretch position, a user-specific application for position referencing must be created and executed.

13.16.2 Torque referencing

Description

The LBR Med has a joint torque sensor in each axis which reliably determines the torque currently acting on the axis. These data are used for calculating and monitoring externally acting torques or Cartesian forces, for example.

When referencing the joint torque sensors a check is made to determine whether the expected external torque of an axis matches the actual external torque of the axis:

- The expected torque is calculated for each axis using the robot model and the specified load data.
- The actual torque is determined for each axis on the basis of the value measured by the joint torque sensor.

If the difference between the expected torque and the actual torque exceeds a certain tolerance value, the referencing of the torque sensors has failed.

The safety integrity of the safety functions based upon this is limited until the torque referencing test has been performed successfully. This includes, for example, axis torque and TCP force monitoring as well as collision detection.

If torque referencing fails on at least one axis, all AMFs based on safe torque values are violated. (>>> ["Axis torque-based AMFs" Page 321](#))

Requirements

The joint torque sensor of an axis is not referenced after the following events:

- Robot controller is rebooted.
- Position referencing of the axis fails.
- The maximum torque of the joint torque sensor of the axis has been exceeded.



If the maximum torque of a joint torque sensor is exceeded for longer than 3 seconds, the brake of the corresponding axis is opened automatically for a few milliseconds. The opening of the brakes is repeated every 3 seconds until the maximum torque is no longer exceeded.

These events do not lead to a violation of the safety functions based on safe torque values. The robot can be moved, but the safety integrity of the safety functions is no longer assured.

The safety functions based on safe torque values are only violated after these events if torque referencing of one axis fails. Referencing must be successfully carried out before safety-critical applications can be executed.

The torque referencing status can be used as an AMF in the safety configuration. (>>> [13.14.7 "Evaluating the torque referencing" Page 281](#))

Execution

An application for position and torque referencing of the LBR Med is available from Sunrise.Workbench. Position and torque referencing can be carried out simultaneously with this application.

(>>> [13.16.3 "Creating an application for position and torque referencing" Page 315](#))

A total of 10 measured joint torque values must be given for each axis. For this purpose, 5 measurement poses are defined in the application, each of which can be addressed with positive and negative directions of axis rotation. If the poses cannot be addressed, they must be adapted in the application.



During torque referencing, each of the measurement poses must be addressed in sequence with positive and negative directions of axis rotation before the next measurement pose is addressed. The safety integrity of the referencing of the joint torque sensors is otherwise not given.

The safety controller evaluates the external torque for all 10 measured values and determines the mean value of the external torque for each axis. Referencing is successful if this mean value is below a defined tolerance. Otherwise, referencing has failed.



Before carrying out torque referencing, the user must ensure the following points:

- The load data of the fixed tool mounted on the robot flange must match the load data with which the fixed safety-oriented tool is configured.
- The load data of the tool coupled with the fixed tool (if present) must match the load data of the activated safety-oriented tool.
- The load data of the workpiece that is picked up (if present) must match the load data of the activated safety-oriented workpiece.
- Workpieces that are not taken into consideration by the safety controller must not be picked up.
- No supplementary loads, e.g. dress packages, may be fastened to the robot.

If one of these points is not met, the safety integrity of the referencing of the joint torque sensors is not given.



If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), the user must ensure the following points:

- The carrier kinematic system may not be moved during torque referencing.
- The mounting direction of the kinematic system to be referenced may not differ from the configured mounting direction (e.g. due to tilting of the mobile platform).

If one of these points is not met, the safety integrity of the referencing of the joint torque sensors is not given.

13.16.3 Creating an application for position and torque referencing

Description

The following points must be observed if the application for torque referencing needs to be edited due to measurement poses which cannot be addressed:

- The joint torque values must be measured while the robot is stationary.
- A wait time of at least 2.5 seconds in which the robot does not move is required between the moment the measurement pose is reached and the measurement itself. Wait times which are too short can reduce the referencing accuracy due to oscillations on the robot.
- The measurement is started with the method `sendSafetyCommand()`.
- There may be a maximum of 15 s between 2 consecutive measurements.

Procedure

1. In the **Package Explorer** view, select the desired project or package in which the application is to be created.
2. Select the menu sequence **File > New > Sunrise application**. The wizard for creating a new Sunrise application is opened.
3. In the folder **Application examples > LBR iiwa**, select the application **Position and GMS referencing** and click on **Finish**.

The application **PositionAndGMSReferencing.java** is created in the source folder of the project and opened in the editor area of Sunrise.Workbench.

4. If measurement poses cannot be addressed due to the system configuration, adapt them in the application.
5. Perform project synchronization in order to transfer the application to the robot controller.

13.16.4 External position referencing

Description

The user has the possibility of implementing his own test method or an external system for position referencing, e.g. a tracker, a navigation system or an absolute encoder. Confirmation that the external position referencing has been successfully carried out must be communicated to the robot controller via a safety-oriented input.

The input for external position referencing can be configured in the safety-oriented project settings. If the external signal at this input changes from LOW to HIGH and back to LOW within 2 seconds, the position referencing has been successfully confirmed.



External position referencing is merely an interface for setting the position referencing. The safety maintenance personnel is responsible for the correct use of the referencing input, i.e.:

- They must provide a suitable test method for position mastering.
- The test method for position mastering must be sufficiently accurate. The accuracy of the position-based AMFs depends on the accuracy of the test method.
- They must ensure that the input is only set after the position mastering has been successfully tested.

13.16.4.1 Configuring the input for external position referencing

Description

The safety-oriented input that allows external position referencing is configured in the safety-oriented project settings.

Procedure

1. Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Change project settings** from the context menu.
The **Properties for [Sunrise Project]** window opens.
2. Select **Sunrise > Safety** in the directory in the left area of the window.
3. Make the following settings in the right-hand part of the window:
 - Set the check mark at **Allow external position referencing**.
 - Select the input that is to be used for external position referencing.
The inputs of the discrete safety interface and of the Ethernet safety interface can be used as long as they are configured in WorkVisual.
 - The enabling device of the hand guiding device can also be used as an input.
4. Click on **OK** to save the settings and close the window.

13.17 Safety acceptance overview

The medical device must not be put into operation until the safety acceptance procedure has been completed successfully. For successful safety acceptance, the points in the checklists must be completed fully and confirmed in writing by the safety maintenance technician.



The completed checklists, confirmed in writing, must be kept as documentary evidence.

Safety acceptance must be carried out in the following cases:

- Following initial start-up or recommissioning of the LBR Med
- After a change to the LBR Med
- After a change to the safety configuration
- After a software update, e.g. of the System Software

Safety acceptance after a software update is only necessary if the ID of the safety configuration (= checksum) has changed as a result of the update.

The system integrator determines the required safety functions on the basis of his risk management process. Once the safety configuration is activated on the robot controller, the safety functions must be tested for correct functioning.



If a test requires persons to be present in the danger zone, the test must be conducted in T1 mode.

The following checklists must be used to verify whether the configured safety parameters have been correctly transferred.

The checklists must be processed in the following order:

1. Checklist for basic test of the safety configuration
(>>> [13.17.1 "Checklist – System safety functions" Page 317](#))
2. Checklists for checking the mapped safety-oriented tools
(>>> [13.17.2 "Tool selection table checklist" Page 321](#))
(>>> [13.17.3 "Checklists for safety-oriented tools" Page 323](#))
3. Checklist for checking the rows used in the Customer PSM table
(>>> [13.17.4 "Checklist for rows used in table Customer PSM" Page 328](#))
4. Checklists for checking the ESM states which have been used and not used
(>>> [13.17.5 "Checklists for ESM states" Page 329](#))
5. Checklists for checking the AMFs used
(>>> [13.17.6 "Checklists for AMFs used" Page 330](#))
6. Checklists for checking the safety-oriented project settings
(>>> [13.17.7 "Checklists – safety-oriented project settings" Page 341](#))

It is possible to create a report of the current safety configuration.

(>>> [13.17.8 "Creating a safety configuration report" Page 343](#))

13.17.1 Checklist – System safety functions

Checklist

- Serial number of the robot: _____
- ID of the safety configuration: _____
- Name of safety maintenance technician: _____

No.	Activity	Yes	Not relevant
1	Operator safety: is all operator safety equipment configured, properly connected and tested for correct function?		
2	Operator safety: a stop is triggered if AUT or T2 mode is active with the operator safety open.		
3	Operator safety: a manual reset function is present and activated.		
4	Brake test: is a brake test planned and has an application been created for this purpose?		
5	Hand guiding device enabling state: is the enabling device of the hand guiding device configured, properly connected and tested for correct function?		
6	Local EMERGENCY STOP: are all local EMERGENCY STOP devices configured, properly connected and tested for correct function?		
7	External EMERGENCY STOP: are all external EMERGENCY STOP devices configured, properly connected and tested for correct function?		
8	Local and external EMERGENCY STOP: are the local and external EMERGENCY STOPs each configured as an individual AMF in a row of the PSM table?		
9	If unplugging of the smartPAD is allowed in the station configuration: is at least one external EMERGENCY STOP device installed?		
10	Safety stop: is all operator safety equipment configured, properly connected and tested for correct function?		
11	Safe operational stop: is all equipment for the safe operational stop configured, properly connected and tested for correct function?		
12	When using position-based AMFs: is the limited safety integrity of the position-based AMFs taken into consideration in the absence of position referencing? (>>> <i>"Position-based AMFs" Page 321</i>) Note: Initiation of the safe state in the absence of position referencing can be configured by using the AMF <i>Position referencing</i> .		
13	When using position-based AMFs: has position referencing been carried out successfully?		
14	If external position referencing is used: has a suitable test method for position mastering been provided?		
15	If external position referencing is used: has it been ensured that the input is only set after successful testing?		
16	Velocity monitoring: have all necessary velocity monitoring tests been configured and tested?		
17	Manual guidance: has it been configured in such a way that appropriate velocity monitoring is active in every operating mode for manual guidance?		

No.	Activity	Yes	Not relevant
18	If using the enabling device of the hand guiding device as an input for deactivating safety functions: Has it been taken into consideration that using the enabling device as an input may result in safety functions being deactivated during manual guidance?		
19	Workspace monitoring: have all necessary workspace monitoring tests been configured and tested?		
20	Cartesian workspace monitoring functions: has it been taken into consideration that the system does not monitor the entire structure of the robot, tool and workpiece against the space violation, but only the monitoring spheres on the robot and tool?		
21	Collision detection: have all necessary HRC functionalities been configured?		
22	Collision detection: has it been configured in such a way that velocity monitoring is also always active when collision detection is active?		
23	Collision detection: has it been configured in such a way that velocity monitoring is also always active when TCP force monitoring or monitoring of a base-related TCP force component is active?		
24	Collision detection: When using the AMF <i>Base-related TCP force component</i> : has it been ensured that no hazardous forces can arise in the non-monitored directions?		
25	Collision detection: is a safety stop 0 configured for all safety monitoring functions in order to detect crushing situations?		
26	When using axis torque-based AMFs: is the limited safety integrity of the axis torque-based AMFs taken into consideration in the absence of position referencing and/or torque referencing? (>>> <i>"Axis torque-based AMFs"</i> Page 321) Note: Initiation of the safe state in the absence of position and/or torque referencing can be configured by using the AMF <i>Position referencing</i> and/or the AMF <i>Torque referencing</i> .		
27	In the configuration of all rows in the PSM table and all ESM states, has it been taken into account that the safe state of the AMFs is the "violated" state (state "0")? Note: In the event of an error, an AMF goes into the safe state.		
28	PSM configuration: in the configuration of output signals, has it been taken into account for the safety reaction that an output is LOW (state "0") in the safe state?		

No.	Activity	Yes	Not relevant
29	<p>PSM configuration: Was a check carried out during configuration of the “Brake” safety reaction to see whether there could be an increased risk due to rapid switching to and from the violation state of the AMFs with which the Cartesian velocity monitoring is linked?</p> <p>Note: In the case of rapid switching between the states, it is possible that the “Brake” safety reaction could lead to no reduction in velocity.</p>		
30	ESM configuration: are all ESM states consistent, i.e. does each individual ESM state sufficiently reduce all dangers?		
31	Have torque and position referencing been carried out successfully?		
32	<p>If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit):</p> <p>Has the fact been taken into consideration that, with the AMF <i>Cartesian workspace monitoring / Cartesian protected space monitoring</i>, the monitoring space is defined relative to the base of the monitored kinematic system and moves with the carrier kinematic system?</p>		
33	<p>If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit):</p> <p>Has the fact been taken into consideration that, with the AMF <i>Cartesian velocity monitoring</i>, it is not the absolute velocity, but the velocity of the monitored kinematic system relative to the carrier kinematic system that is monitored?</p>		
34	<p>If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit):</p> <p>Has the fact been taken into consideration that, with the AMF <i>Tool-related velocity component</i>, it is not the absolute velocity, but the velocity of the monitored kinematic system relative to the carrier kinematic system that is monitored?</p>		
35	<p>If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform):</p> <p>Has the fact been taken into consideration that, with the AMF <i>Tool orientation</i>, the reference orientation is defined relative to the carrier kinematic system and moves with the carrier kinematic system?</p>		
36	<p>If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform):</p> <p>Has the fact been taken into consideration that, with the AMF <i>Base-related TCP force component</i>, the reference coordinate system is defined relative to the robot base and the monitored direction of the force component moves with the carrier kinematic system?</p>		
37	<p>If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit):</p> <p>Has the fact been taken into consideration that the safety integrity of the AMFs <i>Collision detection</i>, <i>TCP force monitoring</i> and <i>Base-related TCP force component</i> is only assured as long as the carrier kinematic system is at a standstill?</p>		

By signing, the signatory confirms the correct and complete performance of the safety acceptance test.

Place, date

Place, date

Place, date

Signature

Signature

Signature

Position-based AMFs

The safety integrity of position-based AMFs is only given without limitations when position referencing has been carried out successfully. (Position-based AMFs are only supported by robot types that have corresponding sensor systems, e.g. an LBR.)

AMF	Position referencing	Torque referencing
<i>Standstill monitoring of all axes</i>	✓	✗
<i>Axis range monitoring</i>	✓	✗
<i>Cartesian velocity monitoring</i>	✓	✗
<i>Tool-related velocity component</i>	✓	✗
<i>Cartesian workspace monitoring</i>	✓	✗
<i>Cartesian protected space monitoring</i>	✓	✗
<i>Tool orientation</i>	✓	✗

Axis torque-based AMFs

The safety integrity of axis torque-based AMFs is only given without limitations when position and/or torque referencing has been carried out successfully. (Axis torque-based AMFs are only supported by robot types that have corresponding sensor systems, e.g. an LBR.)

AMF	Position referencing	Torque referencing
<i>Axis torque monitoring</i>	✗	✓
<i>Collision detection</i>	✓	✓
<i>TCP force monitoring</i>	✓	✓
<i>Base-related TCP force component</i>	✓	✓

13.17.2 Tool selection table checklist

Description

If one of the following AMFs is used in the safety configuration, the mapped safety-oriented tools must be checked:

- *Cartesian velocity monitoring*

Only if the monitoring spheres on the tool are configured as a structure to be monitored.

- *Tool-related velocity component*
- *Cartesian workspace monitoring / Cartesian protected space monitoring*

Only if the monitoring spheres on the tool are configured as a structure to be monitored.

- *Tool orientation*
- *Collision detection*
- *TCP force monitoring*
- *Base-related TCP force component*
- *Torque referencing*

For each activated row of the tool selection table, it is necessary to check whether the selected tool has been correctly assigned to the kinematic system. This can be done, for example, using a suitable test for verification of the tool parameters. A test is suitable if it checks a tool parameter, the value of which differs considerably from that of the other safety-oriented tools:

- In the case of considerably different geometric dimensions, it is advisable to check whether the geometric tool data have been specified correctly.

(>>> [13.17.3.5 "Geometry data of the tool" Page 326](#))

- In the case of considerably different load data, it is advisable to check whether the load data of the tool have been specified correctly.

(>>> [13.17.3.6 "Load data of the tool" Page 327](#))

- In the case of considerably different parameters when using the *Tool orientation* AMF, it is advisable to check whether the tool orientation that is to be monitored has been configured correctly.

(>>> [13.17.3.3 "Tool orientation" Page 325](#))

- In the case of considerably different parameters when using the *Tool-related velocity component* AMF, it is advisable to check whether the velocity component that is to be monitored has been configured correctly.

(>>> [13.17.3.4 "Points and orientation for tool-related velocity component" Page 326](#))



For each row in the tool selection table, the points in the checklist must be executed and separately documented.

Precondition

- If the tool is activated via an input: the configured input is HIGH.
- If the tool is always active: only the fixed tool is mounted on the kinematic system.

Checklist

- Row no.: _____
- *Assigned kinematic system*: _____
- *Selected tool*: _____
- *Activation signal* (always active / name of input):

No.	Activity	Yes
1	The row has been checked successfully: the correct tool has been assigned to the kinematic system.	

13.17.3 Checklists for safety-oriented tools

13.17.3.1 Pickup frame of fixed tools

Description

If the fixed tool of a kinematic system can pick up activatable tools, and if one of the following AMFs is used simultaneously in the safety configuration, the position and orientation of the pickup frame of the fixed tool (= default frame for motions of the fixed tool) must be checked:

- *Cartesian velocity monitoring*
Only if the monitoring spheres on the tool are configured as a structure to be monitored.
- *Tool-related velocity component*
- *Cartesian workspace monitoring / Cartesian protected space monitoring*
Only if the monitoring spheres on the tool are configured as a structure to be monitored.
- *Tool orientation*
- *Collision detection*
- *TCP force monitoring*
- *Base-related TCP force component*
- *Torque referencing*

If the fixed tool of a kinematic system can pick up workpieces, and if one of the following AMFs is used simultaneously in the safety configuration, the position and orientation of the pickup frame of the fixed tool must again be checked:

- *Collision detection*
- *TCP force monitoring*
- *Base-related TCP force component*
- *Torque referencing*

If the fixed tool is used for picking up workpieces (no activatable tool can be coupled), the pickup frame of the fixed tool must be verified. For this purpose, check whether the load data of the tool have been specified correctly. The test must be carried out with as heavy a workpiece as possible.

(>>> [13.17.3.6 "Load data of the tool" Page 327](#))

If the fixed tool is used for picking up an activatable tool (e.g. in the case of a tool changer), the pickup frame of the fixed tool must be verified by means of a suitable test with the activatable tool coupled to the fixed tool. A test is suitable if the parameters of the pickup frame have a major influence on the test result:

- In the case of large values for the position of the pickup frame and/or a protruding coupled tool, it is advisable to check whether the geometric tool data have been specified correctly.

(>>> [13.17.3.5 "Geometry data of the tool" Page 326](#))

If the tool is relevant for the monitoring of a tool-related velocity component, it is advisable to check whether the velocity component to be monitored has been configured correctly.

(>>> [13.17.3.4 "Points and orientation for tool-related velocity component" Page 326](#))

- In the case of large values for the position of the pickup frame and a heavy coupled tool, it is advisable to check whether the load data of the tool have been specified correctly.
(>>> [13.17.3.6 "Load data of the tool" Page 327](#))
- If only the tool orientation is monitored for a kinematic system, the orientation of the pickup frame can be verified. The test is only suitable if none of the other AMFs mentioned above is used in the safety configuration for this kinematic system.
(>>> [13.17.3.3 "Tool orientation" Page 325](#))



For each configured fixed tool in the tool selection table, the points in the checklist must be executed and separately documented if the following preconditions are met:

- The tool can pick up workpieces or activatable tools.
- AND: One of the AMFs listed here is used in the safety configuration for the kinematic system to which the tool is assigned.

Precondition

- If the fixed tool picks up workpieces:
 - The tool has picked up the heaviest possible workpiece.
 - In the application, the correct workpiece has been transferred to the safety controller.
- If the fixed tool picks up activatable tools:
 - An activatable tool is coupled to the fixed tool.
 - The input used to activate the coupled tool is HIGH.

Checklist

- Name of the fixed tool: _____

No.	Activity	Yes
1	Position and orientation of the pickup frame have been checked successfully.	

13.17.3.2 Pickup frames of activatable tools

Description

If an activatable tool of a kinematic system can pick up a workpiece and, at the same time, one of the following AMFs is used in the safety configuration, the position and orientation of the pickup frame of the activatable tool must be checked:

- *Collision detection*
- *TCP force monitoring*
- *Base-related TCP force component*
- *Torque referencing*

The pickup frame of the tool can be verified by checking whether the load data of the tool have been specified correctly. The test must be carried out with as heavy a workpiece as possible.



For each configured activatable tool in the tool selection table, the points in the checklist must be executed and separately documented if the following preconditions are met:

- The tool can pick up workpieces.
- AND: One of the AMFs listed here is used in the safety configuration for the kinematic system to which the tool is assigned.

Precondition

- The input used to activate the tool is HIGH.
- The tool has picked up the heaviest possible workpiece.
- In the application, the correct workpiece has been transferred to the safety controller.

Checklist

- Name of the activatable tool: _____

No.	Activity	Yes
1	Position and orientation of the pickup frame have been checked successfully.	

13.17.3.3 Tool orientation

Description

If one of the following AMFs is used in the safety configuration, it is necessary to check whether the tool orientation that is to be monitored has been configured correctly:

- *Tool orientation*

For verification, the *Tool orientation* AMF must be verified successfully.

(>>> [13.17.6.24 "AMF Tool orientation" Page 338](#))



The checklist must be completed for every safety-oriented tool that is mapped in the tool selection table to a kinematic system for which the *Tool orientation* AMF is configured.

Precondition

- Position referencing has been carried out successfully (not necessary in the case of a mobile platform).
- The correct safety-oriented tool is active.
- If a fixed tool is checked, no activatable tool is coupled.

Checklist

- Name of the tool: _____

No.	Activity	Yes
1	The correct configuration of the tool orientation to be monitored has been successfully checked.	

13.17.3.4 Points and orientation for tool-related velocity component

Description

If one of the following AMFs is used in the safety configuration, it is necessary to check whether the points and orientation of the tool-related velocity component have been configured correctly:

- *Tool-related velocity component*

For verification, the *Tool-related velocity component* AMF must be verified successfully.

(>>> [13.17.6.25 "AMF Tool-related velocity component" Page 340](#))



The checklist must be completed for every safety-oriented tool that could be taken into consideration in the *Tool-related velocity component* AMF according to the tool selection table.

Precondition

- Position referencing has been carried out successfully (not necessary in the case of a mobile platform).
- The correct safety-oriented tool is active.
- If a fixed tool is checked, no activatable tool is coupled.

Checklist

- Name of the tool: _____

No.	Activity	Yes
1	The correct configuration of the points and orientation of the tool-related velocity component to be monitored has been successfully checked.	

13.17.3.5 Geometry data of the tool

Description

If one of the following AMFs is used in the safety configuration, it is necessary to check that the geometric tool data have been entered correctly:

- *Cartesian velocity monitoring*

Only if the monitoring spheres on the tool are configured as a structure to be monitored.

- *Cartesian workspace monitoring / Cartesian protected space monitoring*

Only if the monitoring spheres on the tool are configured as a structure to be monitored.

The geometric tool data can be tested by intentionally violating one of the configured monitoring spaces with each tool sphere and checking the reaction.

If no space monitoring functions are used, only the position of the sphere center points is relevant. The configured Cartesian velocity limit can be tested by intentionally exceeding this velocity for each tool sphere and checking the reaction.



The checklist must be completed for every safety-oriented tool that is mapped in the tool selection table to a kinematic system for which one of the AMFs referred to above is configured.

Precondition

- Position referencing has been carried out successfully (not necessary in the case of a mobile platform).
- The correct safety-oriented tool is active.
- If a fixed tool is checked, no activatable tool is coupled.

Checklist

- Name of the safety-oriented tool: _____

No.	Activity	Yes	Not relevant
1	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
2	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
3	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
4	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
5	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
6	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		

13.17.3.6 Load data of the tool

Description

If any of the following AMFs is used in the safety configuration, it is necessary to check that the load data of the safety-oriented tool have been entered correctly:

- Collision detection*
- TCP force monitoring*
- Base-related TCP force component*
- Torque referencing*

It is advisable to check the load data by performing torque referencing in several suitable poses. Suitable poses include those with similar axis angles in the horizontal extended position and the following properties:

- Axes A2, A4 and A6 are loaded.
- The poses differ in their axis value of A7 by 90°.

If the load data are correct, torque referencing must be successful.



The checklist must be completed for every safety-oriented tool that is mapped in the tool selection table to a kinematic system for which one of the AMFs referred to above is configured.

Precondition

- Position and torque referencing have been carried out successfully.
- The correct safety-oriented tool is active.
- If the load data of a fixed tool are being checked: No activatable tool is coupled.
- If a workpiece is picked up by the tool to check the load data: In the application, the correct workpiece has been transferred to the safety controller.

Checklist

- Name of the tool: _____
- Mass: _____
- Center of mass:
 - MS X: _____
 - MS Y: _____
 - MS Z: _____

No.	Activity	Yes
1	Have the load data of the tool been correctly entered and checked?	

13.17.4 Checklist for rows used in table *Customer PSM*

Description

Each row in the *Customer PSM* table must be tested to verify that the expected reaction is triggered. If the reaction is to switch off an output, the test must also ensure that the output is correctly connected.

The “Brake” reaction can be checked by moving the robot at a velocity that exceeds the limit value of the Cartesian velocity monitoring. As soon as all other AMFs of the PSM row are violated, the velocity must be reduced to a value below the limit value. There must be no stop to a complete standstill.

A PSM row in the table can be tested by violating 2 of its AMFs at a time. It is then possible to test the remaining AMF separately in a targeted manner. If fewer than 3 AMFs are used in a row, the unassigned columns are regarded as violated AMFs.

(>>> [13.17.6 "Checklists for AMFs used" Page 330](#))



For each row in the table *Customer PSM*, the points in the checklist must be executed and separately documented.

Checklist

- Row no.: _____

No.	Activity	Yes	Not relevant
1	AMF 1 was tested successfully. Precondition: AMF 2 and AMF 3 are violated. AMF 1: _____		
2	AMF 2 was tested successfully. Precondition: AMF 1 and AMF 3 are violated. AMF 2: _____		

No.	Activity	Yes	Not relevant
3	AMF 3 was tested successfully. Precondition: AMF 1 and AMF 2 are violated. AMF 3: _____		

13.17.5 Checklists for ESM states

13.17.5.1 Used ESM states

Description

Each row in the ESM state must be tested to verify that the expected reaction is triggered when the configured AMF is violated.

(>>> [13.17.6 "Checklists for AMFs used" Page 330](#))



For each ESM state, the points in the checklist must be executed and separately documented.

Checklist

- ESM state: _____

No.	Activity	Yes	Not relevant
1	AMF row 1 was tested successfully. AMF row 1: _____		
2	AMF row 2 was tested successfully. AMF row 2: _____		
3	AMF row 3 was tested successfully. AMF row 3: _____		
4	AMF row 4 was tested successfully. AMF row 4: _____		
5	AMF row 5 was tested successfully. AMF row 5: _____		
6	AMF row 6 was tested successfully. AMF row 6: _____		
7	AMF row 7 was tested successfully. AMF row 7: _____		
8	AMF row 8 was tested successfully. AMF row 8: _____		
9	AMF row 9 was tested successfully. AMF row 9: _____		
10	AMF row 10 was tested successfully. AMF row 10: _____		
11	AMF row 11 was tested successfully. AMF row 11: _____		
12	AMF row 12 was tested successfully. AMF row 12: _____		

No.	Activity	Yes	Not relevant
13	AMF row 13 was tested successfully. AMF row 13: _____		
14	AMF row 14 was tested successfully. AMF row 14: _____		
15	AMF row 15 was tested successfully. AMF row 15: _____		
16	AMF row 16 was tested successfully. AMF row 16: _____		
17	AMF row 17 was tested successfully. AMF row 17: _____		
18	AMF row 18 was tested successfully. AMF row 18: _____		
19	AMF row 19 was tested successfully. AMF row 19: _____		
20	AMF row 20 was tested successfully. AMF row 20: _____		

13.17.5.2 Non-used ESM states

Description

All ESM states which are not used must be tested as to whether a safety stop is triggered when the ESM state is selected.

Checklist

No.	Activity	Yes	Not relevant
1	Selection of non-used ESM state 1 was tested successfully.		
2	Selection of non-used ESM state 2 was tested successfully.		
3	Selection of non-used ESM state 3 was tested successfully.		
4	Selection of non-used ESM state 4 was tested successfully.		
5	Selection of non-used ESM state 5 was tested successfully.		
6	Selection of non-used ESM state 6 was tested successfully.		
7	Selection of non-used ESM state 7 was tested successfully.		
8	Selection of non-used ESM state 8 was tested successfully.		
9	Selection of non-used ESM state 9 was tested successfully.		
10	Selection of non-used ESM state 10 was tested successfully.		

13.17.6 Checklists for AMFs used

An AMF which is used in more than one row in the PSM table must be separately tested in each row. (>>> [13.17.4 "Checklist for rows used in table Customer PSM" Page 328](#))

13.17.6.1 AMF smartPAD Emergency Stop

Checklist

No.	Activity	Yes
1	The configured reaction is triggered by pressing the E-STOP on the smartPAD.	

13.17.6.2 AMF smartPAD enabling switch inactive

Checklist

No.	Activity	Yes
1	The configured reaction is triggered by releasing an enabling switch on the smartPAD.	

13.17.6.3 AMF smartPAD enabling switch panic active

Checklist

No.	Activity	Yes
1	The configured reaction is triggered by pressing an enabling switch down fully on the smartPAD.	

13.17.6.4 AMF Hand guiding device enabling inactive

All enabling switches and panic switches configured for the hand guiding device must be tested.

Checklist

- Used input, enabling switch 1: _____
- Used input, enabling switch 2: _____
- Used input, enabling switch 3: _____
- Used input, panic switch 1: _____
- Used input, panic switch 2: _____
- Used input, panic switch 2: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by releasing enabling switch 1.		
2	The configured reaction is triggered by pressing fully down on enabling switch 1 (panic position).		
3	The configured reaction is triggered by releasing enabling switch 2.		
4	The configured reaction is triggered by pressing fully down on enabling switch 2 (panic position).		
5	The configured reaction is triggered by releasing enabling switch 3.		
6	The configured reaction is triggered by pressing fully down on enabling switch 3 (panic position).		

13.17.6.5 AMF Hand guiding device enabling active

All enabling switches configured for the hand guiding device must be tested.

Checklist

- Used input, enabling switch 1: _____
- Used input, enabling switch 2: _____
- Used input, enabling switch 3: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by pressing enabling switch 1.		
2	The configured reaction is triggered by pressing enabling switch 2.		
3	The configured reaction is triggered by pressing enabling switch 3.		

13.17.6.6 AMF Test mode

Checklist

No.	Activity	Yes
1	The configured reaction is triggered in T1.	
2	The configured reaction is triggered in T2.	
3	The configured reaction is triggered in CRR.	

13.17.6.7 AMF Automatic mode

Checklist

No.	Activity	Yes
1	The configured reaction is triggered in AUT.	

13.17.6.8 AMF Reduced-velocity mode

Checklist

No.	Activity	Yes
1	The configured reaction is triggered in T1.	
2	The configured reaction is triggered in CRR.	

13.17.6.9 AMF High-velocity mode

Checklist

No.	Activity	Yes
1	The configured reaction is triggered in T2.	
2	The configured reaction is triggered in AUT.	

13.17.6.10 AMF Motion enable

Checklist

No.	Activity	Yes
1	The configured reaction is triggered if, for example, the E-STOP is pressed on the smartPAD.	

13.17.6.11 AMF Input signal

Checklist

- Monitoring instance: _____
- Input for safety signal: _____

No.	Activity	Yes
1	The configured reaction is triggered if the input is LOW (state "0").	

13.17.6.12 AMF Standstill monitoring of all axes

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

No.	Activity	Yes
1	The configured reaction is triggered if one axis of the monitored kinematic system is moved.	

13.17.6.13 AMF Axis torque monitoring

Description

The AMF can be tested by displaying the current measured axis torques on the smartPAD and then subjecting the monitored axis to gravitational force or manual loading.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored axis: _____
- Maximum permissible axis torque: _____
- Minimum permissible axis torque: _____

No.	Activity	Yes
1	The configured reaction is triggered if the axis torque exceeds the maximum permissible value.	
2	The configured reaction is triggered if the axis torque falls below the minimum permissible value.	

13.17.6.14 AMF Axis velocity monitoring

Description

The AMF can be tested by moving the monitored axis at a velocity of approx. 10% over the configured velocity limit.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored axis: _____
- Maximum permissible axis velocity: _____

No.	Activity	Yes
1	The configured reaction is triggered if the maximum permissible axis velocity is exceeded.	

13.17.6.15 AMF Position referencing



This AMF is violated after the robot controller is rebooted.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

No.	Activity	Yes
1	The configured reaction is triggered if one or more axes of the monitored kinematic system are not referenced.	

13.17.6.16 AMF Torque referencing



This AMF is violated after the robot controller is rebooted.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

No.	Activity	Yes
1	The configured reaction is triggered if one or more axes of the monitored kinematic system are not referenced.	

13.17.6.17 AMF Axis range monitoring

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored axis: _____
- Lower limit of the permissible axis range: _____
- Upper limit of the permissible axis range: _____

No.	Activity	Yes
1	The configured reaction is triggered if the lower limit of the permissible axis range is exceeded.	
2	The configured reaction is triggered if the upper limit of the permissible axis range is exceeded.	

13.17.6.18 AMF Cartesian velocity monitoring

Description

In order to test the configured velocity limit of the AMF, the monitored kinematic system must be moved in such a way that the monitored point that moves fastest is moved at a velocity of approx. 10% above the configured limit value.

It is also necessary to check whether the structure to be monitored (robot and/or tool) is correctly configured:

- If both structures are monitored, the velocity monitoring must be violated both by the monitoring points on the robot and by the monitoring points on the tool.
- If only one of the two structures is monitored, the velocity monitoring must be violated by either the monitoring points on the robot or the monitoring points on the tool.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored structure: _____
- Maximum permissible Cartesian velocity: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the maximum permissible Cartesian velocity is exceeded at the fastest moving monitored point.		
2	The configured reaction is triggered if the velocity monitoring is violated exclusively by the monitoring points on the robot.		
3	The configured reaction is triggered if the velocity monitoring is violated exclusively by the monitoring points on the tool.		

13.17.6.19 AMF Cartesian workspace monitoring / Cartesian protected space monitoring

Description

The first step is to test whether the orientation of the monitoring space is correctly configured. This involves violating 2 adjoining space surfaces at a minimum of 3 different points in each case.

The second step is to test whether the size of the monitoring space is correctly configured. This involves violating the other space surfaces at a minimum of 1 point in each case. In total, at least 10 points must be addressed.

The third step is to test whether the structure to be monitored is correctly configured. This involves violating the space monitoring, both with the monitoring spheres on the robot and on the tool (if both structures are to

be monitored), or just with the monitoring spheres on the robot or on the tool.

Checklist

- Type of monitoring space: _____
- Instance of the monitoring space: _____
- Monitored kinematic system: _____
- Monitored structure: _____
- Offset of the origin of the monitoring space:
 - X: _____ mm
 - Y: _____ mm
 - Z: _____ mm
- Orientation of the origin of the monitoring space:
 - A: _____ °
 - B: _____ °
 - C: _____ °
- Length of the monitoring space: _____ mm
- Width of the monitoring space: _____ mm

No.	Activity	Yes	Not relevant
1	The correct configuration of the orientation of the monitoring space has been tested as described above. The configured reaction is triggered every time a monitoring space is violated.		
2	The correct configuration of the size of the monitoring space has been tested as described above. The configured reaction is triggered every time a monitoring space is violated.		
3	The configured reaction is triggered if the space monitoring is violated on the monitoring spheres on the robot.		
4	The configured reaction is triggered if the space monitoring is violated on the monitoring spheres on the tool.		

13.17.6.20 AMF Collision detection

Description

The AMF can be tested by displaying the current measured external axis torques on the smartPAD and then loading the individual axes.

Precondition

- Torque referencing has been carried out successfully.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Maximum permissible external axis torque: _____

No.	Activity	Yes
1	The configured reaction is triggered if the external torque of one or more axes of the monitored kinematic system exceeds the maximum permissible external torque.	

13.17.6.21 AMF TCP force monitoring

Description

In order to test the AMF, suitable measuring equipment is required, e.g. a spring balance.

During the test, it must be noted that the monitoring function automatically takes into consideration possible errors in the workpiece load data. This means that the response may be triggered before the permissible external TCP force has been reached.

Premature triggering of the response can be prevented by performing the test as follows:

- Tool has picked up no workpiece.
- In the application, transfer no workpiece to the safety controller.
- Apply the TCP force in the direction of gravitational acceleration (vertically downwards) or perpendicular to gravitational acceleration.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Maximum permissible external TCP force: _____

No.	Activity	Yes
1	The configured reaction is triggered if the external force acting on the TCP exceeds the maximum permissible force.	

13.17.6.22 AMF Base-related TCP force component

Description

In order to test the AMF, suitable measuring equipment is required, e.g. a spring balance.

For the test, a force that is just above the configured maximum permissible TCP force must be exerted on the tool or robot flange in 2 different directions:

- Along the direction of the configured force component
- In a direction perpendicular to the direction of the configured force component

This is to ensure that the AMF is only violated if an excessive force is applied along the direction of the configured force component.

During the test, it must be noted that the monitoring function automatically takes into consideration possible errors in the workpiece load data. This means that the response may be triggered before the permissible external TCP force has been reached.

If, for example, no workpiece is picked up during the test, and if no workpiece has been transferred to the safety controller in the application, this force that is additionally taken into consideration corresponds to the weight of the heaviest workpiece configured in the safety-oriented project settings. The force that is taken into consideration counteracts gravitational acceleration (it is applied vertically upwards).

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

- Maximum permissible external TCP force: _____
- Monitored component of the force vector: _____
- Base-related orientation:
 - A: _____ °
 - B: _____ °
 - C: _____ °

No.	Activity	Yes
1	The configured reaction is triggered if the external force acting along the direction of the monitored component of the force vector exceeds the maximum permissible force.	
2	The configured reaction is not triggered if the force is applied in a direction perpendicular to the direction of the monitored force component.	

13.17.6.23 AMF Time delay

Checklist

- Instance of delay: _____
- Delay time: _____

No.	Activity	Yes
1	The configured reaction is triggered after the configured time.	

13.17.6.24 AMF Tool orientation

Description

In order to test the AMF, the permissible orientation cone must be violated at 3 straight lines offset by approx. 120° to one another. This ensures that the permissible orientation angle, the orientation of the reference vector and the tool orientation are correctly configured.

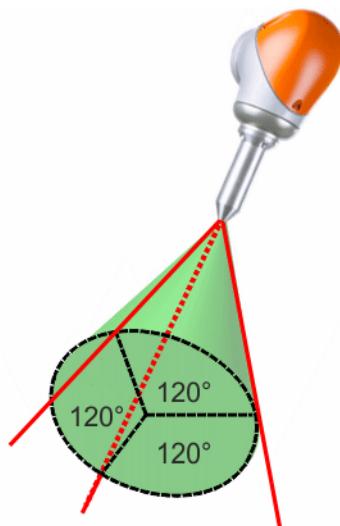


Fig. 13-26: Position of the straight lines on the monitoring cone

The orientation angles of the Z axis of the tool orientation frame are defined using 3 straight lines situated on the edge of the monitoring cone and offset at 120° to one another. These orientation angles must be set in order to test the AMF *Tool orientation*. The AMF must be violated when all 3 orientation angles are exceeded.

Procedure

The procedure describes an example of how the correct configuration of the monitoring cone can be tested.

1. Orient the Z axis of the tool orientation frame according to the reference vector relative to the world coordinate system.
2. Exceed the permissible deviation angle by tilting the tool orientation frame in B or C.
The configured reaction must be triggered.
3. Orient the Z axis of the tool orientation frame according to the reference vector relative to the world coordinate system.
If a stop reaction has been configured, the robot must be switched to CRR mode in order for it to be moved.
4. Rotate the tool orientation frame by 120° in A.
5. Exceed the permissible deviation angle by tilting the tool orientation frame in B or C.
The configured reaction must be triggered.
6. Orient the Z axis of the tool orientation frame according to the reference vector relative to the world coordinate system.
If a stop reaction has been configured, the robot must be switched to CRR mode in order for it to be moved.
7. Rotate the tool orientation frame by 120° in A.
8. Exceed the permissible deviation angle by tilting the tool orientation frame in B or C.
The configured reaction must be triggered.



The test must be carried out for every instance of the AMF and for every tool that is mapped in the tool selection table to a kinematic system for which the AMF *Tool orientation* is configured.

(>>> [13.17.3.3 "Tool orientation" Page 325](#))



If a fixed tool is configured that can pick up activatable tools, the fixed tool must be checked in addition to the activatable tools without an activatable tool being coupled.

Example: 4 instances of the AMF are configured, with 5 different tools that can be selected for fastening to the fixed tool. At least 6 tests are necessary to verify all AMF instances and tool orientations. If, on the other hand, only 2 different tools are available for selection for fastening on the fixed tool, 4 tests are sufficient.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Safety-oriented tool used: _____
- Orientation of the reference vector relative to the world coordinate system:
 - A: _____ °
 - B: _____ °
 - C: _____ °
- Permissible workspace (deviation angle): _____ °

No.	Activity	Yes
1	The correct configuration of the monitoring cone has been checked and the configured reaction is triggered when the permissible angle for all 3 straight lines has been exceeded.	

13.17.6.25 AMF Tool-related velocity component

Description

A motion must be programmed for every point that has been defined as a point for the tool-related velocity component in the configuration parameters of the currently mounted tool.

During this test motion, it must be ensured that the tested tool point has a higher velocity along the monitored direction than the other tool point for the tool-related velocity component. This can be achieved by including a suitable reorientation of the tool in the test motion.

The test motion must be performed twice:

- Once at a velocity slightly above the maximum permissible velocity
- Once at a velocity slightly below the maximum permissible velocity

This is to ensure that the velocity limit is violated only by the tested point.

The test must be repeated for every tool point that has been defined as a point for the tool-related velocity component.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Tool used: _____
- Tested tool point: _____
- Monitored component of the velocity vector: _____
- Maximum permissible Cartesian velocity of the monitored component: _____

No.	Activity	Yes
1	The configured reaction is triggered if the motion is executed with a velocity that exceeds the maximum permissible velocity.	
2	The configured reaction is not triggered if the motion is executed with a velocity that is below the maximum permissible velocity.	



The test must be performed for all instances of the AMF and for all possible safety-oriented tools.

(>>> [13.17.3.4 "Points and orientation for tool-related velocity component" Page 326](#))



If a fixed tool is configured that can pick up activatable tools, the fixed tool must be checked in addition to the activatable tools. When checking the fixed tool, no activatable tool may be coupled.

Example: 4 instances of the AMF are configured. At the same time, 5 different tools are available for selection for mounting on the fixed tool. At least 6 tests are necessary to verify all AMF instances and tool-related velocity components. If, on the other hand, only 2 different tools are available for selection for fastening on the fixed tool, 4 tests are sufficient.

13.17.7 Checklists – safety-oriented project settings

13.17.7.1 smartPAD unplugging allowed

Description

The safety parameter **smartPAD unplugging allowed** in the station configuration determines whether it is possible to move the robot with the smartPAD unplugged. The configured response must be tested while the robot is moving in Automatic mode.

- Disconnection not allowed:
If the smartPAD is disconnected, the robot is stopped with a safety stop.
- Disconnection allowed:
If the smartPAD is disconnected, the robot continues moving.

Checklist

- **smartPAD unplugging allowed** (true/false): _____

No.	Activity	Yes
1	The expected response occurs if the smartPAD is unplugged while the robot is moving in Automatic mode.	

13.17.7.2 Allow muting via input

Description

If an input that allows the deactivation of safety functions is configured in the safety-oriented project settings, a safety stop triggered by one of the following AMFs can be briefly cancelled:

- *Axis range monitoring*
- *Cartesian workspace monitoring*
- *Cartesian protected space monitoring*
- *Tool orientation*
- *Tool-related velocity component*
- *Standstill monitoring of all axes*
- *Position referencing*
- *Torque referencing*
- *Axis torque monitoring*
- *Collision detection*
- *TCP force monitoring*
- *Base-related TCP force component*

The configured input must be tested. For this, a safety stop must be triggered using at least one of the above AMFs, e.g. by violating a workspace or activating a standstill monitoring function.

- Deactivation of safety functions via an input not allowed:
If the configured input is set to HIGH and retains this value, the robot cannot be moved when the corresponding AMF is violated.
- Deactivation of safety functions via an input allowed:
If the configured input is set to HIGH and retains this value, the robot can be moved for 5 seconds even though the corresponding AMF is violated.

Checklist

- **Allow muting via input** (true/false): _____
- Configured input: _____

No.	Activity	Yes
1	The expected response occurs when the configured input is set to HIGH and an attempt is made to move the robot.	

13.17.7.3 Allow external position referencing**Description**

If an input that allows external position referencing is configured in the safety-oriented project settings, this input must be tested.

The axis positions are not referenced after a reboot of the robot controller. If the safety configuration contains a position-based AMF, the warning “Axis is not referenced” is displayed. The warning may no longer be displayed if the input via which the external position referencing is carried out is set to HIGH for less than 2 seconds..

Checklist

- **Allow external position referencing** (true/false): _____
- Configured input: _____

No.	Activity	Yes
1	The expected response occurs if the configured input is set to HIGH for less than 2 seconds.	

13.17.7.4 Mass of the heaviest workpiece**Description**

If workpieces are picked up in an application of a kinematic system and, at the same time, one of the following AMFs is used in the safety configuration, it is necessary to check whether the mass of the heaviest workpiece has been configured correctly in the safety-oriented project settings:

- *TCP force monitoring*
- *Base-related TCP force component*

To check the configuration, 2 workpieces of different masses must be transferred in the application with `setSafetyWorkpiece(...)`:

- The mass of the workpiece is 1 g greater than the configured mass of the heaviest workpiece (invalid workpiece mass):
If the invalid workpiece mass is transferred, a message must be displayed, indicating that the workpiece load data are invalid.
- The mass of the workpiece is the same as the configured mass of the heaviest workpiece (valid workpiece mass):
When the valid workpiece mass is transferred, no such message may be displayed.

On transferring the workpieces, one of the specified AMFs must also be active.

Checklist

- Monitored kinematic system: _____
- Configured mass of the heaviest workpiece: _____

No.	Activity	Yes
1	If <code>setSafetyWorkpiece(...)</code> is used to transfer a workpiece whose mass is 1 g greater than the configured mass of the heaviest workpiece, a message stating that the workpiece load data used are invalid is displayed.	
2	If <code>setSafetyWorkpiece(...)</code> is used to transfer a workpiece whose mass is the same as the configured mass of the heaviest workpiece, no message stating that the workpiece load data used are invalid is displayed.	

13.17.8 Creating a safety configuration report

Description

A report of the current safety configuration can be created and displayed in the Editor. The report can be edited and printed for documentation purposes.

The safety configuration report contains the following information for the unambiguous assignment of the safety configuration:

- Name of the Sunrise project to which the safety configuration belongs
- Safety version used
- Safety ID (checksum of the safety configuration)
The safety ID must match the ID of the safety configuration which is activated on the robot controller and is to be tested.
- Date and time of the last modification to the safety configuration

Checklists

The report provides the following checklists matching the safety configuration:

- Checklist for checking the rows used in the *Customer PSM* table
- Checklists for checking the ESM states which have been used and not used
- Checklists for checking the AMFs used
- Checklists for checking the safety-oriented project settings



The checklists provided by the safety configuration report are not sufficient for a complete safety acceptance procedure. The following additional checklists must be used for complete safety acceptance:

- Checklist for basic test of the safety configuration
- Checklists for checking the safety-oriented tool
- Checklist for checking the tool selection table

Warnings

The safety configuration is checked. There are warnings for the following situations:

- One row in the *Customer PSM* table is deactivated.
- One row in an ESM state is deactivated.
- Unplugging of the smartPAD is allowed, but no external EMERGENCY STOP is used.

- The input for deactivating safety functions is used in the tool selection table.
- Warning of the possible need to perform the brake test if a position-based or torque-based monitoring function is configured.
- The “Brake” safety reaction is configured.
A check must be carried out to ensure that there is no increased risk due to rapid switching to and from the violation state of the AMFs with which the Cartesian velocity monitoring is linked.

The safety maintenance technician must give reasons why a warning may be ignored.

Procedure

- Right-click on the desired project in the **Package Explorer** view and select **Sunrise > Create safety configuration report** from the context menu.
The report of the current safety configuration is created and opened in the editor area.

14 Basic principles of motion programming

This chapter describes the theoretical principles of motion programming. The programming of motions in KUKA Sunrise.Workbench is described in the following chapter: (>>> [15 "Programming" Page 367](#))

14.1 Overview of motion types



The start point of a motion is always the end point of the previous motion.

The following motion types can be programmed as an individual motion:

- Point-to-point motion (PTP)
(>>> [14.2 "PTP motion type" Page 345](#))
- Linear motion (LIN)
(>>> [14.3 "LIN motion type" Page 346](#))
- Circular motion (CIRC)
(>>> [14.4 "CIRC motion type" Page 346](#))
- Manual guidance motion with hand guiding device
(>>> [14.7 "Manual guidance motion type" Page 353](#))

The following types of motion can be programmed as segments of a CP spline block:

- Linear motion (LIN)
- Circular motion (CIRC)
- Polynomial motion (SPL)

The following types of motion can be programmed as segments of a JP spline block:

- Point-to-point motion (PTP)
(>>> [14.6 "Spline motion type" Page 347](#))

The following motions are known as CP ("Continuous Path") motions:

- LIN, CIRC, SPL, CP spline blocks

The following motions are known as JP ("Joint Path") motions:

- PTP, JP spline blocks

14.2 PTP motion type

The robot guides the TCP along the fastest path to the end point. The fastest path is generally not the shortest path in space and is thus not a straight line. As the motions of the robot axes are simultaneous and rotational, curved paths can be executed faster than straight paths.

PTP is a fast positioning motion. The exact path of the motion is not predictable, but is always the same, as long as the general conditions are not changed.

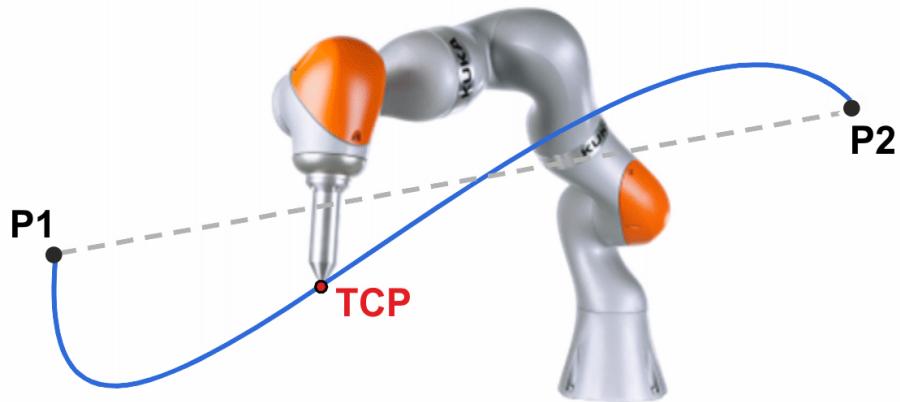


Fig. 14-1: PTP motion

14.3 LIN motion type

The robot guides the TCP at the defined velocity along a straight path in space to the end point.

In a LIN motion, the robot configuration of the end pose is not taken into account.

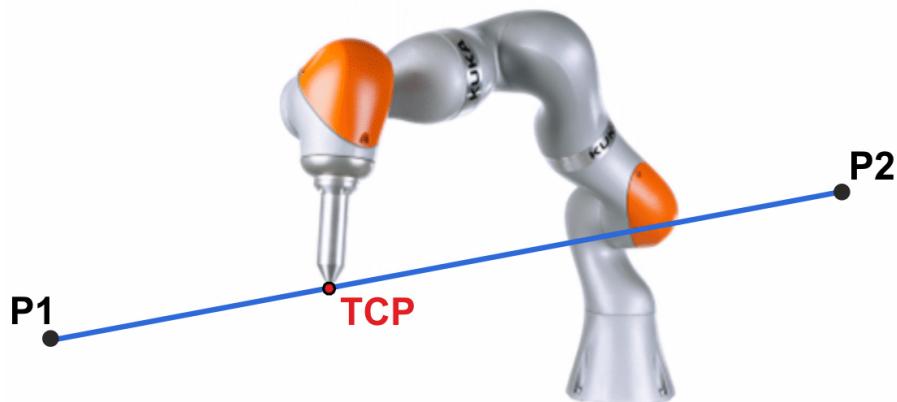


Fig. 14-2: LIN motion

14.4 CIRC motion type

The robot guides the TCP at the defined velocity along a circular path to the end point. The circular path is defined by a start point, auxiliary point and end point.

In a CIRC motion, the robot configuration of the end pose is not taken into account.

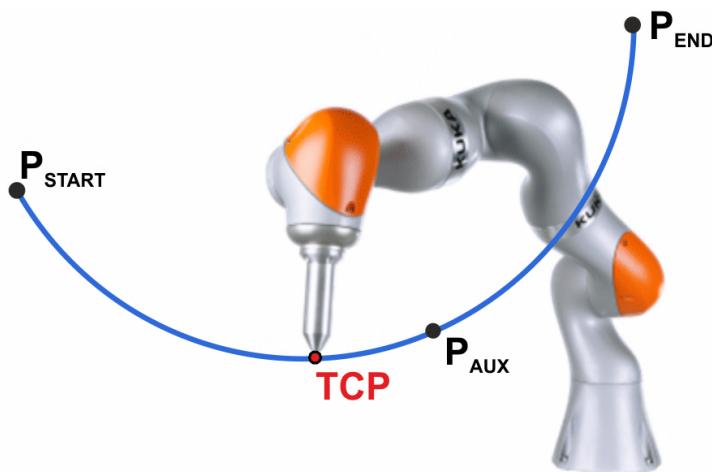


Fig. 14-3: CIRC motion

14.5 SPL motion type

The motion type SPL enables the generation of curved paths. SPL motions are always grouped together in spline blocks. The resulting paths run smoothly through the end points of the SPL motion.

In an SPL motion, the robot configuration of the end pose is not taken into account.



Curved lines are achieved by grouping together 2 or more SPL segments. If a single SPL segment is executed, the result is the same as for a LIN command.

14.6 Spline motion type

Spline is a motion type that is particularly suitable for complex, curved paths. With a spline motion, the robot can execute these complex paths in a continuous motion. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, however.

Splines are programmed in spline blocks. A spline block is used to group together several individual motions as an overall motion. The spline block is planned and executed by the robot controller as a single motion block.

The motions contained in a spline block are called spline segments.

- A CP spline block can contain SPL, LIN and CIRC segments.
- A JP spline block can contain PTP segments.

In a Cartesian spline motion, the robot configuration of the end pose is not taken into account.

The configuration of the end pose of a spline segment depends on the robot configuration at the start of the spline segment.

Path of a spline block

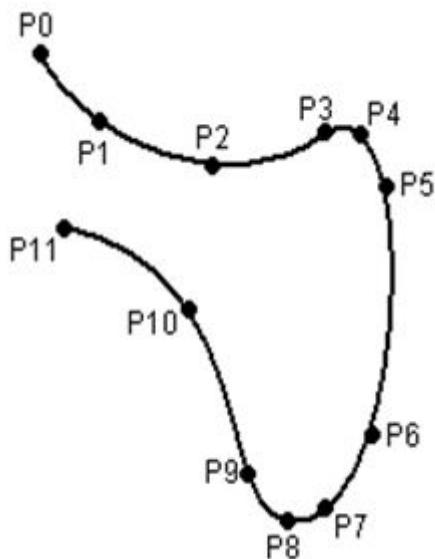


Fig. 14-4: Curved path with spline block

- The path is defined by means of points that are located on the path. These points are the end points of the individual spline segments.
 - All points are passed through without exact positioning.
Exception: The velocity is reduced to 0.
(>> [14.6.1 "Velocity profile for spline motions" Page 348](#))
 - If all points are situated in a plane, then the path is also situated in this plane.
 - If all points are situated on a straight line, then the path is also a straight line.
- There are a few cases in which the velocity is reduced.
(>> [14.6.1 "Velocity profile for spline motions" Page 348](#))
- The path always remains the same, irrespective of the override setting, velocity or acceleration.
- Circles and tight radii are executed with great precision.

14.6.1 Velocity profile for spline motions

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.

The path always remains the same, irrespective of the override setting, velocity or acceleration.

Only dynamic effects, such as those caused by high tool loads or the installation angle of the robot, may result in slight path deviations.

Reduction of the velocity

With spline motions, the velocity falls below the programmed velocity in the following cases:

- Tight corners, e.g. due to abrupt change in direction
- Major reorientation
- Motion in the vicinity of singularities

Reduction of the velocity due to major reorientation can be avoided with spline segments by programming the orientation control `SplineOrientationType.Ignore`.

(>>> [14.9 "Orientation control with LIN, CIRC, SPL" Page 356](#))

Reduction of the velocity to 0

With spline motions, exact positioning is carried out in the following cases:

- Successive spline segments with the same end points
- Successive LIN and/or CIRC segments. Cause: inconstant velocity direction.

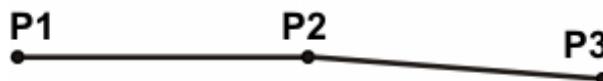


Fig. 14-5: Exact positioning at P2

In the case of LIN-CIRC transitions, the velocity also drops to 0 if the straight line is a tangent of the circle. This is caused by the fact that at the transition point between the straight line (curvature equals 0) and the circle (curvature is not equal to 0) the curvature characteristic is not constant.

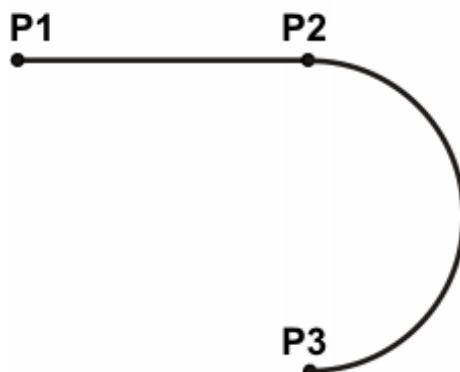


Fig. 14-6: Exact positioning at P2

Exceptions:

- In the case of successive LIN segments that result in a straight line and in which the orientations change uniformly, the velocity is not reduced.

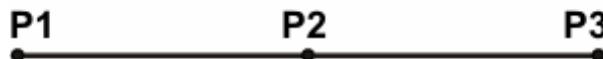


Fig. 14-7: P2 is executed without exact positioning.

- In the case of a CIRC-CIRC transition, the velocity is not reduced if both circles have the same center point and the same radius and if the orientations change uniformly. Since the required accuracy is difficult to achieve by teaching the end point and auxiliary point, calculation of the points on the circle is recommended.

14.6.2 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 LIN segment or vice versa, the path changes in the previous segment and the next segment.

Example 1

Original path:

```
Spline mySpline = new Spline(  
    splgetApplicationData().getFrame("/P1")),  
    splgetApplicationData().getFrame("/P2")),  
    splgetApplicationData().getFrame("/P3")),  
    splgetApplicationData().getFrame("/P4")),  
    circ(getApplicationData().getFrame("/P5"),  
        getApplicationData().getFrame("/P6")),  
    splgetApplicationData().getFrame("/P7")),  
    lin(getApplicationData().getFrame("/P8"))  
)  
;  
// ...  
robot.move(ptp(getApplicationData().getFrame("/P0")));  
robot.move(mySpline);
```

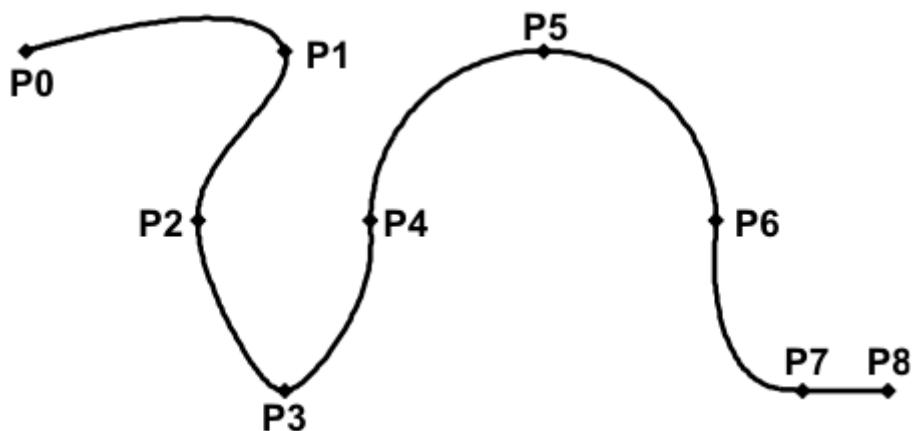


Fig. 14-8: Original path

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 a CIRC segment and a circular path is thus defined.

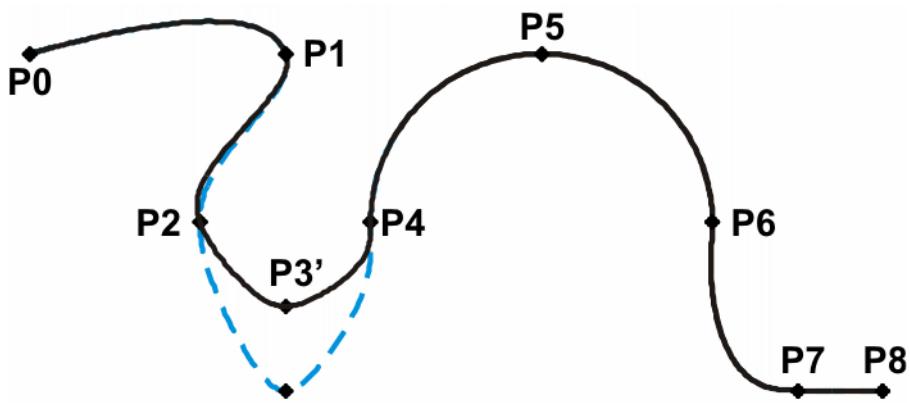


Fig. 14-9: Point has been offset

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 LIN. The path changes in segments P1 - P2, P2 - P3 and P3 - P4.

```
Spline mySpline = new Spline(
    splgetApplicationData().getFrame("/P1")),
    splgetApplicationData().getFrame("/P2")),
    lin(getApplicationData().getFrame("/P3")),
    splgetApplicationData().getFrame("/P4")),
    circ(getApplicationData().getFrame("/P5")),
    getApplicationData().getFrame("/P6")),
    splgetApplicationData().getFrame("/P7")),
    lin(getApplicationData().getFrame("/P8"))
);
// ...
robot.move(ptp(getApplicationData().getFrame("/P0")));
robot.move(mySpline);
```

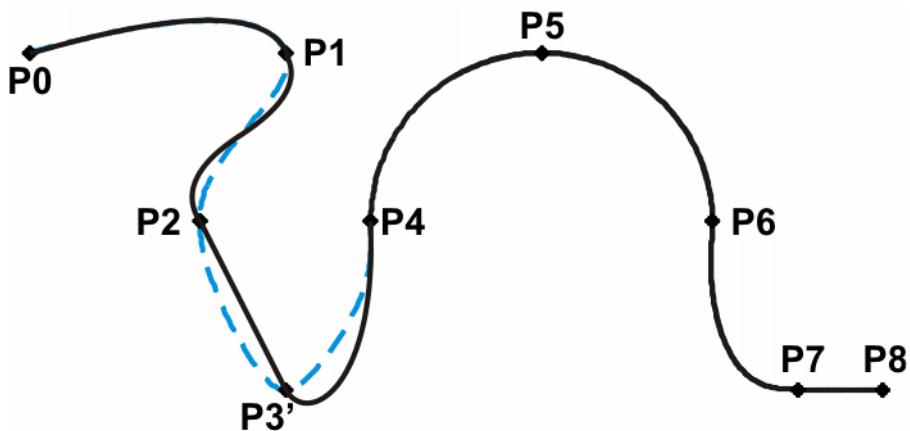


Fig. 14-10: Segment type has been changed

Example 2

Original path:

```
Spline mySpline = new Spline(
    splgetApplicationData().getFrame("/P2")),
    splgetApplicationData().getFrame("/P3")),
```

```

spl(getApplicationData().getFrame("/P4")),
spl(getApplicationData().getFrame("/P5")),
);
// ...
robot.move(mySpline);

```

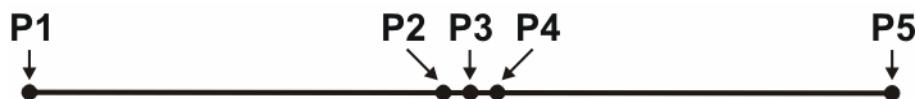


Fig. 14-11: Original path

The following frame coordinates were taught:

Frame	X	Y	Z
P2	100.0	0.0	0.0
P3	102.0	0.0	0.0
P4	104.0	0.0	0.0
P5	204.0	0.0	0.0

A point is offset relative to the original path:

P3 is moved slightly in the Y direction. This causes the path to change in all the segments illustrated.

Frame	X	Y	Z
P3	102.0	1.0	0.0

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.

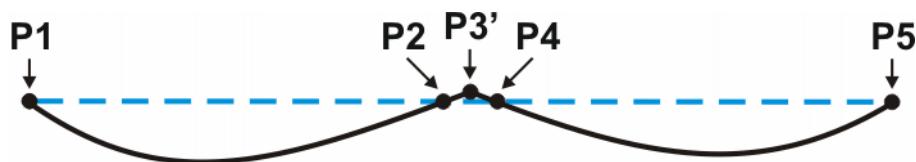


Fig. 14-12: Point has been offset

Remedy:

- Distribute the points more evenly.
- Program straight lines (except very short ones) as LIN segments

14.6.3 LIN-SPL-LIN transition

In the case of a LIN-SPL-LIN 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.

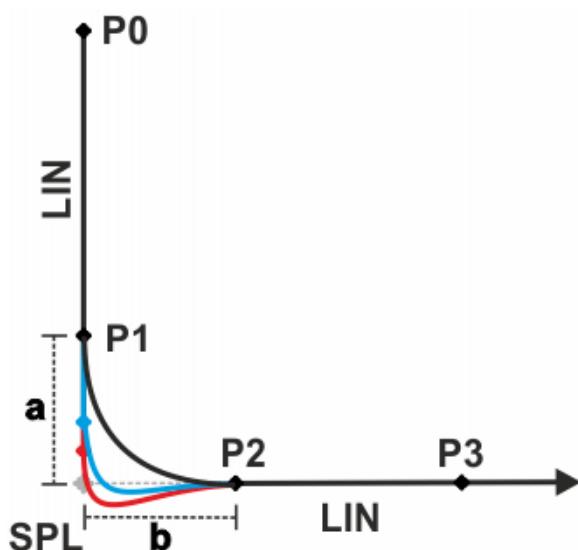


Fig. 14-13: LIN-SPL-LIN

The path remains inside the smaller angle if the following conditions are met:

- The extensions of the two LIN segments intersect.
 - $\frac{2}{3} \leq a/b \leq 3/2$
- a = distance from start point of the SPL segment to intersection of the LIN segments
b = distance from intersection of the LIN segments to end point of the SPL segment

14.7 Manual guidance motion type

Description

The robot can be guided using a hand guiding device. The hand guiding device is a device equipped with an enabling device and which is required for the manual guidance of the robot.

Manual guidance mode can be switched on in the application using the motion command `handGuiding()`. Manual guidance begins at the actual position which was reached before the mode was switched on.

(>>> [15.9 "Programming manual guidance" Page 398](#))

In Manual guidance mode, the robot reacts compliantly to outside forces and can be manually guided to any point in the Cartesian space. The impedance parameters are automatically set when the robot is switched to Manual guidance mode. The impedance parameters for manual guidance cannot be modified.

A manual guidance motion command can only be executed by an application in Automatic mode. If the application is paused in Manual guidance mode, e.g. because of a safety stop triggered by an EMERGENCY STOP, the manual guidance motion is terminated. When the application is resumed, the next motion command is executed directly.

Precondition

- Hand guiding device with enabling device
- On the **Software** tab of the station configuration, the catalog elements **Human Robot Collaboration** and **Handguiding Support** are selected (check box activated) and have been installed in the project.

- A *Hand guiding device enabling inactive* is configured in the safety configuration and monitors the enabling switch on the hand guiding device. If the enabling switch on the hand guiding device is pressed, the robot can be guided manually.
- Automatic mode
- The safety equipment must be HRC-compliant.



In Manual guidance mode, incorrectly selected parameters (e.g. incorrect load data, incorrect tool), incorrect information (e.g. from defective torque sensors) or additional overlaid forces can be interpreted as external forces. This can result in unpredictable robot motions.



If the signal for manual guidance is issued before manual guidance mode is switched on in the application, manual guidance mode will be active as soon as it is switched on. This means that motion execution is not paused when the mode is switched on, making for a smooth transition between application mode and manual guidance mode.

Precondition for this response: the application velocity is less than the maximum permissible velocity configured for manual guidance.

(>>> [13.14.5.3 "Velocity monitoring during manual guidance" Page 280](#))

If the application is executed at a higher velocity, the application is paused before switching to manual guidance mode. (Then release the enabling switch, press the Start key and wait until the application is paused again.)

14.8 Approximate positioning

Approximate positioning means that the motion does not stop exactly at the end point of the programmed motion, allowing continuous robot motion. During motion programming, different parameters can influence the approximate positioning.

The point at which the original path is left and the approximate positioning arc begins is referred to as the approximate positioning point.



To approximate motions without exact positioning, they must be executed asynchronously or grouped in a MotionBatch.

(>>> [15.6.1 "Synchronous and asynchronous motion execution" Page 386](#))

(>>> [15.6.6 "MotionBatch" Page 390](#))



In the case of approximate positioning of motions executed synchronously, an exact positioning point is executed at the start of the approximate positioning arc. This also applies, in the case of synchronous execution, to the last motion within a MotionBatch.

PTP motion

The TCP leaves the path that would lead directly to the end point and moves, instead, along a path that allows it to pass the end point without exact positioning. The path thus goes past the point and no longer passes through it.

During programming, the relative maximum distance from the end point at which the TCP may deviate from its original path in axis space is defined. A relative distance of 100% corresponds to the entire path from the start point to the end point of the motion.

The approximation contour executed by the TCP is not necessarily the shorter path in Cartesian space. The approximated point can thus also be located within the approximate positioning arc.

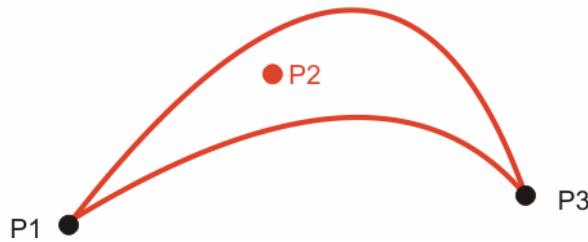


Fig. 14-14: PTP motion, P2 is approximated

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.

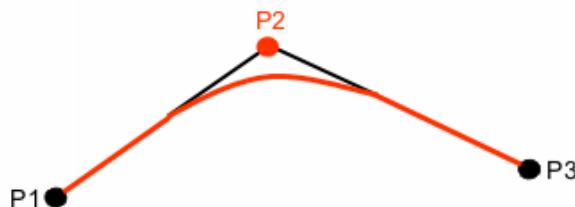


Fig. 14-15: LIN motion, P2 is approximated

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 auxiliary point may fall within the approximate positioning range and not be passed through exactly. This is dependent on the position of the auxiliary point and the programmed approximation parameters.

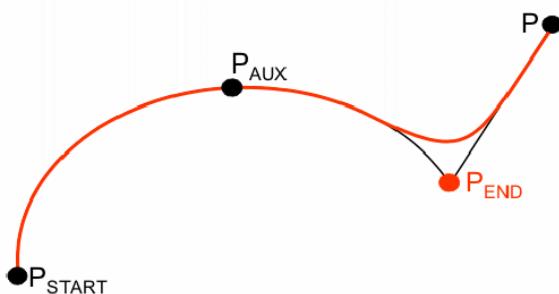


Fig. 14-16: CIRC motion, P_{END} is approximated

All spline blocks and all individual motions can be approximated with one another. It makes no difference whether they are CP or JP 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 PTP-LIN approximation, for example, the approximate positioning arc is of type CP.

If a spline block is approximated, the entire last segment is approximated. If the spline block only consists of one segment, a maximum of half the segment is approximated (this also applies for PTP, LIN and CIRC).

Approximate positioning not possible due to time:

If approximation is not possible due to delayed motion commanding, the robot waits at the start of the approximate positioning arc. The robot moves again as soon as it has been possible to plan the next block. The robot then executes the approximate positioning arc. Approximate positioning is thus technically possible; it is merely delayed.

No approximate positioning in Step mode:

In Step mode, 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 in relation to the path in standard mode.

In all other segments of both spline blocks, the path is identical in both program run modes.

Approximated motions which were sent to the robot controller asynchronously before Step mode was activated and which are waiting there to be executed will stop at the approximate positioning point. For these motions, the approximate positioning arc will be executed when the program is resumed.

14.9 Orientation control with LIN, CIRC, SPL

Description

The orientation of the TCP can be different at the start point and end point of a motion. During motion programming, it is possible to define how to deal with the different orientations.

Orientation control is set as a motion parameter by the `setOrientationType(...)` method. Orientation control is a value of type `Enum SplineOrientationType`.

Orientation control	Description
Constant	<p>The orientation of the TCP remains constant during the motion.</p> <p>The orientation of the start point is retained. The orientation of the end point is not taken into consideration.</p>
Ignore	<p>The orientation of the TCP changes during the motion.</p> <p>This option is only available for individual spline segments, not for the entire spline block or individual motions. The controller calculates the orientation control on the basis of the orientation of the surrounding control points, unless their orientation is also ignored.</p> <p>Ignore is used if no specific orientation is required for a spline segment.</p> <p>(>>> <i>"Ignore" Page 358</i>)</p> <p>Note: In the case of Ignore, the orientation of the end point is not taken into consideration. If it is important for the taught orientation to be maintained at the end point, e.g. to avoid collisions, Ignore must not be used.</p>

Orientation control	Description
OriJoint	<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>Note: Use OriJoint if, with VariableOrientation, the robot passes through a wrist axis singularity. The orientation of the TCP changes continuously during the motion, but not uniformly. OriJoint is thus not suitable if a specific orientation must be maintained exactly, e.g. in the case of laser welding.</p>
VariableOrientation	<p>During the motion, a continuous transition of the orientation of the TCP occurs from the orientation of the start point to the orientation of the end point.</p> <p>If the orientation control is not set, this orientation control is applied as standard.</p>

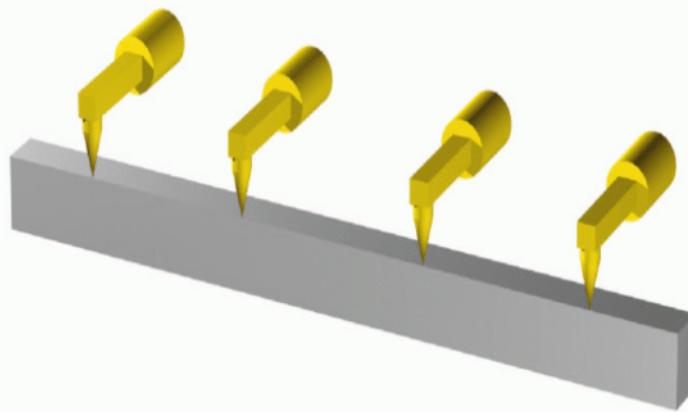


Fig. 14-17: Constant orientation (constant)

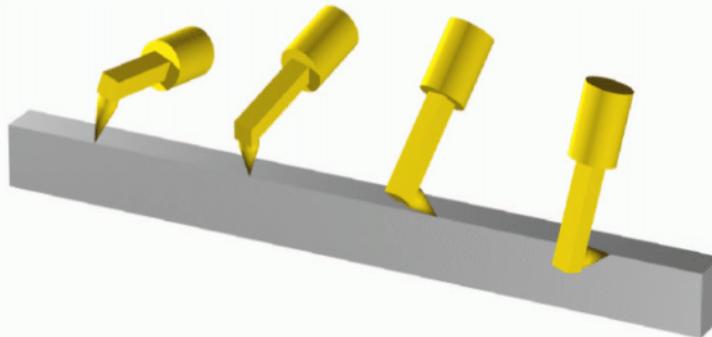


Fig. 14-18: Variable orientation (VariableOrientation or OriJoint)

CIRC motion

It is possible to define for CIRC motions whether the orientation control is to be space-related or path-related.

(>>> [14.9.1 "Orientation control reference system for CIRC" Page 358](#))

During CIRC motions, the robot controller only takes the orientation of the end point into consideration. It is possible to define 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.

Ignore

The orientation type SplineOrientationType.Ignore is used if no specific orientation is required at a point. 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:

```
robot.move(P0);
Spline path6 = new Spline(
    spl(P1),
    spl(P2),
    spl(P3).setOrientationType(SplineOrientationType.Ignore),
    spl(P4).setOrientationType(SplineOrientationType.Ignore),
    spl(P5),
    spl(P6)
);
// ...
robot.move(path6);
```

The taught or programmed orientation of P3 and P4 is ignored.

SplineOrientationType.Ignore is not allowed for the following spline segments:

- The first and last segment in a spline block
- CIRC segments with OrientationReferenceSystem.Path
- Segments followed by a CIRC segment with OrientationReferenceSystem.Path
- Segments followed by a segment with SplineOrientationType.Constant
- Successive segments in a spline block with the same end point

14.9.1 Orientation control reference system for CIRC

Description

It is possible to define for CIRC motions whether the orientation control is to be space-related or path-related.

The reference system of the orientation control is set as a motion parameter by the setOrientationReferenceSystem(...) method. Orientation control is a value of type Enum OrientationReferenceSystem.

The reference system of the orientation control can only be specified before the orientation type.

Reference system	Description
Base	Base-related orientation control during the circular motion
Path	Path-related orientation control during the circular motion

Example

Path-related circular motion with constant orientation:

```
robot.move(circ(P6, P7)
    .setOrientationReferenceSystem(
        OrientationReferenceSystem.Path)
```

```
.setOrientationType(SplineOrientationType.Constant));
```

Limitation

OrientationReferenceSystem.Path is not allowed for the following spline segments:

- CIRC segments with SplineOrientationType.Ignore
- CIRC segments preceded by a segment with SplineOrientationType.Ignore

14.9.2 Combination of reference system and orientation type for CIRC



If the reference system of the orientation control is combined with SplineOrientationType.OriJoint, the reference system has no influence on the orientation control.

Path-related circular motion with constant orientation:

- OrientationReferenceSystem.Path
- SplineOrientationType.Constant

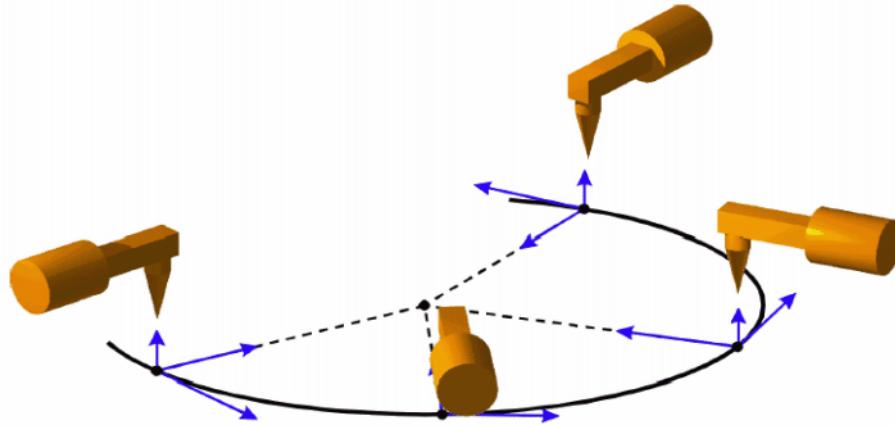


Fig. 14-19: Constant orientation, path-related

Path-related circular motion with variable orientation:

- OrientationReferenceSystem.Path
- SplineOrientationType.VariableOrientation

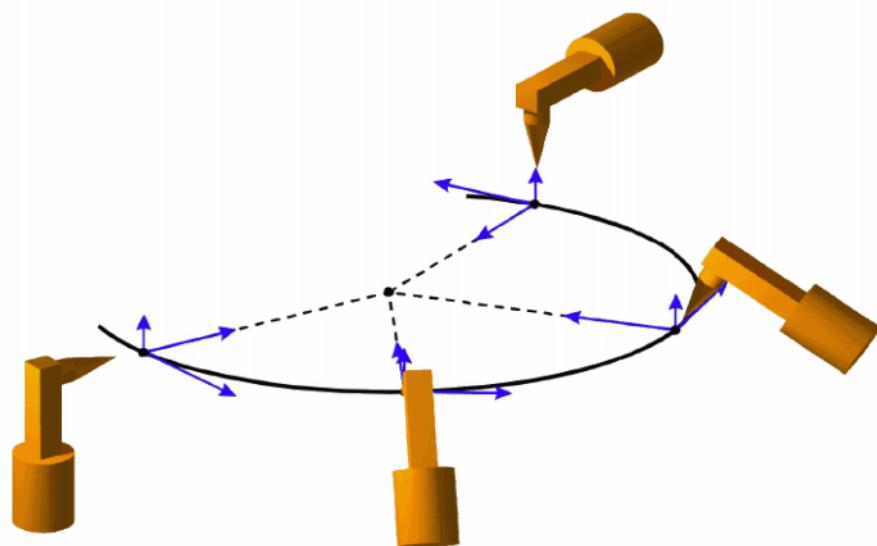


Fig. 14-20: Variable orientation, path-related

Base-related circular motion with constant orientation:

- OrientationReferenceSystem.Base
- SplineOrientationType.Constant

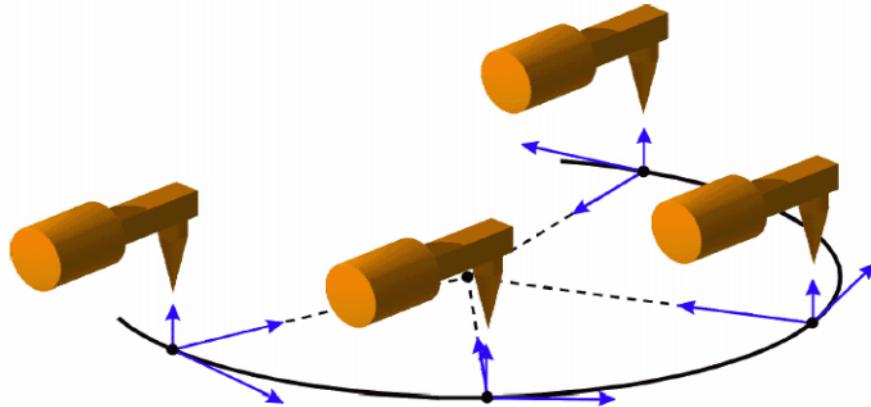


Fig. 14-21: Constant orientation, base-related

Base-related circular motion with variable orientation:

- OrientationReferenceSystem.Base
- SplineOrientationType.VariableOrientation

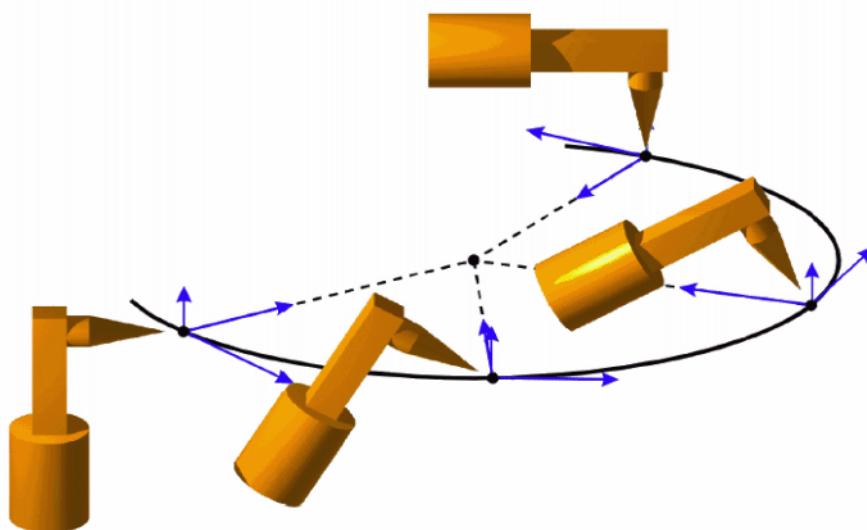


Fig. 14-22: Variable orientation, base-related

14.10 Redundancy information

For a given axis position of a robot, the resulting point in Cartesian space at which the TCP is located is unambiguously defined. Conversely, however, the axis position of the robot cannot be unambiguously determined from the Cartesian position X, Y, Z and orientation A, B, C of the TCP. A Cartesian point can be reached with multiple axis configurations. In order to determine an unambiguous configuration, the Status parameter must be specified.

Robots with 6 axes already have ambiguous axis positions for a given Cartesian point. With its additional 7th axis, an LBR is able to reach a given position and orientation with a theoretically unlimited number of axis poses. To unambiguously determine the axis pose for an LBR, the redundancy angle must be specified in addition to the Status.

The Turn parameter is required for axes which can exceed the angle $\pm 180^\circ$. In PTP motions, this helps to unambiguously define the direction of rotation of the axes. Turn has no influence on CP motions.

Status, Turn and the redundancy angle are saved during the teaching of a frame. They are managed as arrays of the data type AbstractFrame.

Programming

The Status of a frame is only taken into account in PTP motions to this frame. With CP motions, the Status given by the axis configuration at the start of the motion is used.

In order to avoid an unpredictable motion at the start of an application and to define an unambiguous axis configuration, it is advisable to program the first motion in an application with one of the following instructions: The axis configuration should not be in the vicinity of a singular axis position.

- PTP motion to a specified axis configuration with specification of all axis values:

```
ptp(double a1, double a2, double a3, double a4, double a5, double a6, double a7)
```

- PTP motion to a specified axis configuration:

```
ptp(JointPosition joints)
```

- PTP motion to a taught frame (AbstractFrame type):


```
ptp(getApplicationContext().getFrame(String frameName));
```

14.10.1 Redundancy angle

With its 7th axis, an LBR is able to reach a point in space with a theoretically unlimited number of different axis configurations. An unambiguous pose is defined via the redundancy angle.

In an LBR, the redundancy angle has the value of the 3rd axis.

The following applies for all motions:

- The redundancy angle of the end frame is taken into account when the robot that was used when teaching the frame also executes the motion command. In particular, the robot name defined in the station configuration must match the device specified in the frame properties.
- If the robots do not match or if calculated frames are used, the redundancy angle given at the start of motion by the axis configuration is retained.

14.10.2 Status

The Status specification prevents ambiguous axis positions. The Status is described by a binary number with 3 bits.

Bit 0

Specifies the position of the wrist root point (intersection of axes A5, A6, A7) with reference to the X axis of the coordinate system of axis A1. The alignment of the A1 coordinate system is identical to the robot base coordinate system if axis A1 is at 0°. It moves with axis A1.

Position	Value
Overhead area The robot is in the overhead area if the X value of the position of the wrist root point, relative to the A1 coordinate system, is negative.	Bit 0 = 1
Basic area The robot is in the basic area if the X value of the position of the wrist root point, relative to the A1 coordinate system, is positive.	Bit 0 = 0

Bit 1

Specifies the position of axis A4.

Position	Value
A4 < 0°	Bit 1 = 1
A4 ≥ 0°	Bit 1 = 0

Bit 2

Specifies the position of axis A6.

Position	Value
A6 < 0°	Bit 2 = 1
A6 ≥ 0°	Bit 2 = 0

The following applies for PTP motions:

- The Status of the end frame is taken into account when the robot which was used when teaching the frame also executes the motion command. In particular, the robot name defined in the station configuration must match the device specified in the frame properties.
- If the robots do not match or if calculated frames are used, the Status given by the axis configuration at the start of the motion is retained.

The following applies for CP motions:

- The Status of the end frame is not taken into account. The Status given by the axis configuration at the start of the motion is retained.
- **Exception:** A change of Status is possible if the end frame is addressed with the `SplineOrientationType.ORI_JOINT` orientation control. The Status of the end frame is not taken into consideration in this case either. The Status at the end of the motion is determined by the path planning, which selects the shortest route to the end frame.

14.10.3 Turn

The Turn specification makes it possible to move axes through angles greater than $+180^\circ$ or less than -180° without the need for special motion strategies (e.g. auxiliary points). The Turn is specified by a binary number with 7 bits.

With rotational axes, the individual bits determine the sign before the axis value in the following way:

Bit = 0: Angle $\geq 0^\circ$

Bit = 1: Angle $< 0^\circ$

Value	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A7 $\geq 0^\circ$	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	A7 $< 0^\circ$	A6 $< 0^\circ$	A5 $< 0^\circ$	A4 $< 0^\circ$	A3 $< 0^\circ$	A2 $< 0^\circ$	A1 $< 0^\circ$

The Turn is not taken into account in an LBR because none of its axes can rotate over $\pm 180^\circ$.

14.11 Singularities

Due to the axis position, Cartesian motions of the robot may be limited. Due to the combination of axis positions of the entire robot, no motions can be transferred from the drives to the flange (or to an object on the flange, e.g. a tool) in at least one Cartesian direction. In this case, or if very slight Cartesian changes require very large changes to the axis angles, one speaks of singularity positions.



It is advisable to move the robot as slowly as possible near singularities.



No Cartesian motions may be started in the vicinity of singularities. This also includes the default HOME position of the robot, for example.

14.11.1 Kinematic singularities

The flexibility due to the redundancy of a 7-axis robot, in contrast to the 6-axis robot, requires 2 or more kinematic conditions (e.g. extended position, 2 rotational axes coincide) to be active at the same time in order to reach a singularity position. There are 4 different robot positions in which flange motion in one Cartesian direction is no longer possible. Here only

the position of 1 or 2 axes is important in each case. The other axes can take any position.

A4 singularity

This kinematic singularity is given when $A4 = 0^\circ$. It is called the extended position.

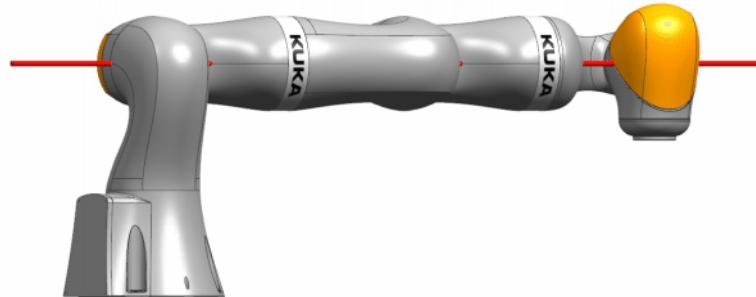


Fig. 14-23: Extended position $A4 = 0^\circ$

In this example: Motion is blocked in the direction of the robot base or parallel to axis A3 or A5.

A2/A6 singularity

This kinematic singularity is given when $A2 = 0^\circ$ and $A6 = 0^\circ$.

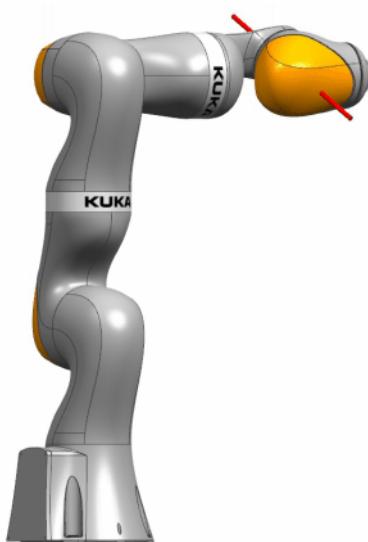


Fig. 14-24: $A2 = 0^\circ$ and $A6 = 0^\circ$

In this example: Motion parallel to axis A6 or A2 is blocked.

A2/A3 singularity

This kinematic singularity is given when $A2 = 0^\circ$ and $A3 = \pm 90^\circ$ ($\pi/2$).

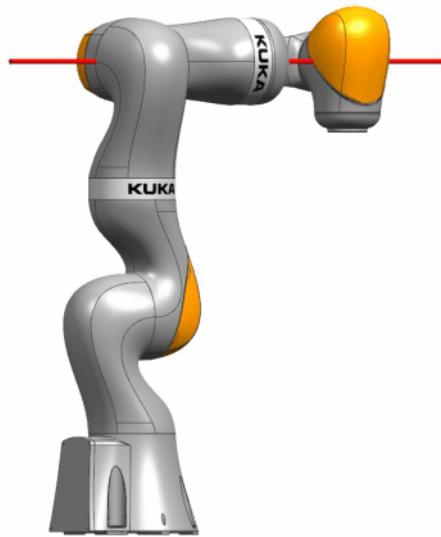


Fig. 14-25: $A_2 = 0^\circ$ and $A_3 = \pm 90^\circ (\pi/2)$

In this example: Motion is blocked in the direction of the robot or parallel to axis A5.

A5/A6 singularity

This kinematic singularity is given when $A_5 = \pm 90^\circ (\pi/2)$ and $A_6 = 0^\circ$.



Fig. 14-26: $A_5 = \pm 90^\circ (\pi/2)$ and $A_6 = 0^\circ$

In this example: Motion parallel to axis A3 is blocked.

14.11.2 System-dependent singularities

Description

The redundant configuration of the LBR with its 7th axis allows the robot arm to move without the flange moving. In this null space motion, all axes move except A4, the “elbow axis”. In addition to the normal redundancy, it is possible, under certain circumstances, that only subchains of the robot can move and not all axes.

All of the robot positions in this category have in common that slight Cartesian changes result in very large changes to the axis angles. They are very similar to the singularities in 6-axis robots since, in the LBR too, a division is made into the position part and orientation part of the wrist root point.

Wrist axis singularity

Wrist axis singularity means the axis position $A_6 = 0^\circ$. The position of axes A_5 and A_7 can thus no longer be resolved. There are an infinite number of ways to position these two axes to generate the same position on the flange.

A1 singularity

If the wrist root point is directly over A_1 , no reference value can be specified for the redundancy circle according to the definition above. The reason for this is that any A_1 value is permissible here for $A_3 = 0^\circ$.

Every axis position of A_1 can be compensated for with a combination of A_5 , A_6 and A_7 so that the flange position remains unchanged.

A2 singularity

With an extended “shoulder”, the position of axes A_1 and A_3 can no longer be resolved according to the pattern above.

A2/A4 singularity

If A_1 and A_7 coincide, the position of axes A_1 and A_7 can no longer be resolved according to the pattern above.



System-dependent singularities can be avoided in most cases by a suitable elbow position.

15 Programming

15.1 Java Editor

15.1.1 Opening a robot application in the Java Editor

Description

The Java Editor allows more than one file to be open simultaneously. If required, they can be displayed side by side or one above the other. This provides a convenient way of comparing contents, for example.

Precondition

- Robot application has been created.

Procedure

- Double-click on a Java file in the **Package Explorer** view.

Alternative procedure

- Right-click on the Java file and select **Open** or **Open With > Java Editor** from the context menu.

15.1.2 Structure of a robot application

```

1 package application;
2 +
3 * Implementation of a robot application.
4 public class RobotApplication extends RoboticsAPIApplication {
5     @Inject
6     private LBR lbr_iwia_7_R800_1;
7
8     @Override
9     public void initialize() {
10         // initialize your application here
11     }
12
13     @Override
14     public void run() {
15         // your application execution starts here
16         lbr_iwia_7_R800_1.move(ptpHome());
17     }
18 }

```

Fig. 15-1: Structure of a robot application

Item	Description
1	This line contains the name of the package in which the robot application is located.
2	The import section contains the imported classes which are required for programming the robot application. Note: Clicking on the “+” icon opens the section, displaying the imported classes.
3	Header of the robot application (contains the class name of the robot application) (>>> "Header" Page 368)

Item	Description
4	<p>Declaration section</p> <p>The data arrays of the classes required in the robot application are declared here.</p> <p>When the robot application is created, instances of the necessary classes are automatically integrated by means of dependency injection. As standard, this is the instance of the robot used, here an LBR.</p> <p>(>>> 15.3.3 "Dependency injection" Page 379)</p>
5	<p>initialize() method</p> <p>In this method, initial values are assigned to data arrays that have been created in the declaration section and are not integrated using dependency injection.</p>
6	<p>run() method</p> <p>The programming of the robot application begins in this method.</p> <p>When the robot application is created, a motion instruction which moves the robot to the HOME position is automatically inserted.</p> <p>(>>> 15.15 "HOME position" Page 428)</p>

Header

In a robot application, this is the special form of Java class:

```
public class RobotApplication extends RoboticsAPIApplication
```

Element	Description
public	The keyword public designates a class which is publicly visible. Public classes can be used across all packages.
class	The keyword class designates a Java class. The name of the class is derived from the name of the application.
extends	The application is subordinate to the RoboticsAPIApplication class.

15.1.3 Edit functions

15.1.3.1 Renaming variables

Description

A variable name can be changed in a single action at all points where it occurs.

Procedure

1. Select the desired variable at any point.
2. Right-click and select **Refactor > Rename...** from the context menu.
OR: Press the keyboard shortcut ALT + SHIFT +R.
3. The variable is framed in blue and can now be edited. Change the name and confirm with the Enter key.

15.1.3.2 Auto-complete

Description

An auto-complete function is available in the Java Editor.

When entering code, it is possible to display an “Auto-complete” list containing entries that are compatible with characters which have already been entered. These entries are prioritized according to their frequency of use, i.e. the selection is dynamically adapted to the user’s actions.

An entry from the “Auto-complete” list can be inserted into the program code as needed. This makes it unnecessary to retype the complex syntax of methods, for example. All that is then required is to enter the variable elements in the syntax manually.

Procedure

1. Begin typing the code.



When entering a dot operator for a data array or enum, the “Auto-complete” list is automatically displayed. The list contains the following entries:

- Available methods of the corresponding class (only for data arrays)
- Available constants of the corresponding class

2. Press CTRL + space bar. The “Auto-complete” list containing the available entries is displayed.



If the list contains only one matching entry, this can automatically be inserted into the program code by pressing CTRL + space bar.

3. Select the appropriate entry from the list and press the Enter key. The entry is inserted in the program code.

If an entry is selected, the Javadoc information on this entry is displayed automatically.

(>>> [15.1.4 "Displaying Javadoc information" Page 371](#))

4. Complete the syntax if necessary.

Navigating and filtering

There are various ways to navigate to the “Auto-complete” list and to filter the available entries:

- Use the arrow keys on the keyboard to move from one entry to the next (up or down)
- Scrolling
- Complete the entered code with additional characters. The list is filtered and only the entries which correspond to the characters are displayed.
- Press CTRL + space bar. Only the available template suggestions are displayed.

15.1.3.3 Templates – Fast entry of Java statements

Description

Templates for fast entry are available for common Java statements, e.g. FOR loops.

Procedure

1. Begin typing the code.
2. Press CTRL + space bar (twice). A list of the template suggestions that are compatible with the characters already entered is displayed.
3. Accept the instruction with the Enter key. Or double-click on a different instruction.
4. Complete the syntax.

Alternative procedure

Selecting templates in the **Templates** view:

1. Select the menu sequence **Window > Show View > Other...**. The **Show View** window opens.
2. In the **General** folder, select **Templates**. Confirm with **OK**. The **Templates** view opens.
3. Position the cursor in the line in which the code template is to be inserted.
4. Double-click on the desired Java instruction in the **Templates** view. The code is inserted in the editor.
5. Complete the syntax.

15.1.3.4 Creating user-specific templates**Description**

User-specific templates can be created, e.g. templates for motion blocks with specific motion parameters which are used frequently during programming.

Procedure

1. In the Templates view, select the context in which the template is to be inserted.
2. Right-click on the context and select **New...** from the context menu. or: Click on the **Create a New Template** icon. The **New Template** window opens.
3. Enter a name for the template in the **Name** box.
4. Enter a description in the **Description** box (optional).
5. In the **Pattern** box, enter the desired code.
6. Confirm the template properties with **OK**. The template is created and inserted into the Templates view.

15.1.3.5 Extracting methods**Description**

Parts of the program code can be extracted from the robot application and made available as a separate method. This makes particular sense for frequently recurring tasks, as it increases clarity within the robot application.

Procedure

1. Select the desired program code.
2. Right-click in the editor area.
3. Select **Refactor > Extract Method...** from the context menu.

OR: Press the keyboard shortcut ALT + SHIFT +M.

The **Extract method** window opens.

4. Enter a unique method name and select the desired **Access modifier**. Confirm with **OK**.

The method is generated and the selected program code is inserted into this method. At all other points within the class where the extracted code excerpt additionally occurs, it is likewise replaced with the method call.

Access modifier

This option defines which classes can call the extracted method.

Option	Description
private	This method can only be called by the corresponding class itself.
default	The following classes can call the method: <ul style="list-style-type: none"> • The corresponding class • The inner classes of the corresponding class • All classes of the package in which the corresponding class is located
protected	The following classes can call the method: <ul style="list-style-type: none"> • The corresponding class • The subclasses of the corresponding class (inheritance) • The inner classes of the corresponding class • All classes of the package in which the corresponding class is located
public	All classes can call the method, regardless of the relationship to the corresponding class and of the package assignment.

15.1.4 Displaying Javadoc information

Description

Javadoc is a documentation generated from specific Java comments. The functionalities and use of classes, methods and libraries are described in Javadoc.

The Javadoc information can be displayed during programming. The information is only available in English.

The various display functions are described using the example of the LBR class.

Procedure

Displaying Javadoc information in auto-complete:

1. In auto-complete (CTRL + space bar), select an entry in the “Auto-complete” list. The associated Javadoc information is displayed in a separate window in the editor area.

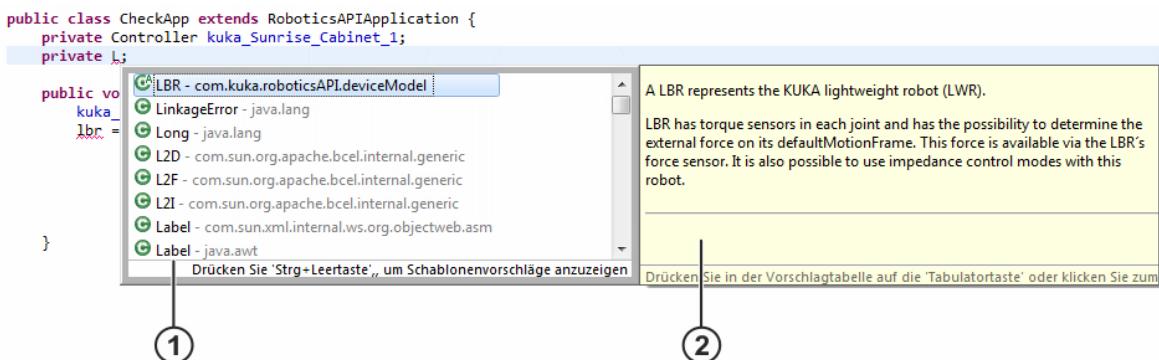


Fig. 15-2: Displaying Javadoc information in auto-complete

1 "Auto-complete" list

2 Javadoc information

2. In order to pin the window in the editor area, press the tab key or click inside the window.
Pinning the window makes it possible to navigate to the Javadoc description, e.g. by scrolling.

Displaying Javadoc information using the mouse pointer:

- Move the mouse pointer to the desired element name in the program code. The associated Javadoc information is automatically displayed in a window in the editor area.

The following elements react to the mouse pointer:

- Methods
- Classes (data types, not user-defined data arrays)
- Interfaces
- Enums

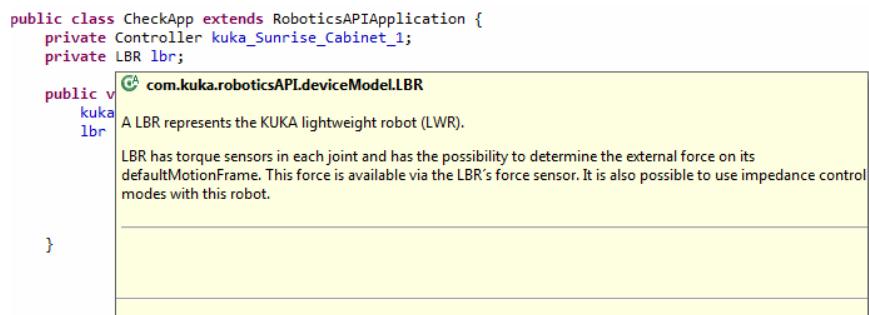


Fig. 15-3: Displaying Javadoc information using the mouse pointer

- Further display options are available from here:
 - In order to be able to navigate to the Javadoc description, e.g. by scrolling, move the mouse pointer in the window.
The window is not pinned. If the mouse pointer is moved out of the window, the window closes.
 - In order to pin the window in the editor area, press the F2 key or click inside the window.
It is also possible to navigate to the Javadoc description in the pinned window.
 - To additionally display the Javadoc information in the **Javadoc** view, left-click on the selected element.

If the window is not pinned in the editor area, it is closed.

Navigation

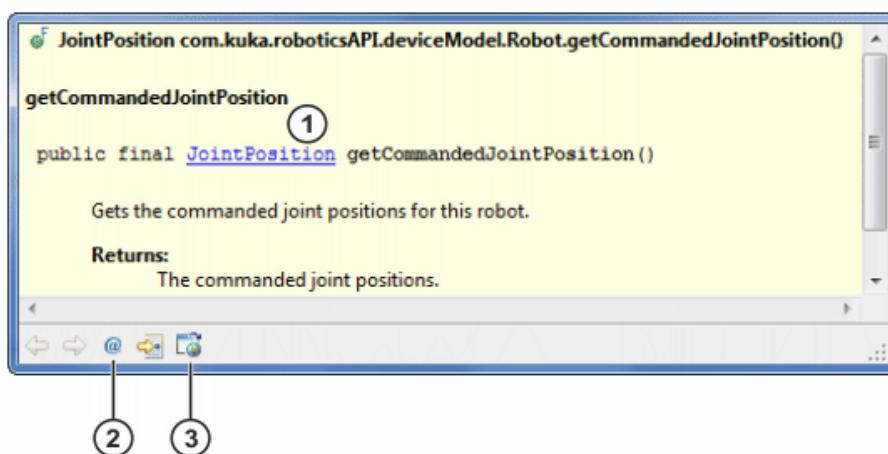


Fig. 15-4: Navigating to the Javadoc description

Item	Description
1	<p>Linked class</p> <p>Left-clicking on the linked class displays the complete Javadoc information relating to this class in the Javadoc browser.</p> <p>Note: If the corresponding link in the Javadoc view is selected, the complete Javadoc information is displayed in the view itself.</p>
2	<p>Show in Javadoc View button</p> <p>The window in the editor section closes and the Javadoc information is displayed in the Javadoc view.</p>
3	<p>Open Attached Javadoc Browser button</p> <p>The window in the editor section closes and the complete Javadoc information relating to the corresponding class is displayed in the Javadoc browser.</p>



There is a further option for displaying the complete Javadoc information on a specific element in the Javadoc browser: Select the desired element in the program code and press SHIFT + F2.

15.1.4.1 Configuration of the Javadoc browser

The configuration of the Javadoc browser is described briefly using the example of the LBR class.

① Overview Package Class Tree Deprecated Index Help
[PREV CLASS](#) [NEXT CLASS](#)
 SUMMARY: NESTED | FIELD | CONSTR | METHOD
[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: FIELD | CONSTR | METHOD

com.kuka.roboticsAPI.deviceModel
Class LBR

```
java.lang.Object
  ↘ com.kuka.common.TaggableObject
    ↘ com.kuka.roboticsAPI.geometricModel.SceneGraphObject
      ↘ com.kuka.roboticsAPI.geometricModel.SpatialObject
        ↘ com.kuka.roboticsAPI.geometricModel.PhysicalObject
          ↘ com.kuka.roboticsAPI.deviceModel.Device
            ↘ com.kuka.roboticsAPI.deviceModel.Robot
              ↘ com.kuka.roboticsAPI.deviceModel.LBR
```

All Implemented Interfaces:
[com.kuka.common.ITaggable](#), [IOperationModeProvider](#)

Direct Known Subclasses:
[SunriseLBR](#)

```
public abstract class LBR
  extends Robot
```

③ A LBR represents the KUKA lightweight robot (LWR).

LBR has torque sensors in each joint and has the possibility to determine the external force on its defaultMotionFrame. This force is available via the LBR's force sensor. It is also possible to use impedance control modes with this robot.

Field Summary

protected	_gmsSensorLimits
double[]	torque sensor limits.

Fields inherited from class [com.kuka.roboticsAPI.deviceModel.Device](#)

```
hardwareVersion
```

④ Constructor Summary

```
LBR(Controller controller, java.lang.String name)
Creates a new LBR instance with the given controller.
```

Method Summary

boolean	checkTorqueSensor(JointEnum joint)
Checks if the torque sensor of the given axis works properly.	

Field Detail

[_gmsSensorLimits](#)
 protected double[] [_gmsSensorLimits](#)
 torque sensor limits.

Constructor Detail

LBR

```
public LBR(Controller controller,
           java.lang.String name)
```

⑤ Creates a new LBR instance with the given controller.

Parameters:

controller - The controller to which this device belongs.
 name - The name of the device.

Method Detail

initializeJointCount

```
protected int initializeJointCount()
```

Fig. 15-5: Configuration of the Javadoc browser

Item	Description
1	Navigation
2	<p>Class hierarchy (>>> <i>Fig. 15-6</i>)</p> <p>The inheritance relationships of the class are displayed here.</p>
3	<p>Description of the class</p> <p>The task of the class and its functionality is described here. Special aspects of using the class are normally indicated in this area. It may also contain short examples for using the class.</p> <p>The earliest library version in which the class is available is normally specified at the end of the description. The description may additionally contain a list of references to further classes or methods which may be of interest.</p>
4	<p>Oversviews</p> <ul style="list-style-type: none"> • Field Summary Overview of the data fields which belong to the class The data fields inherited from a parent class are listed here. • Constructor Summary Overview of the constructors which belong to the class • Method Summary Overview of the methods which belong to the class The methods inherited from a parent class are listed here. <p>The overviews contain short descriptions of the data fields, constructors and methods of the class, provided that these were specified during the creation of Javadoc. Inherited data fields and methods are only listed.</p> <p>Detailed descriptions on the data fields, constructors and methods can be found in the Details area. Click on the respective name to directly access the detailed description.</p>
5	<p>Details</p> <ul style="list-style-type: none"> • Field Detail Detailed description of the data fields which belong to the class • Constructor Detail Detailed description of the constructors which belong to the class • Method Detail Detailed description of the methods which belong to the class <p>The detailed description may, for example, contain a list and description of the transferred parameters and return value. Provided there are any, the exceptions which may occur when executing a method or constructor are also named here.</p>

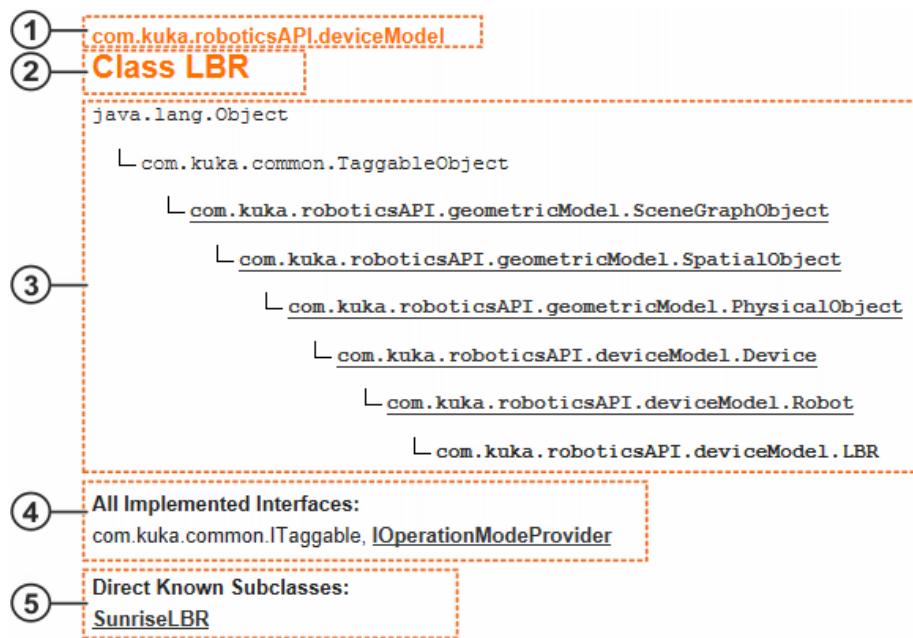


Fig. 15-6: Class hierarchy

Item	Description
1	Name of the package to which the class belongs
2	Name of the class
3	Class hierarchy (parentage of the class)
4	List of interfaces implemented by the class
5	List of subclasses derived from the class

15.2 Symbols and fonts

The following symbols and fonts are used in the syntax descriptions:

Syntax element	Appearance
Java code	<ul style="list-style-type: none"> Courier font Upper/lower-case letters <p>Examples: <code>private</code>; <code>new</code>; <code>linRel</code>; <code>Tool</code></p>
Elements that must be replaced by program-specific entries	<ul style="list-style-type: none"> Italics Upper/lower-case letters <p>Examples: <code>endpoint</code>; <code>name</code>; <code>mode</code></p>
Optional elements	<ul style="list-style-type: none"> In angle brackets <p>Example: <code><.setVelocity(<i>value</i>)></code></p>
Elements that are mutually exclusive	<ul style="list-style-type: none"> Separated by the “ ” symbol <p>Example: <code>++ --</code></p>

15.3 Data types

There are 2 kinds of data type in Java:

- Primitive data types

- Complex data types

Complex data types are defined in Java by classes.

Data types that are frequently required for robot programming are pre-defined in the KUKA RoboticsAPI.

Overview of important data types:

Data type	Description
int	Integer • -2³¹ ... +2³¹-1 Examples: -1; 32; 8000
double	Double-precision floating-point number • -1.7E+308 ... +1.7E+308 Examples: 1.25; -98.76; 123.456
boolean	Logic state • true • false
char	Character (1 character) • ASCII character Examples: 'A'; '1'; 'q'
String	Character string • ASCII character Examples: "KUKA"; "tool"



The names of the primitive data types are displayed in violet in the Java Editor.

15.3.1 Declaration

Description

To allow programming in Java, the necessary objects must first be created (declared), i.e. the data type and identifier must be defined.

Syntax

Data type Name;

Explanation of the syntax

Element	Description
<i>Data type</i>	Data type of the variable
<i>Name</i>	Name of the variable

Examples

```
int counter;
double value;
Controller kuka_Sunrise_Cabinet;
ForceCondition contactForceReached;
```

15.3.2 Initialization

Before an object can be used in the program, it must be assigned an initial value.



Primitive data types are automatically assigned a default value when they are created. The initial value depends on the data type.

15.3.2.1 Primitive data types

Description

In the case of primitive data types, the assignment is done by the operator “=” followed by the desired value.

Primitive data types can be created and used in the run() method of an application, for example.

Example

The variables `a` and `b` are created in an application and assigned an initial value. Subsequently, the variable `c` is created and assigned the sum of the variables `a` and `b`.

```
@Override
public void run() {
    // ...
    int a = 3;
    int b = 5;
    // ...
    int c = a + b;
    // ...
}
```

15.3.2.2 Complex data types

Description

Complex data types are always instanced by the call of a constructor in conjunction with the keyword `new`. The instancing can take place either directly or within a method that supplies an object of the data type as the return value.

Depending on the specific implementation of the associated class, parameters for the first instancing can be transferred to the constructor if required.

Further values are assigned to the properties by the methods provided by the class.

In robot applications, complex data types are usually created after the header and initialized in the initialize() method.

Example

In an application, data arrays for a Cartesian impedance mode and a force break condition are created and initialized.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    // ...
    private CartesianImpedanceControlMode softInToolX;
    private ForceCondition contactForceReached;
```

```

@Override
public void initialize() {
    softInToolX = new CartesianImpedanceControlMode();
    softInToolX.setDampingToDefaultValue();
    // ...
    contactForceReached =
        ForceCondition.createSpatialForceCondition(...);
}

@Override
public void run() {
    // ...
    robot.move(ptp(getFrame("/P20"))
        .breakWhen(contactForceReached));
}
}

```

15.3.3 Dependency injection

Description

With the aid of dependency injection, it is no longer necessary to actually generate instances of certain object types. It is sufficient to provide the points where the objects are to be used with an appropriate annotation so that the runtime system performs the generation. This allows an application that is based on multiple Java classes to access common objects without having to transfer the objects to the classes in each case.

Dependency injection can only be used in classes that are themselves generated by dependency injection. If such a class is instanced with `new`, the corresponding points remain non-initialized (“null”). As the runtime system generates robot and background applications with dependency injection, the function can be used there.

Syntax

```
@Inject
<Modifier> Data type Name;
```

Explanation of the syntax

Element	Description
<code>@Inject</code>	Annotation for initializing the array of type <i>Data type</i> with dependency injection.
<i>Modifier</i>	If required, valid modifiers can be used here for the array declaration, e.g.: <ul style="list-style-type: none"> • <code>public</code>, <code>private</code>, <code>protected</code>, etc. The modifier <code>static</code> cannot be used for arrays with <code>@Inject</code> and <code>final</code> should also be avoided.
<i>Data type</i>	Data type of the array
<i>Name</i>	Name of the array

Example

Injection and use of an array

```
@Inject
private InjectableClass myField;

public void myMethod() {
    myField.doSomething();
}
```



Constructor and method injection are also possible in addition to array injection. These are described in greater detail in the documentation of the **Guice** injection library, for example.

15.3.3.1 Dependency injection for Sunrise types

Description

The most important types in Sunrise can be integrated using dependency injection. This applies to the following types, among others:

- Controller
- Robot
- LBR
- Tool
- Workpiece
- ITaskLogger
- IStatusController
- IApplicationData
- All generated I/O groups
- All classes created in Sunrise.Workbench which are derived from Tool or Workpiece and have been configured as **Class of Template** in the properties of an object template.



Other classes and interfaces in addition to those described here can also be integrated using dependency injection. A list of these objects can be found in the Javadoc of UseRoboticsAPIContext.

Procedure

To call Javadoc of UseRoboticsAPIContext:

1. Move the mouse pointer over RoboticsAPIApplication in the header of an application.
2. A window opens. In it, click on the link **Available type for Dependency Injection**.
3. The Javadoc file is displayed in the editor area.



If object templates are to be integrated using dependency injection, the annotation `@Named("TemplateName")` must additionally be used. The name of the object template as configured in the **Object templates** view must be entered as the `TemplateName`.

Examples

An LBR iiwa and a gripper are integrated in a robot application by means of dependency injection. An object template with the name "Gripper" has been created for the gripper. The gripper is attached to the robot during initialization. Motions with both devices are executed in the application.

In addition, a logger object is integrated which is used to display LOG information of the smartHMI.

```

public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private ITaskLogger logger;
    @Inject
    private IApplicationData data;
    @Inject
    private LBR robot;
    @Inject
    @Named("Gripper")
    private Tool gripper;

    @Override
    public void initialize() {
        // initialize your application here
        gripper.attachTo(robot.getFlange());
        logger.info("Application initialized!");
    }

    @Override
    public void run() {
        // your application execution starts here
        robot.move(ptpHome());
        robot.move(ptp(data.getFrame("/Start")));
        // ...
        logger.info("Move gripper");
        gripper.move(linRel().setXOffset(25.0));
        // ...
    }
}

```

The signals of an I/O group are to be used in both the robot application and a background application.

Use in robot application:

```

public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private LEDsIOGroup appLEDs;

    @Override
    public void initialize() {
        // initialize your application here
    }

    @Override
    public void run() {
        // your application execution starts here
        // ...
        appLEDs.setBlueLight(true);
        robot.move(handGuiding());
        appLEDs.setBlueLight(false);
        // ...
    }
}

```

Use in background application:

```

public class MonitoringTask extends
RoboticsAPICyclicBackgroundTask {
    ...
    private boolean appRunning;
    @Inject
    private LEDsIOGroup bgtLEDs;

    @Override
    public void initialize() {
        // initialize your task here
        initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
        CycleBehavior.BestEffort);
    }

    @Override
    public void runCyclic() {
        ...
        if (appRunning) {
            // If application is running,
            // LED changes its state continuously
            bgtLEDs.setYellowLight(!bgtLEDs.getYellowLight());
        }
        else {
            // If application is not running, LED remains off
            bgtLEDs.setYellowLight(false);
        }
        ...
    }
}

```

15.3.3.2 Dependency injection for dedicated types

Description

A class can be used via dependency injection if it meets one of the following conditions:

- The class has a public constructor without parameters. An `@Inject` annotation on the constructor is not absolutely essential in this case.
- The class has a public constructor with an `@Inject` annotation which either contains no parameters or for which all parameters can be obtained via dependency injection.

All classes that are present in an application and meet the specified conditions can be integrated in all constituent parts of the application using `@Inject`. A new instance of the class is generated as standard for each integration using `@Inject`.

Singletons

If a dedicated class is provided with the annotation `@Singleton` in addition to dependency injection, this results in only one instance of this class being used in the application. This means that all objects of this class generated via dependency injection refer to the same instance.



Use of the annotation `@Singleton` enables an application to be subdivided into multiple classes which can be called from the main application.

State variables, e.g. of tool and workpiece classes, can be used by various program sections through this mechanism.

(>>> *15.10.4 "Integrating dedicated object classes with dependency injection" Page 409*)



This procedure represents an alternative to the process data, though the state variables are not automatically saved.

Example

The classes `Vehicle`, `Motor` and `Wheel` are used in a project. The classes `Motor` and `Wheel` are to be available in the `Vehicle` class via dependency injection. As a vehicle usually only has one motor (or engine), the `Motor` class is to be defined as a singleton.

2 objects of each of the classes `Motor` and `Wheel` are integrated in the `Vehicle` class. Comparison of the objects is then intended to show that the objects of the `Motor` class refer to the same instance whereas the objects of the `Wheel` class refer to different instances.

The `Vehicle` class is likewise integrated in a robot application using dependency injection. An object of the `ITaskLogger` class is integrated in both the robot application and the `Vehicle` class by means of dependency injection. Integrating the `ITaskLogger` interface via dependency injection also enables information from the `Vehicle` class to be displayed on the smartHMI.

Wheel class:

```
public class Wheel
{
    @Inject
    public Wheel() {}
    // ...
}
```

Motor class:

```
@Singleton
public class Motor
{
    @Inject
    public Motor() {}
    // ...
}
```

Vehicle class:

```
public class Vehicle {
    @Inject
    private ITaskLogger logger;
    @Inject
    private Wheel frontWheel;
    @Inject
    private Wheel rearWheel;
    @Inject
    private Motor motor;
    @Inject
    private Motor additionalMotor;
    // ...
    @Inject
    private Vehicle() {
    }

    public void setCarStatus() {
        frontWheel.setName("FrontWheel");
        rearWheel.setName("RearWheel");
        motor.setName("Motor");
    }
}
```

```

        additionalMotor.setName("AdditionalMotor");
        // ...
    }

    public void printCarStatus() {
        logger.info("*****");
        logger.info("Comparing the instances of Wheel:");
        if (frontWheel == rearWheel) {
            logger.info(frontWheel + " and " +
                        rearWheel + " are equal.");
        }
        else {
            logger.info(frontWheel + " and " +
                        rearWheel + " are NOT equal.");
        }

        logger.info("Comparing the instances of Motor:");
        if (motor == additionalMotor) {
            logger.info(motor + " and " + additionalMotor +
                        " are equal.");
        }
        else {
            logger.info(motor + " and " + additionalMotor +
                        " are NOT equal.");
        }
        logger.info("*****");
    }
    // ...
}

```

Robot application (CarApplication class):

```

public class CarApplication extends RoboticsAPIApplication {
    @Inject
    private ITaskLogger logger;
    @Inject
    private Vehicle myNewCar;

    @Override
    public void initialize() {
        myNewCar.setName("Isolde")
        // ...
    }

    @Override
    public void run() {
        logger.info("Name of vehicle:" + myNewCar.getName());
        myNewCar.setCarStatus();
        myNewCar.printCarStatus();
    }
}

```

The screenshot (>>> Fig. 15-7) shows the information displayed on the smartHMI when the robot application is executed. Besides the displays relating to the robot application it also contains information from the Vehicle class. This was made possible through integration of the ITaskLogger interface by means of dependency injection.

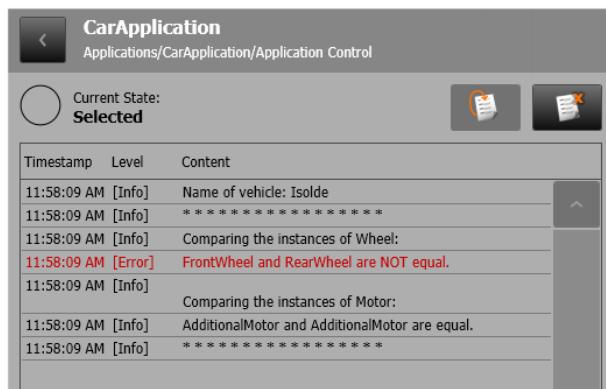


Fig. 15-7: CarApplication – Display on smartHMI

- Instances of Wheel

The compared objects are not identical. The result of the ELSE branch was displayed on the smartHMI and the names of the 2 objects are different.

- Instances of Motor

The result of the IF branch was displayed on the smartHMI. As both objects refer to the same instance due to the @Singleton annotation, the name is changed twice and corresponds to the one last set (here "AdditionalMotor").

15.4 Requesting individual values of a vector

Methods which request data from a frame generally return an object of the Vector class (package: com.kuka.roboticsAPI.geometricModel.math). The components of the vector can be requested individually.

Overview

The following methods of the Vector class are available:

Method	Description
getX()	Return value type: double Requests the X component of the vector
getY()	Return value type: double Requests the Y component of the vector
getZ()	Return value type: double Requests the Z component of the vector
get(index)	Return value type: double Requests the components determined by the <i>index</i> parameter Values of <i>index</i> (type: int): <ul style="list-style-type: none"> • 0: X component of the vector • 1: Y component of the vector • 2: Z component of the vector

15.5 Network communication via UDP and TCP/IP

Certain ports are enabled on the robot controller for communication with external devices via UDP or TCP/IP.

The following port numbers (client or server socket) can be used in a robot application:

- 30,000 to 30,010

15.6 Motion programming: PTP, LIN, CIRC

15.6.1 Synchronous and asynchronous motion execution

Description

In Sunrise, motion commands can be used for all movable objects of a station. A movable object can be a robot, for example, but also a tool which is attached to the robot flange or a workpiece held by a tool (e.g. a gripper).

Motion commands can be executed synchronously and asynchronously. The following methods are available for this:

- move(...) for synchronous execution

Synchronous means that the motion commands are sent in steps to the real-time controller and executed. The further execution of the program is interrupted until the motion has been executed. Only then is the next command sent.

- moveAsync(...) for asynchronous execution

Asynchronous means that the next program line is executed directly after the motion command is sent. The asynchronous execution of motions is required for approximating motions, for example.



CAUTION

If asynchronous motions are approximated, the robot may perform an approximate positioning motion at an unexpected point. Such unexpected approximate positioning motions can be avoided by grouping together approximated individual motions in a MotionBatch.
(>>> [15.6.6 "MotionBatch" Page 390](#))

The way in which the different motion types are programmed is shown by way of example for the object "robot".

Motion programming for tools and workpieces is described here:
(>>> [15.10.3 "Moving tools and workpieces" Page 408](#))



During programming, it is possible to specify values with a higher accuracy than the robot can achieve. For example, it is possible to specify position data in the nanometer range, but it is not possible to achieve this accuracy.

Syntax

Executing a motion synchronously:

`Object.move (Motion) ;`

Executing a motion asynchronously:

`Object.moveAsync (Motion) ;`

Explanation of the syntax

Element	Description
<i>Object</i>	Object of the station which is being moved The variable name of the object declared and initialized in the application is specified here.
<i>Motion</i>	Motion which is being executed The motion to be executed is defined by the following elements: <ul style="list-style-type: none"> • Motion type or block: ptp, lin, circ, spl or spline, splineJP, batch • Target position • Further optional motion parameters

15.6.2 PTP

Description

Executes a point-to-point motion to the end point. The coordinates of the end point are absolute.

The end point can be programmed in the following ways:

- Insert a frame from the application data in a motion instruction.
- Create a frame in the program and use it in the motion instruction.



The redundancy information for the end point – Status, Turn and redundancy angle – must be correctly specified. Otherwise, the end point cannot be correctly addressed.

- Specify the angles of axes A1 ... A7. All axis values must always be specified.

Syntax

PTP motion with a specified frame:

```
ptp(getApplicationContext().getFrame("/End point")) <.Motion parameters>
```

PTP motion with specified axis angles:

```
ptp(A1, A2, ... A7) <.Motion parameters>
```

Explanation of the syntax

Element	Description
<i>End point</i>	Path of the frame in the frame tree or variable name of the frame (if created in the program)
<i>A1 ... A7</i>	Axis angles of axes A1 ... A7 (type: double; unit: rad)
<i>Motion parameters</i>	Further motion parameters, e.g. velocity or acceleration

Examples

PTP motion to the “StartPos” frame:

```
robot.move(ptp(getApplicationContext().getFrame("/StartPos")));
```

PTP motion into the vertical stretch position:

```
robot.move(ptp(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
```

PTP motion to the “StartPos” frame with a specified relative velocity:

```
robot.move(ptpgetApplicationData().getFrame("/StartPos"))
.setJointVelocityRel(0.25);
```

15.6.3 LIN

Description

Executes a linear motion to the end point. The coordinates of the end point are Cartesian and absolute.

The end point can be programmed in the following ways:

- Insert a frame from the application data in a motion instruction.
- Create a frame in the program and use it in the motion instruction.

Syntax

```
lin(getApplicationData().getFrame("/End point")) <.Motion parameters>
```

Explanation of the syntax

Element	Description
<i>End point</i>	Path of the frame in the frame tree or variable name of the frame (if created in the program) The redundancy information for the end point – Status and Turn – are ignored in the case of LIN (and CIRC) motions. Only the redundancy angle is taken into account.
<i>Motion parameters</i>	Further motion parameters, e.g. velocity or acceleration

Examples

LIN motion to the “/Table/P1” frame:

```
robot.move(lin(getApplicationData().getFrame("/Table/P1")));
```

LIN motion with the Cartesian velocity specified:

```
robot.move(lin(getApplicationData().getFrame("/Table/P1"))
.setCartVelocity(150.0));
```

15.6.4 CIRC

Description

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 Cartesian and absolute.

The auxiliary point and end point can be programmed in the following ways:

- Insert a frame from the application data in a motion instruction.
- Create a frame in the program and use it in the motion instruction.

Syntax

```
circ(getApplicationData().getFrame("/Auxiliary point"), getApplicationData().getFrame("/End point"))
<.Motion parameters>
```

Explanation of the syntax

Element	Description
<i>Auxiliary point</i>	Path of the frame in the frame tree or variable name of the frame (if created in the program) The redundancy information for the end point – Status, Turn and redundancy angle – are ignored.
<i>End point</i>	Path of the frame in the frame tree or variable name of the frame (if created in the program) The redundancy information for the end point – Status and Turn – are ignored in the case of CIRC (and LIN) motions. Only the redundancy angle is taken into account.
<i>Motion parameters</i>	Further motion parameters, e.g. velocity or acceleration

Examples

CIRC motion to the end frame “/Table/P4” via the auxiliary frame “/Table/P3”:

```
robot.move(circ(getApplicationData().getFrame("/Table/P3"),
getApplicationData().getFrame("/Table/P4")));
```

CIRC motion with the absolute acceleration specified:

```
robot.move(circ(getApplicationData().getFrame("/Table/P3"),
getApplicationData().getFrame("/Table/P4")).setCartAcceleration(25));
```

15.6.5 LIN REL

Description

Executes a linear motion to the end point. The coordinates of the end point are relative to the end position of the previous motion, unless this previous motion is terminated by a break condition. In this case, the coordinates of the end point are relative to the position at which the motion was interrupted.

In a relative motion, the end point is offset as standard in the coordinate system of the moved frame. Another reference coordinate system in which to execute the relative motion can optionally be specified. The coordinates of the end point then refer to this reference coordinate system. This can for example be a frame created in the application data or a calibrated base.

The end point can be programmed in the following ways:

- Enter the Cartesian offset values individually.
- Use a frame transformation of type Transformation. The frame transformation has the advantage that the rotation can also be specified in degrees.

Syntax

LinRel motion with offset values:

```
linRel(x, y, z<, a, b, c>
<, Reference system>)
```

LinRel motion with frame transformation:

```
linRel(Transformation.ofDeg|ofRad(x, y, z, a, b, c)
<, Reference system>)
```

Explanation of the syntax

Element	Description
x, y, z	Offset in the X, Y and Z directions (type: double, unit: mm)
a, b, c	Rotation about the Z, Y and X axes (type: double) The unit depends on the method used: <ul style="list-style-type: none"> • Offset values and Transformation.ofRad: rad • Transformation.ofDeg: degrees
Reference system	Type: AbstractFrame Reference coordinate system in which the motion is executed

Examples

The moving frame is the TCP of a gripper. This TCP moves 100 mm in the X direction and 200 mm in the negative Z direction in the tool coordinate system from the current position. The orientation of the TCP does not change.

```
gripper.getFrame("/TCP2").move(linRel(100, 0, -200));
```

The robot moves 10 mm from the current position in the coordinate system of the P1 frame. The robot additionally rotates 30° about the Z and Y axes of the coordinate system of the P1 frame.

```
robot.move(linRel(Transformation.ofDeg(10, 10, 10, 30, 30,
0), getApplicationData().getFrame("/P1")));
```

15.6.6 MotionBatch

Description

Several individual motions can be grouped in a MotionBatch and thus transmitted to the robot controller at the same time. As a result, motions can be approximated within the MotionBatch.

The motion parameters, e.g. velocity, acceleration, orientation control, etc. can be programmed for the entire batch or per motion.



Only axis-specific motion parameters, e.g. setJoint...Rel(...), can be specified for the entire batch. Cartesian motion parameters, e.g. setCart...(...), must be specified in the individual block.

Both variants can appear together, e.g. to assign another parameter value to an individual motion than to the batch.



The motion parameter specified for a single block overwrites the motion parameter specified for the entire batch. This also applies if a lower parameter value is specified for the batch than for the individual block.

Syntax

```
Object.move (batch  
    Motion,  
    Motion,  
    ...  
    Motion,  
    Motion  
    ) < .Motion parameter >);
```

Explanation of the syntax

Element	Description
<i>Object</i>	Object of the station which is being moved The variable name of the object declared and initialized in the application is specified here.
<i>Motion</i>	Motion with or without motion parameters <ul style="list-style-type: none"> • ptp, lin, circ or spline
<i>Motion parameters</i>	Motion parameters which are programmed at the end of the batch apply to the entire batch. Only axis-specific motion parameters can be programmed!

15.7 Motion programming: spline

15.7.1 Programming tips for spline motions

- The number of spline segments in a spline block is only limited by the available memory.
- The planning of a spline motion with many small segments and small distances between points can take a very long time. To avoid excessively long planning times:
 - Program a maximum of 500 segments per spline block.
 - If possible, program distances between points > 5 mm.
- Spline motions (with many segments) can be programmed using an array of spline segments.
- 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 LIN and CIRC segments in cases where the workpiece necessitates straight lines and arcs. (Exception: use SPL segments for very short straight lines.) Otherwise, use SPL segments, particularly if the points are close together.
- Procedure for defining the path:
 1. First teach or calculate a few characteristic points. Example: points at which the curve changes direction.

2. Test the path. At points where the accuracy is still insufficient, add more SPL points.
- Avoid successive LIN and/or CIRC segments, as this often reduces the velocity to 0. To avoid this:
 - Program SPL segments between LIN and CIRC 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.
 - Replace a LIN segment with several SPL segments in a straight line. In this way, the path becomes a straight line.
 - Avoid successive points with identical Cartesian coordinates, as this reduces the velocity to 0.
 - If the robot executes points which lie on a work surface, a collision with the work surface is possible when approaching the first point.

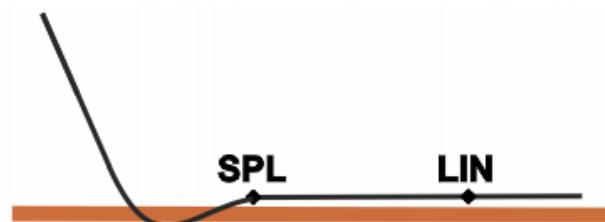


Fig. 15-8: Collision with work surface

A collision can be avoided by inserting a LIN segment before the work surface. Observe the recommendations for the LIN-SPL-LIN transition.

(>>> [14.6.3 "LIN-SPL-LIN transition" Page 352](#))

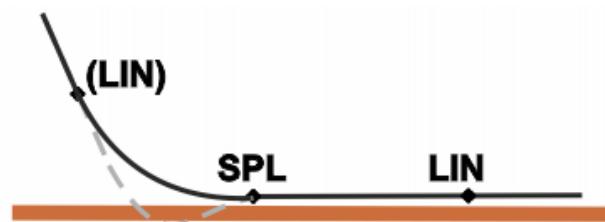


Fig. 15-9: Avoiding a collision with the work surface

- Avoid using SPL segments if the robot moves near the workspace limit. It is possible to exceed the workspace limit with SPL, even though the robot can reach the end frame in another motion type or by means of jogging.

15.7.2 Creating a CP spline block

Description

A CP spline block can be used to group together several SPL, LIN and/or CIRC segments to an overall motion.

A spline block must not include any other instructions, e.g. variable assignments or logic statements.

The motion parameters, e.g. velocity, acceleration, orientation control, etc. can be programmed for the entire spline block or per segment. Both variants can appear together, e.g. to assign a different parameter value to an individual segment than to the block.



The motion parameter specified for a single spline segment overwrites the motion parameter specified for the entire spline block. This also applies if a lower parameter value is specified for the spline block than for the spline segment.

Syntax

```
Spline Name = new Spline(
    Segment,
    Segment,
    ...
    Segment,
    Segment
) < .Motion parameter>;
```

Explanation of the syntax

Element	Description
Name	Name of the spline block
Segment	Motion with or without motion parameters <ul style="list-style-type: none"> • spl, lin or circ
Motion parameters	Motion parameters which are programmed at the end of the spline block apply to the entire spline block.

Example

```
Spline mySpline = new Spline(
    splgetApplicationData().getFrame("/P1"),
    circgetApplicationData().getFrame("/P2"),
    getApplicationData().getFrame("/P3"),
    splgetApplicationData().getFrame("/P4")
        .setCartVelocity(150),
    lingetApplicationData().getFrame("/P5")
        .setCartVelocity(250);
```

15.7.3 Creating a JP spline block

Description

A JP spline block can be used to group together several PTP segments as an overall motion.

A spline block must not include any other instructions, e.g. variable assignments or logic statements.

The motion parameters, e.g. velocity, acceleration, etc. can be programmed for the entire spline block or per segment. Both variants can appear together, e.g. to assign a different parameter value to an individual segment than to the block.



The motion parameter specified for a single spline segment overwrites the motion parameter specified for the entire spline block. This also applies if a lower parameter value is specified for the spline block than for the spline segment.

Syntax

```
SplineJP Name = new SplineJP(
```

```

Segment,
Segment,
...
Segment,
Segment
) < .Motion parameter>;

```

Explanation of the syntax

Element	Description
Name	Name of the spline block
Segment	PTP motion with or without motion parameters
Motion parameters	Motion parameters which are programmed at the end of the spline block apply to the entire spline block.

Example

```

SplineJP mySpline = new SplineJP(
    ptpgetApplicationData().getFrame("/P1"),
    ptpgetApplicationData().getFrame("/P2"))
.setJointVelocityRel(0.75);

```

15.7.4 Using spline in a motion instruction

Description

The spline motion programmed in a spline block is used as the motion type in the motion instruction.

Syntax

```
Object.move (spline block);
```

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved The variable name of the object declared and initialized in the application is specified here.
Spline block	Name of the spline block

Example

```
robot.move (mySpline);
```

15.8 Overview of motion parameters (PTP, LIN, CIRC, SPL, Spline)

The required motion parameters can be added in any order to the motion instruction. Dot operators and "set" methods are used for this purpose.

Method	Description
setCartVelocity(...)	<p>Absolute Cartesian velocity (type: double, unit: mm/s)</p> <ul style="list-style-type: none"> • > 0.0 <p>This value specifies the maximum Cartesian velocity at which the robot may move during the motion. Due to limitations in path planning, the maximum velocity may not be reached and the actual velocity may be lower.</p> <p>If no velocity is specified, the motion is executed with the fastest possible velocity.</p> <p>Note: This parameter cannot be set for PTP motions.</p>
setJointVelocity Rel(...)	<p>Axis-specific relative velocity (type: double, unit: %)</p> <ul style="list-style-type: none"> • 0.0 ... 1.0 <p>Refers to the maximum value of the axis velocity in the machine data.</p> <p>Maximum axis velocity, LBR Med 7 R800 / LBR Med 14 R820 (unit: °/s)</p> <ul style="list-style-type: none"> • A1: 98 / 85 • A2: 98 / 85 • A3: 100 / 100 • A4: 130 / 75 • A5: 140 / 130 • A6: 180 / 135 • A7: 180 / 135 <p>(>>> <i>15.8.1 "Programming axis-specific motion parameters"</i> <i>Page 397</i>)</p>
setCart Acceleration(...)	<p>Absolute Cartesian velocity (type: double, unit: mm/s²)</p> <ul style="list-style-type: none"> • > 0.0 <p>If no acceleration is specified, the motion is executed with the fastest possible acceleration.</p> <p>Note: This parameter cannot be set for PTP motions.</p>
setJointAcceleration Rel(...)	<p>Axis-specific relative acceleration (type: double, unit: %)</p> <ul style="list-style-type: none"> • 0.0 ... 1.0 <p>Refers to the maximum value of the axis acceleration in the machine data.</p> <p>Maximum acceleration, LBR Med 7 R800 / LBR Med 14 R820 (unit: °/s²)</p> <ul style="list-style-type: none"> • A1: 63.6326927 / 50.6895761 • A2: 64.1139773 / 51.7839255 • A3: 554.915354 / 86.8374834 • A4: 570.528454 / 91.4211458 • A5: 3533.68855 / 2983.47718 • A6: 1948.26277 / 833.848398 • A7: 5253.94277 / 4941.4344 <p>(>>> <i>15.8.1 "Programming axis-specific motion parameters"</i> <i>Page 397</i>)</p>

Method	Description
setCartJerk(...)	<p>Absolute Cartesian jerk (type: double, unit: mm/s³)</p> <ul style="list-style-type: none"> • > 0.0 <p>If no jerk is specified, the motion is executed with the fastest possible change in acceleration.</p> <p>Note: This parameter cannot be set for PTP motions.</p>
setJointJerkRel(...)	<p>Axis-specific relative jerk (type: double, unit: %)</p> <ul style="list-style-type: none"> • 0.0 ... 1.0 <p>Refers to the maximum value of the axis-specific change in acceleration in the machine data.</p> <p>(>>> 15.8.1 "Programming axis-specific motion parameters" Page 397)</p>
setBlendingRel(...)	<p>Relative approximation distance (type: double)</p> <ul style="list-style-type: none"> • 0.0 ... 1.0 <p>The relative approximation distance is the furthest distance before the end point at which approximate positioning can begin. If “0.0” is set, the approximation parameter does not have any effect.</p> <p>The maximum distance (= 1.0) is always the length of the individual motion or the length of the last segment in the case of splines. For motions which are not commanded within a spline, only the range between 0% and 50% is available for approximate positioning. In this case, if a value greater than 50% is parameterized, approximate positioning nevertheless begins at 50% of the block length.</p>
setBlendingCart(...)	<p>Absolute approximation distance (type: double, unit: mm)</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>The absolute approximation distance is the furthest distance before the end point at which approximate positioning can begin. If “0.0” is set, the approximation parameter does not have any effect.</p>
setBlendingOri(...)	<p>Orientation parameter for approximate positioning (type: double, unit: rad)</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>Approximation starts, at the earliest, when the absolute difference of the dominant orientation angle for the end orientation falls below the value set here. If “0.0” is set, the approximation parameter does not have any effect.</p>
setOrientationType(...)	<p>Orientation control (type: Enum)</p> <ul style="list-style-type: none"> • Constant • Ignore • OriJoint • VariableOrientation (default) <p>(>>> 14.9 "Orientation control with LIN, CIRC, SPL" Page 356)</p>

Method	Description
setOrientationReferenceSystem(...)	<p>Only relevant for CIRC motions: Reference system of orientation control (type: Enum)</p> <ul style="list-style-type: none"> • Base • Path <p>(>>> 14.9.1 "Orientation control reference system for CIRC" Page 358)</p>

15.8.1 Programming axis-specific motion parameters

Description

The following axis-specific motion parameters can be programmed:

- Relative velocity setJointVelocityRel(...)
- Relative acceleration setJointAccelerationRel(...)
- Relative jerk setJointJerkRel(...)

There are various ways of specifying these axis-specific relative values. A valid value for all axes, different values for each individual axis or a value for an individual axis.

By way of example, these possibilities are described using the relative velocity:

- `setJointVelocityRel(Value)`
If a value of type double is transferred, the relative velocity applies to all axes.
- `setJointVelocityRel(Array_variable)`
In order to assign each axis its own relative velocity, a double array is transferred with the corresponding axis values. In an array, the axis values of up to 12 axes can be defined, beginning with axis A1.
- `setJointVelocityRel(Axis, Value)`
To specify the relative velocity of an individual axis, this axis is transferred as JointEnum.

Examples

All axes move at 50% of maximum velocity:

```
robot.move(ptpgetApplicationData().getFrame("/P1"))
.setJointVelocityRel(0.5);
```

Axis A5 moves at 50%, all other axes move at 20% of maximum velocity:

```
double[] velRelJoints = {0.2, 0.2, 0.2, 0.2, 0.5, 0.2, 0.2};
robot.move(ptpgetApplicationData().getFrame("/P1"))
.setJointVelocityRel(velRelJoints);
```

Axis A4 moves at 50% of maximum velocity, all other axes move at maximum velocity:

```
robot.move(ptpgetApplicationData().getFrame("/P1"))
.setJointVelocityRel(JointEnum.J4, 0.5);
```

15.9 Programming manual guidance

Description

The robot can be guided using a hand guiding device. Manual guidance mode can be switched on in the application using the motion command `handGuiding()`. Manual guidance begins at the actual position which was reached before the mode was switched on.

In the application, motions before and after manual guidance are generally required. The time at which the enabling signal for manual guidance is issued has the following effect on execution of the application:

- If the signal for manual guidance is issued before manual guidance mode is switched on in the application, manual guidance mode will be active as soon as it is switched on. This means that the application is not paused when the mode is switched on, making for a smooth transition between Application mode and Manual guidance mode.

Precondition for this response: the application velocity is less than the maximum permissible velocity configured for manual guidance.

(>>> [13.14.5.3 "Velocity monitoring during manual guidance"](#)
[Page 280](#))

If the application is executed at a higher velocity, the application is paused before switching to manual guidance mode. (Then release the enabling switch, press the Start key and wait until the application is paused again.)

- If activation of manual guidance mode is reached in the application, even though the enabling signal for manual guidance has not yet been issued, the application waits for the enabling signal. Once the enabling signal has been issued, the robot can be guided manually.
- Manual guidance mode has ended when the signal for manual guidance has been cancelled, e.g. by releasing the enabling switch. If manual guidance mode in the application is followed directly by a motion command, the motion command is executed immediately after the enabling switch has been released.



CAUTION

The integrator must take the following into account in his risk management process:

If manual guidance motions are used in the robot application, the next motion command must not take effect immediately after the enabling switch has been released. It must be ensured that no hazards arise from the next (autonomous) motion command. An active confirmation can be requested before the next motion command is allowed to start, for example, in order to avoid hazards.

(>>> [15.27 "Program execution control"](#) [Page 501](#))

Preparation

The `handGuiding()` motion command belongs to the `HRCMotions` class. The class must be manually inserted into the import section of the robot application. The following line must be programmed:

```
import static com.kuka.roboticsAPI.motionModel.HRCMotions.*;
```



To enable the `HRCMotions` class to be integrated, the catalog element **Handguiding Support** must be selected in the station configuration (**Software** tab).

Syntax

```
Object.move(handGuiding());
```

Explanation of the syntax

Element	Description
<i>Object</i>	Object of the station which is being moved The variable name of the object declared and initialized in the application is specified here.

Example

```
1 robot.move(ptpgetApplicationData().getFrame("/P1"));
2 robot.move(handGuiding());
3 waitForUserInteraction();
4 robot.move(ptpgetApplicationData().getFrame("/P2"));
```

Line	Description
1	Frame “/P1” is addressed with a PTP motion.
2	Manual guidance mode is activated. The robot can be guided manually as soon as the enabling switch on the hand guiding device is pressed and held in the center position. When the signal for manual guidance has been cancelled, e.g. by releasing the enabling switch, Manual guidance mode has ended.
3	Program code to be implemented by the system integrator in accordance with his risk management process that interrupts program execution in such a way that the following motion command cannot take effect immediately (e.g. active confirmation).
4	Frame “/P2” is addressed with a PTP motion.

15.9.1 Overview of motion parameters (manual guidance)

For manual guidance, axis limits and velocity limits can be programmed. The required motion parameters can be added in any order to the motion command handGuiding(). Dot operators and “set” methods are used for this purpose.

Axis limits:

Method	Description
setJointLimitsEnabled(...)	Activation of the axis limits for manual guidance (type: boolean[]) <ul style="list-style-type: none"> • true: Axis limit active • false: Axis limit not active Note: This method refers to the limits that the user can set using the methods setJointLimitsMax(...) and setJointLimitsMin(...). The outermost axis limits of the robot (software limit switches) are always monitored.
setJointLimitsMax(...)	Upper axis limits (type: double[]; unit: rad)

Method	Description
setJointLimitsMin(...)	<p>Lower axis limits (type: double[]; unit: rad)</p> <p>Note: The lower axis limit must always be lower than the corresponding upper axis limit.</p>
setJointLimitViolationFreezesAll(...)	<p>Response if an axis limit is reached (type: boolean)</p> <ul style="list-style-type: none"> • true: If an axis limit is reached, all axes involved in the motion work against a further motion towards the limit switch. • false: If an axis limit is reached, only the affected axis works against a further motion towards the limit switch. <p>Default: true</p> <p>If this value is not set, the default value is automatically applied.</p>
setPermanentPullOnViolationAtStart(...)	<p>Response if an axis limit is already exceeded at the start of manual guidance (type: boolean)</p> <ul style="list-style-type: none"> • true: When the enabling signal for manual guidance is issued, the axis is moved automatically out of the non-permissible range. When the permissible range is reached, the motion is stopped automatically. • false: When the enabling signal for manual guidance is issued, the axis does not move. It must be moved out of the non-permissible range manually. <p>Default: false</p> <p>If this value is not set, the default value is automatically applied.</p>

Velocity limits:

Method	Description
setCartVelocityLimit(...)	<p>Cartesian velocity limit (type: double, unit: mm/s)</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 500.0</p> <p>If the velocity limit is exceeded, increasing torques act against the motion and cushion it.</p>
setJointVelocityLimit(...)	<p>Velocity limitation for all axes (type: double; unit: rad/s)</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 1.0</p> <p>If the velocity limit is exceeded, increasing torques act against the motion and cushion it.</p>

15.9.2 Axis limitation for manual guidance

Description

The motion range of each axis is limited by means of software limit switches. For manual guidance, additional axis limitations can be programmed, thereby further restricting the motion range:

- setJointLimitsMin(...), setJointLimitsMax(...)
- Define a lower and upper axis limit that must be specified individually for each axis.
- Defining a lower and an upper axis limit results in a permissible axis range, in which manual guidance is freely possible, and 2 non-permis-

sible axis ranges between the upper/lower axis limit and the respective software limit switch.

- `setJointLimitsEnabled(...)`

The defined axis limitation must be activated or deactivated individually for each axis.

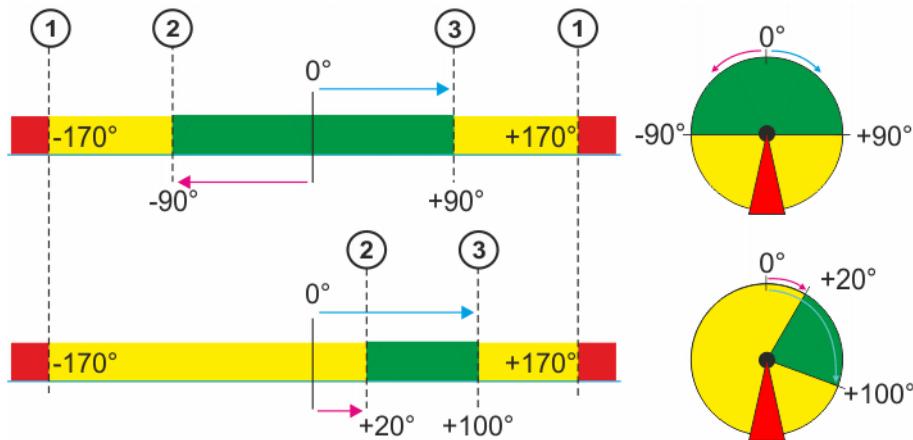


Fig. 15-10: Axis limits for manual guidance (examples)

- 1 Position of software limit switch
- 2 Lower limit of the permissible axis range
- 3 Upper limit of the permissible axis range

If one of the axis limits is reached during manual guidance, a virtual spring damper system is tensioned. This generates a resistance against any further motion towards the limit switch, with the resistance becoming greater the nearer an axis comes to the limit switch.

The following applies as standard:

- If an axis limit is reached, all axes involved in the motion work against a further motion towards the limit switch.

With `setJointLimitViolationFreezesAll(false)`, it is possible to define that only the axis that has reached the limit works against a further motion towards the limit switch.

- If an axis limit is already exceeded at the start of manual guidance, the affected axis must be moved manually out of the non-permissible range.

With `setPermanentPullOnViolationAtStart(true)`, it is possible to define that the axis is to move automatically out of the non-permissible range.

Example

```
@Inject
private LBR robot;
private HandGuidingMotion motion;
// ...

motion = handGuiding()
.setJointLimitsMax(+1.407, +0.872, +0.087, -0.785, +0.087,
+1.571, +0.087)
.setJointLimitsMin(-1.407, +0.175, -0.087, -1.571, -0.087,
-1.571, -0.087)
.setJointLimitsEnabled(false, true, false, true, false,
true, false)
```

```
.setJointLimitViolationFreezesAll(false)
.setPermanentPullOnViolationAtStart(true);

robot.move(motion);
```

15.9.3 Velocity limitation for manual guidance

Description

In addition to the axis limitation, velocity limits that may not be exceeded can be programmed for manual guidance:

- `setCartVelocityLimit(...)`
Defines the maximum permissible Cartesian velocity. It is monitored both at the robot flange and at the TCP.
- `setJointVelocityLimit(...)`
Defines the maximum permissible axis velocity for all axes.

As soon as the operator exceeds one of these maximum velocity limits axis in manual guidance, increasing torques act against the motion and cushion it.

Axis velocity reduction before axis limitation:

If the programmed maximum permissible axis velocity is exceeded near the axis limits during manual guidance, the axis velocity is continuously reduced to a minimum axis velocity specified by KUKA. This ensures that the manual guidance motion is automatically decelerated before the axis limits and the operator can only approach the axis limits at reduced velocity.

The method `setJointLimitViolationFreezesAll(...)` determines whether only the velocity of the affected axis is reduced, or the velocity of all axes involved in the motion.

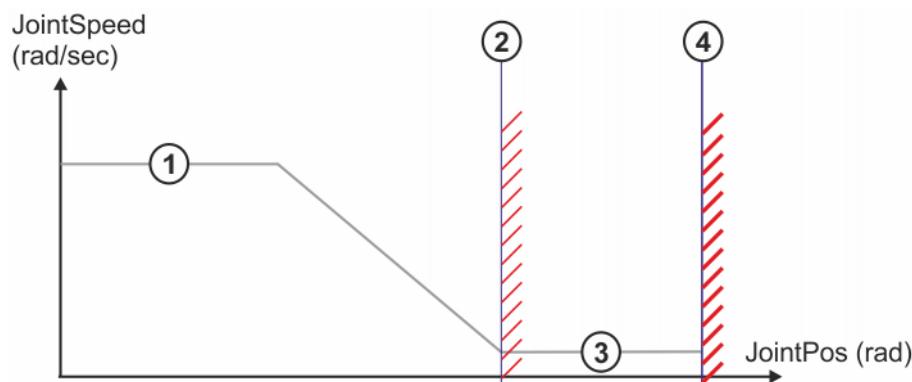


Fig. 15-11: Axis velocity reduction before axis limitation

- 1 Programmed maximum axis velocity for manual guidance
- 2 Programmed axis limit for manual guidance
- 3 Specified minimum axis velocity
- 4 Axis limitation by means of software limit switches

Example

```
@Inject
private LBR robot;
private HandGuidingMotion motion;
// ...
```

```

motion = handGuiding()
.setJointLimitsMax(+1.407, +0.872, +0.087, -0.785, +0.087,
+1.571, +0.087)
.setJointLimitsMin(-1.407, +0.175, -0.087, -1.571, -0.087,
-1.571, -0.087)
.setJointLimitsEnabled(false, true, false, true, false,
true, false)
.setJointVelocityLimit(0.5)
.setCartVelocityLimit(500.0)
.setJointLimitViolationFreezesAll(false)
.setPermanentPullOnViolationAtStart(true);

robot.move(motion);

```

15.10 Using tools and workpieces in the program

An application constitutes a programmed model of a real station and must therefore contain all movable objects and fixed geometric objects in the station. Examples of movable objects for a station are robots, tools and workpieces. Examples of fixed objects are support tables or conveyors.

Robots are automatically declared and initialized when the robot application is created. Tools and workpieces used in the robot application must be integrated using dependency injection.

Tools and workpieces can be created and managed as object templates in the **Object templates** view.

(>>> [9.3 "Object management" Page 168](#))

Data types

The data types for the objects in a station are predefined in the Robotic-
sAPI, e.g.:

Data type	Object
Controller	Robot controller
LBR	Lightweight robot
Tool	Tool
Workpiece	Workpiece

15.10.1 Integrating tools and workpieces

Description

Tools and workpieces created in the object templates can be integrated into robot and background applications using dependency injection. The name of the template is specified by means of an additional annotation.

Syntax

```

@.Inject
@Named("Template name")
private Data type Object name;

```

Explanation of the syntax

Element	Description
@Inject	Annotation for integrating resources by means of dependency injection
@Named	Annotation for specifying the object template to be used
<i>Template name</i>	Name of the object template as specified in the Object templates view
<i>private</i>	The keyword designates locally valid variables. Locally valid means that the data array can only be used by the corresponding class.
<i>Data type</i>	Class of the resource (Tool or Workpiece) that is to be integrated
<i>Object name</i>	Name of the identifier, as it is to be used in the application



The annotation @Named may be omitted for tools if there is only one single object template for a tool. The annotation is always required for other objects.

Example

Tools and workpieces in the object templates:

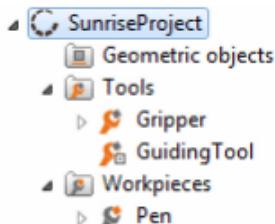


Fig. 15-12: Object templates

Declaration of the objects in the robot application:

```

public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    @Named("Gripper")
    private Tool gripper;

    @Inject
    @Named("GuidingTool")
    private Tool guidingTool;

    @Inject
    @Named("Pen")
    private Workpiece pen;

    @Override
    public void initialize() {
        // initialize your application here
    }

    @Override
    public void run() {
        // your application execution starts here
    }
}
    
```

```
}
```

15.10.2 Attaching tools and workpieces to the robot

In order to be able to use tools and workpieces as movable objects in motion instructions, they must be attached to the robot in the application via the method `attachTo(...)`.

- Tools are directly or indirectly attached to the robot flange.
- Workpieces are indirectly attached to the robot via a tool or another workpiece.

As soon as a tool or workpiece is attached to the robot via the method `attachTo(...)`, the load data from the robot controller are taken into account. In addition, all frames of the attached object can be used for the motion programming.

15.10.2.1 Attaching a tool to the robot flange

Description

Via the method `attachTo(...)`, the origin frame of a tool is attached to the flange of a robot used in the application. The robot flange is accessed via the method `getFlange()`.

Syntax

```
Tool.attachTo(Robot.getFlange());
```

Explanation of the syntax

Element	Description
<i>Tool</i>	Name of the tool variable
<i>Robot</i>	Name of the robot variable

Example

A guiding tool is attached to the robot flange.

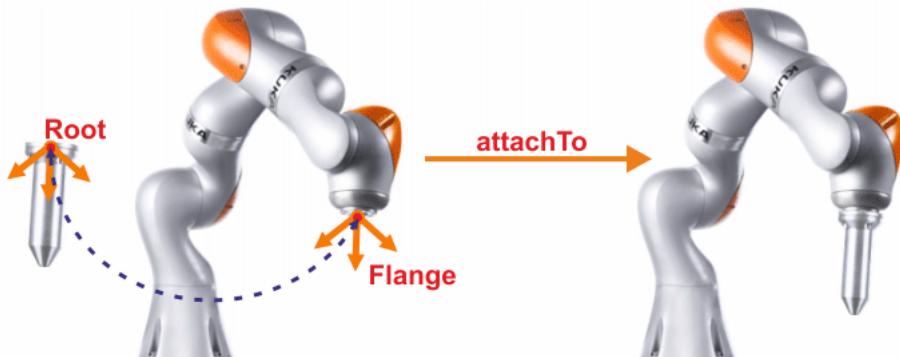


Fig. 15-13: Attaching the guiding tool to the flange.

```
@Inject
private LBR robot;
@Inject
private Tool guidingTool;
// ...
```

```

@Override
public void initialize() {
    // ...
    guidingTool.attachTo(robot.getFlange());
    // ...
}

```

15.10.2.2 Attaching a workpiece to other objects

Description

As standard, the origin frame of the workpiece is used to attach it to the frame of another object.

However, every other frame created for a workpiece can also be used as a reference point for attaching to another object.

Frames for tools and workpieces can be created in the **Object templates** view.

(>>> [9.3.6 "Creating a frame for a tool or workpiece" Page 172](#))

In order to use a frame in the program, the object frame is requested with the method `getFrame(...)`. As an input parameter, this contains the path of the frame as a string.

Syntax

To use the origin frame for the attachment:

`Workpiece.attachTo(Object.getFrame("End frame"));`

To use another reference frame for the attachment:

`Workpiece.getFrame("Reference frame").attachTo(Object.getFrame("End frame"));`

Explanation of the syntax

Element	Description
<code>Workpiece</code>	Name of the workpiece variable
<code>Reference frame</code>	Reference frame of the workpiece which is used for the attachment to the other object
<code>End frame</code>	Frame of the object to which the reference frame of the workpiece is attached



After the attach, the reference frame of the workpiece and the end frame of the object attached to it match.

Example 1

A pen is attached to the gripper frame via its origin frame.

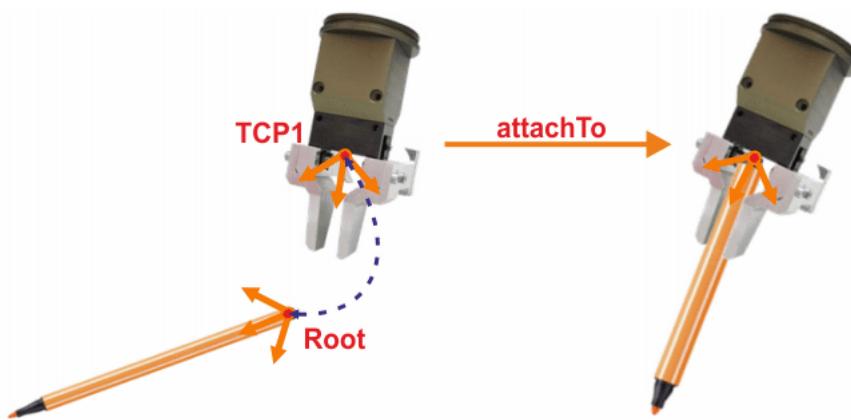


Fig. 15-14: Pen in gripper (attachment via origin frame)

```
@Inject
private LBR robot;
@Inject
private Tool gripper;

@Inject
@Named("Pen")
private Workpiece pen;
// ...

@Override
public void run() {
    // ...
    pen.attachTo(gripper.getFrame("/TCP1"));
    // ...
}
```

Example 2

A 2nd frame is defined at the tip of the gripper. If this is to be used to grip the pen, attachment via the origin frame of the pen is not possible. For this purpose, a grip point was created on the pen. This is used as the reference frame for the attachment to the gripper.

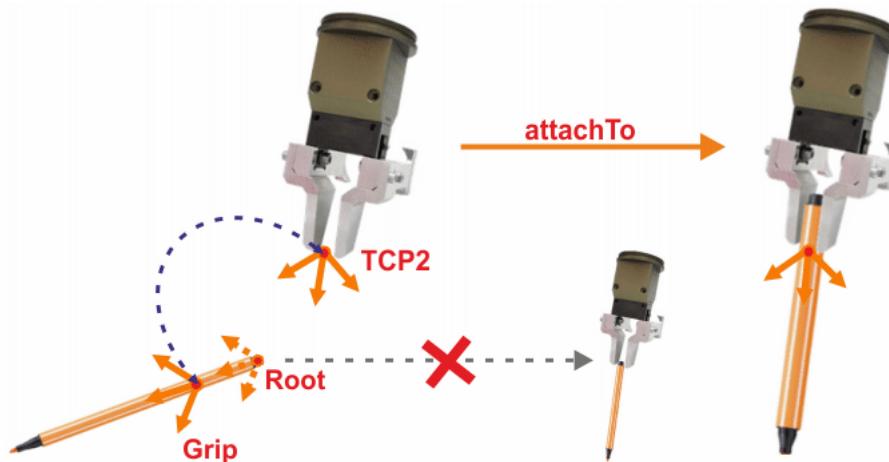


Fig. 15-15: Pen in gripper (connection via grip frame)

```
@Inject
private LBR robot;
@Inject
private Tool gripper;
```

```

@Inject
@Named("Pen")
private Workpiece pen;
// ...

@Override
public void run() {
    // ...
    pen.getFrame("/Grip").attachTo(gripper.getFrame("/TCP2"));
    // ...
}

```

15.10.2.3 Detaching objects

Description

If a tool is removed or a workpiece is set down, the object must also be detached in the application. The method `detach()` is used for this purpose.

Syntax

`Object.detach();`

Explanation of the syntax

Element	Description
<code>Object</code>	Name of the object variable

Example

The guidance tool is detached.

```
guidingTool.detach();
```

15.10.3 Moving tools and workpieces

Description

Every movable object in a station can be moved with `move(...)` and `moveAsync(...)`. The reference point of the motion is dependent on the object type:

- If a robot is moved, the reference point is always the robot flange center point.
- If a tool or workpiece is moved, the standard reference point is the default motion frame which was defined for this object in the **Object templates** view.

(>>> [9.3.8 "Defining a default motion frame" Page 175](#))

In this case, the tool or workpiece is linked directly to the motion command via the variable name declared in the application.

- However, any other frame created for a tool or workpiece can also be programmed as a reference point of the motion.

In this case, using the method `getFrame(...)`, the path to the frame of the object used for the motion must be specified (on the basis of the origin frame of the object).

Syntax

To use the default frame of the object for the motion:

```
Object.move(Motion);
```

To use a different frame of the object for the motion:

```
Object.getFrame("Moved frame").move(Motion);
```

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved The variable name of the object declared and initialized in the application is specified here.
Moved frame	Path to the frame of the object which is used for the motion
Motion	Motion which is being executed

Examples

The PTP motion to point P1 is executed with the default frame of the gripper.

```
gripper.attachTo(robot.getFlange());
gripper.move(ptpgetApplicationData().getFrame("/P1"));
```

The PTP motion to point P1 is executed with a different frame than the default frame of the gripper, here TCP1:

```
gripper.attachTo(robot.getFlange());
gripper.getFrame("/TCP1").move(ptpgetApplicationData().getFrame("/P1"));
```

A pen is gripped. The next motion is a PTP motion to point P20. This point is executed with the default frame of the workpiece "pen".

```
gripper.attachTo(robot.getFlange());
// ...
pen.attachTo(gripper.getFrame("/TCP1"));
pen.move(ptpgetApplicationData().getFrame("/P20"));
```

15.10.4 Integrating dedicated object classes with dependency injection

Description

Tools and workpieces created in the object templates are based on the classes Tool and Workpiece. Specific properties or functions that tools and workpieces generally have are not considered by these basic classes. For a gripper, examples might include functions for opening and closing.

Such specific object properties and functions can be defined in separate object classes. The following steps are required in order to be able to use the user-defined object classes in the same way as the basic classes in applications:

Step	Description
1	<p>Derive a new object class from a suitable basic class:</p> <ul style="list-style-type: none"> • Basic class for tools: com.kuka.roboticsAPI.geometricModel.Tool • Basic class for workpieces: com.kuka.roboticsAPI.geometricModel.Workpiece <p>The constructor of the created object class must have the following properties:</p> <ul style="list-style-type: none"> • Visibility level public • Transfer parameter of type String (name of the object template is transferred) • Must not be annotated with <code>@Inject</code>
2	<p>Define object properties and functions in the new object class.</p>
3	<p>In the object templates, assign the new object class to the desired objects. For this, enter the full identifier (Package name.Class name) of the object class under Template class in the Properties view.</p> <p>Note: Object templates that use an object class derived from a basic class are integrated into an application such as this by means of dependency injection.</p>



Entering the object class as a template in the properties is especially important because the load data of the templates are then automatically assigned to the integrated object. If this is not done, the object class behaves like a specially created class without dependencies.
 (>>> [15.3.3.2 "Dependency injection for dedicated types" Page 382](#))
 If the load data are required for motions (e.g. for a robot under impedance control), this can result in unexpected motions of the robot.



Dependency injection can also be used in dedicated classes. For example, the ITaskLogger can be integrated in order to display output information on the smartHMI.

Singletons

Object classes that are derived from Tool and used in a Sunrise project as described here are always integrated as singletons. This means that each object annotated with the type of the object class refers to the same instance.

As standard, object classes that are derived from Workpiece are not singletons. When annotating, a new instance is therefore created every time. Workpieces can be made singletons by placing the annotation `@Singleton` before the header of the class.

Procedure

Derive a new object class from a basic class:

1. Select the desired Sunrise project in the **Package Explorer**.
2. Select the menu sequence **File > New > Class**. The **New Java class** window opens.
3. In the **Package**: box, enter a name for the Java package in which the new class is to be created.
4. Enter a name for the new class in the **Name** box.

5. To the right of the **Superclass:** box, click on **Browse....** The **Super-class selection** window is opened.
6. Enter the name of the basic class in the **Select type** box (**Tool or Workpiece**).
7. Confirm the selection with **OK**. The name of the basic class is now displayed in the **Superclass:** box.
8. Click on **Finish**. The Java package with the newly created class is inserted into the source folder of the Sunrise project and opened in the editor area.
9. Create a constructor with the desired properties.
10. The required arrays and methods can now be defined.

Example

Step 1:

For a gripper, the object class Gripper is created using the procedure described above. The class Gripper is derived from the basic class Tool and expands the basic class to include the functions for opening and closing the gripper.

```

1 package tools;
2 import com.kuka.roboticsAPI.geometricModel.Tool;
3 public class Gripper extends Tool {
4     @Inject
5     private ITaskLogger logger;
6
7     public Gripper(String name) {
8         super(name);
9     }
10
11    /**
12     * Opens the gripper
13     */
14    public void openGripper() {
15        // ...
16        logger.info("Gripper is open.");
17    }
18
19    /**
20     * Closes the gripper
21     */
22    public void closeGripper() {
23        // ...
24        logger.info("Gripper is closed.");
25    }
26 }
```

Line	Description
1	Name of the Java package that contains the class Gripper
4 ... 5	Integration of the ITaskLogger interface by means of dependency injection
7 ... 9	Standard constructor of the class Gripper (adopted from Tool)
14 ... 17	Method openGripper() for opening the gripper
22 ... 25	Method closeGripper() for closing the gripper

Line	Description
16, 24	Information displayed on smartHMI with the aid of the ITas-kLogger interface

Step 2:

An object template with the name “ExampleGripper” is created for the gripper. The object class Gripper is assigned to the object template:

- Entry under **Template class** in the **Properties** view: tools.Gripper
The name of the Java package (here “tools”) that contains the class Gripper must be specified.

Step 3:

The object class Gripper and the corresponding functions can be used in the robot application.

```

1 public class ExampleApplication extends
   RoboticsAPIApplication {
2   @Inject
3   private LBR robot;
4   @Inject
5   private Gripper gripper;
6
7   @Override
8   public void initialize() {
9     // initialize your application here
10    // ...
11    gripper.attachTo(robot.getFlange());
12    // ...
13  }
14
15  @Override
16  public void run() {
17    // your application execution starts here
18    // ...
19    gripper.openGripper();
20    gripper.move(lin(getFrame("/GripPos")));
21    gripper.closeGripper();
22    // ...
23  }
24 }
```

Line	Description
4 ... 5	A tool of type Gripper is integrated. The tool has the functions defined in the object class Gripper.
11	The tool is attached to the robot flange.
19 ... 21	The functions defined in the object class Gripper are used to program a gripping process: <ul style="list-style-type: none"> Open gripper, move to grip position, close gripper.

15.10.5 Transferring workpiece load data to the safety controller

Description

If a workpiece load-specific AMF is used in the safety configuration and, at the same time, workpieces are picked up, the user must use the setSa-

fetyWorkpiece(...) method to communicate to the safety controller which workpiece is currently being used. Workpiece load-specific AMFs include:

- *TCP force monitoring*
- *Base-related TCP force component*
- *Collision detection*

(>>> [9.3.11 "Safety-oriented use of workpieces" Page 181](#))

The setSafetyWorkpiece(...) method belongs to the LBR class and can be used in both robot applications and background applications.

setSafetyWorkpiece(...) is used to transfer the workpiece load data to the safety controller. If a workpiece is set down again and there are no longer any workpiece load data to be taken into consideration, the value `null` must be transferred.



The workpiece load data transferred to the safety controller using setSafetyWorkpiece(...) are not safety-oriented. For this reason, in the event of an error, the AMF *Collision detection* may use load data which deviate from the actual workpiece load. These deviations are misinterpreted as external axis torques.

The workpiece load data transferred with setSafetyWorkpiece(...) are taken into consideration exclusively by the safety controller. If the workpiece load data are also to be taken into consideration in the non-safety-oriented part of the robot controller, the workpieces must also be integrated by means of the corresponding commands.

(>>> [15.10.1 "Integrating tools and workpieces" Page 403](#))

(>>> [15.10.2.2 "Attaching a workpiece to other objects" Page 406](#))

Syntax

```
robot.setSafetyWorkpiece(Workpiece);
```

Explanation of the syntax

Element	Description
<code>robot</code>	Type: LBR Name of the robot
<code>Workpiece</code>	Type: PhysicalObject Workpiece whose load data are to be transferred to the safety controller If no workpiece is to be taken into consideration any longer, <code>null</code> must be transferred.

Example

A safety-oriented tool and 2 workpieces are created in the object templates.

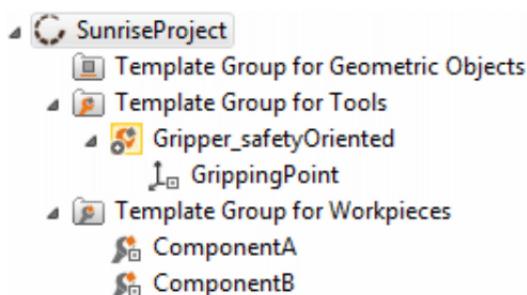


Fig. 15-16: Object templates: workpieces and tool

The tool contains the frame “GrippingPoint”, which serves as a gripping point for workpieces and which is selected as the standard frame for motions.

In the application, the workpiece “ComponentA” is picked up and set down. The workpiece “ComponentB” is then picked up. All load changes are to be taken into consideration in both the safety-oriented and non-safety-oriented part of the robot controller.

```
public class ChangeOfLoadExample extends RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private Tool gripper;

    @Inject
    @Named("ComponentA")
    private Workpiece componentA;

    @Inject
    @Named("ComponentB")
    private Workpiece componentB;

    @Override
    public void initialize() {
        // ...
        // attach gripper to robot flange
        gripper.attachTo(robot.getFlange());
    }

    @Override
    public void run() {
        // ...
        // after pick-up, attach workpiece to set load data for
        // motion control
        componentA.attachTo(gripper.getDefaultMotionFrame());
        // set load data for safety controller
        robot.setSafetyWorkpiece(componentA);
        // ...
        // after putting it down, detach workpiece to no longer
        // consider its load for motion control
        componentA.detach();

        // workpiece is no longer considered for safety
        // controller
        robot.setSafetyWorkpiece(null);
        // ...
        // pick-up of second workpiece
    }
}
```

```

        componentB.attachTo(gripper.getDefaultMotionFrame());
        robot.setSafetyWorkpiece(componentB);
        // ...
    }
}

```

15.11 Using inputs/outputs in the program

When exporting an I/O configuration from WorkVisual, a separate Java class is created for each I/O group in the corresponding Sunrise project. Each of these Java classes contains the methods required for programming, in order to be able to read the inputs/outputs of an I/O group and write to the outputs of an I/O group.



The source code of the Java classes of the package **com.kuka.generated.ioAccess** must not be changed manually. To expand the functionality of an I/O group, it is possible to derive further classes from the classes created or to continue to use objects from these classes, e.g. as arrays of their own classes (aggregating).

To use the inputs/outputs of an I/O group in the application, the user must integrate the I/O group by means of dependency injection.

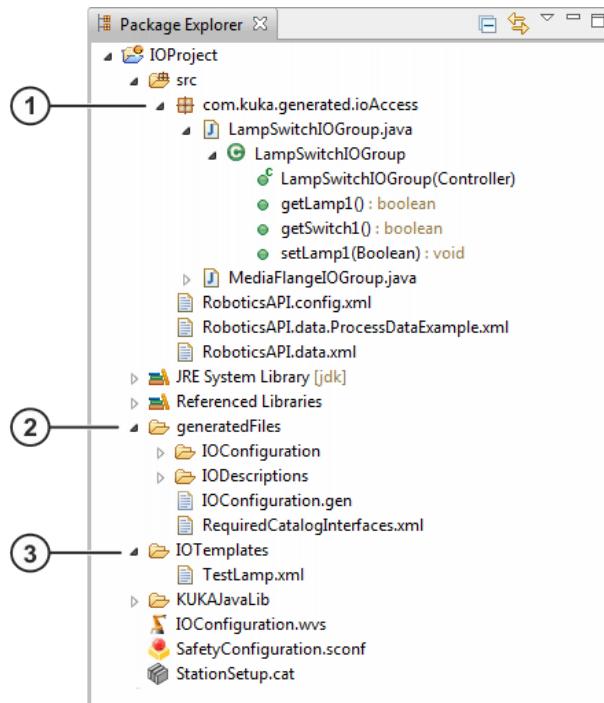


Fig. 15-17: Project structure after exporting the I/O configuration

Item	Description
1	<p>com.kuka.generated.ioAccess Java package</p> <p>The class created for an I/O group and the methods of this class are saved in the package.</p> <p>The Java class <i>NameIOGroup.java</i> (here: LampSwitch-IOGroup.java) contains the following elements:</p> <ul style="list-style-type: none"> • Class name of the I/O group: <i>NameIOGroup</i> • Constructor for assigning the robot controller to the I/O group: <i>NameIOGroup(Controller)</i> • get and set methods for every configured output: <i>getOutput()</i>, <i>setOutput(Value)</i> • “Get” method for every configured input: <i>getInput()</i>
2	<p>generatedFiles folder</p> <ul style="list-style-type: none"> • IODescriptions folder <p>The data in an I/O group are saved in an XML file. The XML file can be displayed but not edited.</p>
3	<p>IOTemplates folder</p> <p>The data of an I/O group saved as a template are saved in an XML file. The XML file can be displayed but not edited.</p> <p>A template can be copied into another Sunrise project in order to be used there. The template can then be imported into Work-Visual, edited there and re-exported.</p> <p>(>>> 11.5.8 "Importing an I/O group from a template" Page 216) (>>> 11.5.7 "Exporting an I/O group as a template" Page 216)</p>



The **generatedFiles** folder is used by the system and must not be used for saving files created by the user.

15.11.1 Integrating an I/O group

Description

I/O groups can be integrated into robot and background applications by means of dependency injection. As a result, the Java package **com.kuka.generated.ioAccess** is automatically imported with the classes and methods of the I/O group.

Syntax

```
@Inject
private Data type Group name;
```

Explanation of the syntax

Element	Description
<code>@Inject</code>	Annotation for integrating resources by means of dependency injection
<code>private</code>	The keyword designates locally valid variables. Locally valid means that the data array can only be used by the corresponding class.

Element	Description
<i>Data type</i>	Class of the resource (I/O group) that is to be integrated Class name of the I/O group: <ul style="list-style-type: none"> • <i>NameIOGroup</i> <i>Name</i> = Name of the I/O group, as defined in WorkVisual
<i>Group name</i>	Name of the identifier, as it is to be used in the application

Example

Integrating the I/O group “LampSwitch”:

```
public class ExampleApplication extends
RoboticsAPIApplication {
    // ...
    @Inject
    private LampSwitchIOGroup lampSwitch;
    // ...

    @Override
    public void initialize() {
        // ...
    }

    @Override
    public void run() {
        // ...
    }
}
```

15.11.2 Reading inputs/outputs

Description

The “get” method of an input/output is used to request the state of the input/output.

Syntax

Group name.get*Input|Output*() ;

Explanation of the syntax

Element	Description
<i>Group name</i>	Name of the identifier of the I/O group
<i>Input</i>	Name of the input (as defined in WorkVisual)
<i>Output</i>	Name of the output (as defined in WorkVisual)

Example

The state of the switch at input “Switch1” and of the lamp at output “Lamp1” is requested.

```
public void run() {
    // ...
    lampSwitch.getLamp1();
    lampSwitch.getSwitch1();
```

```
// ...
}
```

15.11.3 Setting outputs



WARNING

Outputs are switched in certain situations although a safety-oriented stop request is present (e.g. in the case of a pressed EMERGENCY STOP or violated space monitoring). This can cause unexpected motions of the connected periphery (e.g opening of a gripper).

The following situations can now occur:

- Background application switches output.
- Function called via user key switches output.
- Robot applications continue running to the next synchronous motion command after a stop request. The code executed up to that point switches the output.

The behavior described can also be desirable; however, there must never be any danger to human and machine. This must be ensured by the integrator, e.g. by deenergizing outputs with hazard potential.

NOTICE

It is not permissible to set outputs in a robot application that signal system states to the external controller. Failure to observe this precaution may result in malfunctioning of the external controller and damage to property.

Description

The “set” method of an output is used to change the value of the output.



No “set” methods are available for inputs. They can only be read.

Syntax

```
Group name.setOutput(Value);
```

Explanation of the syntax

Element	Description
<i>Group name</i>	Name of the identifier of the I/O group
<i>Output</i>	Name of the output (as defined in WorkVisual)
<i>Value</i>	Value of the output The data type of the value to be transferred depends on the output type.

Example

The lamp at output “Lamp1” is switched on and then switched off after 2000 ms.

```
public void run() {
  // ...
  lampSwitch.setLamp(true);
  ThreadUtil.millisleep(2000);
  lampSwitch.setLamp(false);
  // ...
}
```

15.12 Requesting axis torques

Description

Certain robot types, e.g. the LBR, have a joint torque sensor in each axis which measures the torque acting on the axis. The interface `ITorqueSensitiveRobot` contains the methods required for requesting sensor data from the robot.

- `getMeasuredTorque()`

The measured torque values can be requested and evaluated in the application via the method `getMeasuredTorque()`.

- `getExternalTorque()`

Frequently, it is not the pure measured values which are of interest but rather only the externally acting torques, without the component resulting from the weight of the robot and mass inertias during motion. These values are referred to as external torques. These external torques be accessed via the method `getExternalTorque()`.



In order to be able to display external torques correctly, the load mounted on the robot must be configured correctly and communicated to the system.

- `getSingleTorqueValue(...)`, `getTorqueValues()`

The methods `getMeasuredTorque()` and `getTorqueValues()` return an object of the type `TorqueSensorData` containing the torque sensor data of all axes. From this object, it is then possible to request either all values as an array with `getTorqueValues(...)` or a single axis value with `getSingleTorqueValue(...)`.



The data requested from the joint torque sensors via Java are not available in real time. This means that the data supplied by the system in the program were already created several milliseconds earlier.

Syntax

To request the measured sensor data:

```
TorqueSensorData measuredData = robot.getMeasuredTorque();
```

To request externally acting torque data:

```
TorqueSensorData externalData = robot.getExternalTorque();
```

To request torque values of all axes from the sensor data:

```
double[] allValues = measuredData|externalData.getTorqueValues();
```

To request torque values of a specific axis from the sensor data:

```
double singleValue =  
measuredData|externalData.getSingleTorqueValues(joint);
```

Explanation of the syntax

Element	Description
<i>measuredData</i>	Type: <code>TorqueSensorData</code> Variable for the return value of <code>getMeasuredTorque()</code> . The return value contains the measured sensor data.
<i>externalData</i>	Type: <code>TorqueSensorData</code> Variable for the return value of <code>getExternalTorque()</code> . The return value contains the externally acting torques.

Element	Description
<i>robot</i>	Type: LBR Name of the robot from which the sensor data are requested
<i>allValues</i>	Type: double[]; unit: Nm Array with all torque values which are requested from the sensor data
<i>singleValue</i>	Type: double; unit: Nm Torque value of the axis which is requested from the sensor data
<i>joint</i>	Type: JointEnum Axis whose torque value is to be requested

Example

For a specific process step, the measured and externally acting torques are requested in all axes and saved in an array to be evaluated later. The measured torque in axis A2 is read and displayed on the smartHMI. For output purposes, a logger object has been integrated with dependency injection.

```
TorqueSensorData measuredData = robot.getMeasuredTorque();
TorqueSensorData externalData = robot.getExternalTorque();

double[] measuredTorques = measuredData.getTorqueValues();
double[] externalTorques = externalData.getTorqueValues();

double torqueA2 =
measuredData.getSingleTorqueValue(JointEnum.J2);
logger.info("Currently measured torque for joint 2 [Nm]:" + torqueA2);
```

15.13 Reading Cartesian forces and torques

Certain robot types, e.g. an LBR, have a joint torque sensor in each axis which measures the torque acting on the axis. The robot controller calculates the Cartesian forces and torques using the measured torques.

The interface **IForceSensitiveRobot** contains the methods for requesting the external Cartesian forces and torques currently acting on the robot flange, the TCP of a tool or any point of a gripped workpiece.

The following points must be taken into consideration:

- The Cartesian forces and torques are estimated based on the measured values of the joint torque sensors.

A force application point must be specified for the calculation. The external Cartesian forces and torques calculated for the force application point are only meaningful in terms of the physics involved if there are no external forces acting on any other points on the robot.
- The reliability of the calculated values can decrease considerably in extreme poses, e.g. extended positions or singularities.
- The quality and validity of the calculated values can be checked.
- When changing the load data, e.g. with the **attachTo** command, the request can only be executed after the motion command has been sent to the robot controller. For this purpose, a null space motion or the motion command **positionHold(...)** is sufficient.

15.13.1 Requesting external Cartesian forces and torques

Description

The method `getExternalForceTorque(...)` is used by the robot to read the external Cartesian forces and torques currently acting on the robot flange, the TCP of a tool or any point of a gripped workpiece.

The method receives a frame as the transfer parameter. The transferred frame is the reference frame for calculating the forces and torques, e.g. the tip of a probe. The method calculates the externally applied forces and torques for the position described by the frame.

For a meaningful calculation in terms of the physics involved, the transferred frame must describe a point which is mechanically fixed to the flange. The given frame must also be statically connected to the robot flange frame in the frame structure.

Optionally, a second frame can be transferred to the method as a parameter. This frame specifies the orientation of a coordinate system in which the forces and torques are represented.

Syntax

```
ForceSensorData data = robot.getExternalForceTorque(  
    measureFrame<, orientationFrame>);
```

Explanation of the syntax

Element	Description
<i>data</i>	Type: ForceSensorData Variable for the return value of <code>getExternalForceTorque(...)</code> . The return value contains the calculated Cartesian forces and torques.
<i>robot</i>	Type: LBR Name of the robot
<i>measure Frame</i>	Type: AbstractFrame Reference frame for calculation of the Cartesian forces and torques.
<i>orientation Frame</i>	Type: AbstractFrame Optional: Orientation of the frame in which the forces and torques are represented.

Examples

Requesting the external forces and torques acting on the robot flange:

```
ForceSensorData data =  
    robot.getExternalForceTorque(robot.getFlange());
```

Requesting the external forces and torques acting on the robot flange with the orientation of the world coordinate system:

```
ForceSensorData data =  
    robot.getExternalForceTorque(robot.getFlange(),  
    World.Current.getRootFrame());
```

15.13.2 Requesting forces and torques individually

Description

The external Cartesian forces and torques requested with `getExternalForceTorque()` can be requested separately from one another. The class `ForceSensorData` provides the following methods for this:

- `getForce()`
- `getTorque()`

The result of these requests is a vector in each case. The values for each degree of freedom can be requested individually with the methods of the `Vector` class.

(>>> 15.4 "Requesting individual values of a vector" Page 385)

Syntax

To request a force vector:

```
Vector force = data.getForce();
```

To request a torque vector:

```
Vector torque = data.getTorque();
```

Explanation of the syntax

Element	Description
<code>force</code>	Type: vector (<code>com.kuka.roboticsAPI.geometricModel.math</code>) Vector with the Cartesian forces which act in the X, Y and Z directions (unit: N)
<code>torque</code>	Type: vector (<code>com.kuka.roboticsAPI.geometricModel.math</code>) Vector with the Cartesian torques which act about the X, Y and Z axes (unit: Nm)
<code>data</code>	Type: <code>ForceSensorData</code> Variable for the return value of <code>getExternalForceTorque(...)</code> . The return value contains the calculated Cartesian forces and torques.

Example

Requesting the Cartesian force which is currently acting on the robot flange in the X direction:

```
ForceSensorData data =
robot.getExternalForceTorque(robot.getFlange());

Vector force = data.getForce();
double forceInX = force.getX();
```

15.13.3 Checking the reliability of the calculated values

Description

In unfavorable robot positions, the calculated Cartesian forces and torques can deviate from the actual forces and torques applied. In particular near singularities, several of the calculated values are highly unreliable and can be invalid. Depending on the axis position, this only applies to some of the calculated values.

The quality and validity of the calculated values can be evaluated and requested in the program. The class ForceSensorData provides the following methods for this:

- `getForceInaccuracy()`, `getTorqueInaccuracy()`

The inaccuracy of the calculated force and torque values can be requested.

The result of these requests is a vector in each case. The values for each degree of freedom can be requested individually with the methods of the Vector class.

(>>> **15.4 "Requesting individual values of a vector" Page 385**)

Depending on the axis position, the quality of the calculated values for the individual degrees of freedom may be different. By requesting the individual values, it is possible to determine the degrees of freedom for which the calculation of forces and torques in the current pose supplies valid values.

- `isForceValid(...)`, `isTorqueValid(...)`

The validity of the calculated force and torque values can be requested.

A limit value for the maximum permissible inaccuracy up to which the calculated values are still valid is transferred as a parameter for each method.

Syntax

Requesting the inaccuracy of the calculated values:

```
Vector force = data.getForceInaccuracy();  
Vector torque = data.getTorqueInaccuracy();
```

Requesting the validity of the calculated values:

```
boolean valid = data.isForceValid(tolerance);  
boolean valid = data.isTorqueValid(tolerance);
```

Explanation of the syntax

Element	Description
force	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the values for the inaccuracy with which the Cartesian forces acting in the X, Y and Z directions are calculated (unit: N)
torque	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the values for the inaccuracy with which the Cartesian torques acting about the X, Y and Z axes are calculated (unit: Nm)
data	Type: ForceSensorData Variable for the return value of <code>getExternalForceTorque(...)</code> . The return value contains the calculated Cartesian forces and torques.
tolerance	Type: double; unit: N or Nm Limit value for the maximum permissible inaccuracy up to which the calculated Cartesian forces and torques are still valid

Element	Description
<code>valid</code>	<p>Type: boolean</p> <p>Variable for the return value of <code>isForceValid(...)</code> or <code>isTorqueValid(...)</code></p> <ul style="list-style-type: none"> • true: The inaccuracy value in all Cartesian directions is less than or equal to the limit value defined with <i>tolerance</i>. • false: The inaccuracy value in one or more Cartesian directions exceeds the <i>tolerance</i> value

Example

A certain statement block should only be executed if the external Cartesian forces acting along the axes of the flange coordinate system have been calculated with an accuracy of 20 N or better.

```
ForceSensorData data =
robot.getExternalForceTorque(robot.getFlange()) ;

if (data.isForceValid(20)) {
    //do something
}
```

15.14 Requesting the robot position

The axis-specific and Cartesian robot position can be requested in the application. It is possible to request the actual and the setpoint position for each.

Overview

The following methods of the Robot class are available:

Method	Description
<code>getCommandedCartesianPosition(...)</code>	<p>Return value type: Frame</p> <p>Requests the Cartesian setpoint position</p>
<code>getCommandedJointPosition()</code>	<p>Return value type: JointPosition</p> <p>Requests the axis-specific setpoint position</p>
<code>getCurrentCartesianPosition(...)</code>	<p>Return value type: Frame</p> <p>Requests the Cartesian actual position</p>
<code>getCurrentJointPosition()</code>	<p>Return value type: JointPosition</p> <p>Requests the axis-specific actual position</p>
<code>getPositionInformation(...)</code>	<p>Return value type: PositionInformation</p> <p>Requests the Cartesian position information</p> <p>The return value contains the following information:</p> <ul style="list-style-type: none"> • Axis-specific actual position • Axis-specific setpoint position • Cartesian actual position • Cartesian setpoint position • Cartesian setpoint/actual value difference (rotational) • Cartesian setpoint/actual value difference (translational)

15.14.1 Requesting the axis-specific robot position

Description

For requesting the axis-specific actual or setpoint position of the robot, the position of the robot axes is first saved in a variable of type JointPosition.

From this variable, the positions of individual axes can then be requested. The axis whose position is to be requested can be specified using either its index or the Enum JointEnum.

Syntax

To request the axis-specific actual position:

```
JointPosition position = robot.getCurrentJointPosition();
```

To request the axis-specific setpoint position:

```
JointPosition position = robot.getCommandedJointPosition();
```

Requesting the position of an individual axis:

```
double value = position.get(axis);
```

Explanation of the syntax

Element	Description
<i>position</i>	Type: JointPosition Variable for the return value. The return value contains the requested axis positions.
<i>robot</i>	Type: Robot Name of the robot from which the axis positions are requested
<i>value</i>	Type: double; unit: rad Position of the requested axis
<i>axis</i>	Type: int or JointEnum Index or JointEnum of the axis whose position is requested <ul style="list-style-type: none"> • 0 ... 11: Axis A1 ... Axis A12 • JointEnum.J1 ... JointEnum.J12: Axis A1 ... Axis A12

Example

First the axis-specific actual position of the robot and then the position of axis A3 are requested via the index of the axis. The angle for axis A3 is displayed in degrees on the smartHMI. For output purposes, a logger object has been integrated with dependency injection.

```
JointPosition actPos = robot.getCurrentJointPosition();
double a3 = actPos.get(2);
logger.info("Position A3: " + Math.toDegrees(a3) + "°");
```

15.14.2 Requesting the Cartesian actual or setpoint position

Description

It is possible to request the Cartesian actual or setpoint position of the robot flange as well as any other frame below it. This means every frame of

an object which is attached to the robot flange via the attachTo command, e.g. the TCP of a tool or the frame of a gripped workpiece.

As standard, the result of the request, i.e. the Cartesian position, refers to the world coordinate system. Optionally, it is possible to specify another reference coordinate system relative to which the Cartesian position is requested. This can for example be a frame created in the application data or a calibrated base.

The result of the request is saved in a variable of type Frame and contains all the necessary redundancy information (redundancy angle, Status and Turn). From this variable, the position (X, Y, Z) and orientation (A, B, C) of the frame can be requested via the type-specific get methods.

Syntax

To request the Cartesian actual position:

```
Frame position = robot.getCurrentCartesianPosition(  
    frameOnFlange<, referenceFrame>);
```

To request the Cartesian setpoint position:

```
Frame position = robot.getCommandedCartesianPosition(  
    frameOnFlange<, referenceFrame>);
```

Explanation of the syntax

Element	Description
<i>position</i>	Type: Frame Variable for the return value. The return value contains the requested Cartesian position.
<i>robot</i>	Type: Robot Name of the robot from which the Cartesian position is requested
<i>frameOnFlange</i>	Type: ObjectFrame Robot flange or a frame subordinated to the flange whose Cartesian position is requested
<i>referenceFrame</i>	Type: AbstractFrame Reference coordinate system relative to which the Cartesian position is requested. If no reference coordinate system is specified, the Cartesian position refers to the world coordinate system.

Examples

Cartesian actual position of the robot flange with reference to the world coordinate system:

```
Frame cmdPos =  
    robot.getCurrentCartesianPosition(robot.getFlange());
```

Cartesian actual position of the TCP of a tool with reference to a base:

```
tool.attachTo(robot.getFlange());  
// ...  
Frame cmdPos =  
    robot.getCurrentCartesianPosition(tool.getFrame("/TCP"),  
    getApplicationData().getFrame("/Base"));
```

15.14.3 Requesting the Cartesian setpoint/actual value difference

Description

The Cartesian setpoint/actual value difference (= difference between the programmed and measured position) can be requested with the `getPositionInformation(...)` method.

The result of the request is saved in a variable of type `PositionInformation`. From this variable, the translational and rotational setpoint/actual value differences can be requested separately from each other.

Syntax

To request position information:

```
PositionInformation info = robot.getPositionInformation(  
    frameOnFlange<, referenceFrame>);
```

To request the translational setpoint/actual value difference:

```
Vector translatoryDiff = info.getTranslationOffset();
```

To request the rotational setpoint/actual value difference:

```
Rotation rotatoryDiff = info.getRotationOffset();
```



The Cartesian actual/setpoint value position saved in the `PositionInformation` object can be read with the methods `getCurrentCartesianPosition(...)` and `getCommandedCartesianPosition(...)` that have already been described.

Explanation of the syntax

Element	Description
<i>info</i>	Type: <code>PositionInformation</code> Variable for the return value. The return value contains the requested position information.
<i>robot</i>	Type: <code>Robot</code> Name of the robot from which the position information is requested
<i>frameOnFlange</i>	Type: <code>ObjectFrame</code> Robot flange or a frame subordinated to the flange whose position information is being requested
<i>referenceFrame</i>	Type: <code>AbstractFrame</code> Reference coordinate system relative to which the position information is requested. If no reference coordinate system is specified, the position information refers to the world coordinate system.
<i>translatory-Diff</i>	Type: <code>vector (com.kuka.roboticsAPI.geometricModel.math)</code> Translational setpoint/actual value difference in the X, Y, Z directions (type: <code>double</code> , unit: <code>mm</code>) The offset values for each degree of freedom can be requested individually with the “get” methods of the <code>Vector</code> class. (>>> 15.4 "Requesting individual values of a vector" Page 385)

Element	Description
<i>rotatoryDiff</i>	Type: rotation (com.kuka.roboticsAPI.geometricModel.math) Setpoint/actual value difference of the axis angles A, B, C (type: double, unit: rad) The offset values for each degree of freedom can be requested individually with the “get” methods of the Rotation class - getAlphaRad(), getBetaRad, getGammaRad().

Example

Reading of the translational setpoint/actual value difference in the X direction and the setpoint/actual value difference of the axis angle C.

```
tool.attachTo(robot.getFlange());
// ...
PositionInformation posInf =
robot.getPositionInformation(tool.getFrame("/TCP"),
getApplicationData().getFrame("/Base"));

Vector transDiff = posInf.getTranslationOffset();
Rotation rotDiff = posInf.getRotationOffset();

double transOffsetInX = transDiff.getX();
double rotOffsetofC = rotDiff.getGammaRad();
```

15.15 HOME position

The HOME position is an application-specific position of the robot. It can be reset for an application during initialization.

As standard, the HOME position has the following values:

Axis	A1	A2	A3	A4	A5	A6	A7
Pos.	0°	0°	0°	0°	0°	0°	0°

15.15.1 Changing the HOME position

Description

The HOME position in an application can be changed with `setHomePosition(...)`. The method belongs to the Robot class.

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.

The new HOME position can be transferred as an axis-specific or Cartesian position (frame). It is only applicable in the application in which it was changed. Other applications continue to use the HOME position with the default values.

Syntax

```
robot.setHomePosition(home);
```

Explanation of the syntax

Element	Description
<i>robot</i>	Type: Robot Name of the robot to which the new HOME position refers
<i>home</i>	Type: JointPosition; unit: rad 1st option: transfer the axis position of the robot in the new HOME position. Type: AbstractFrame 2nd option: transfer a frame as the new HOME position. Note: The frame must contain all redundancy information so that the axis positions of the robot in the HOME position are unambiguous. This is the case with a taught frame, for example.

Examples

To transfer an axis-specific position as the HOME position:

```
@Inject
private LBR robot;
// ...
JointPosition newHome = new JointPosition(0.0, 0.0, 0.0,
Math.toRadians(90), 0.0, 0.0, 0.0);
robot.setHomePosition(newHome);
```

To transfer the taught frame as the HOME position and move to it with `ptpHome()`:

```
@Inject
private LBR robot;
// ...
ObjectFrame newHome = getApplicationData().getFrame("/Homepos");
robot.setHomePosition(newHome);
robot.moveAsync(ptpHome());
```

15.16 Requesting system states

Different system states can be requested from the robot and processed in the application. The requesting of system states is primarily required when using a higher-level controller so that the controller can react to changes in state.

15.16.1 Requesting the HOME position

Description

The following methods of the Robot class are available for requesting the HOME position:

- `getHomePosition()`
Requests the HOME position currently defined for the robot
- `isInHome()`
Checks whether the robot is currently in the HOME position

Syntax

To request the HOME position:

```
JointPosition homePos = robot.getHomePosition();
```

To check whether the robot is currently in the HOME position:

```
boolean result = robot.isInHome();
```

Explanation of the syntax

Element	Description
<i>homePos</i>	Type: JointPosition Variable for the return value of getHomePosition(). The return value contains axis angles of the requested HOME position.
<i>robot</i>	Type: robot Name of the robot from which the HOME position is requested
<i>result</i>	Type: boolean Variable for the return value of isInHome(). The return value is true when the robot is in the HOME position.

Example

As long as the robot is not yet in the HOME position, a certain statement block is to be executed.

```
@Inject
private LBR robot;
// ...
while (! robot.isInHome()) {
    //do something
}
```

15.16.2 Requesting the mastering state

Description

The method `isMastered()` is available for requesting the mastering state. The method belongs to the Robot class.

Syntax

```
boolean result = robot.isMastered();
```

Explanation of the syntax

Element	Description
<i>robot</i>	Type: Robot Name of the robot whose mastering state is requested
<i>result</i>	Type: Boolean Variable for the return value <ul style="list-style-type: none"> • true: All axes are mastered. • false: One or more axes are unmastered.

15.16.3 Checking “ready for motion”

Description

The method `isReadyToMove()` is available for checking whether the robot is ready for motion. The method belongs to the `Robot` class. It returns the value “true” if the robot is ready to move.

The robot is ready to move if the following conditions are met:

- No safety stop is active.
- The drives are in an error-free state.
- Automatic mode is set.

OR:

In mode T1 or T2, the enabling signal is issued via the smartPAD (enabling switch in center position).



When the enabling signal for manual guidance is issued (enabling switch on the hand guiding device in the center position), the `isReadyToMove()` method returns the value “false”. The robot is not ready to move because it can be guided manually and is already being moved.



If the check returns the value “true”, this does not necessarily mean that the brakes are open and that the robot is under active servo control.

Syntax

```
boolean result = robot.isReadyToMove();
```

Explanation of the syntax

Element	Description
<code>robot</code>	Type: Robot Name of the robot which is checked as to whether it is ready for motion
<code>result</code>	Type: boolean Variable for the return value <ul style="list-style-type: none"> • true: Robot is ready for motion. • false: Robot is not ready for motion.

15.16.3.1 Reacting to changes in the “ready for motion” signal

Description

There is a notification service of the `Controller` class in `RoboticsAPI` which reports changes in the “ready for motion” signal. To register for the service, transfer an `IControllerStateListener` object to the `Controller` attribute in the robot application. The method `addControllerListener(...)` is used for this purpose.

The method `onIsReadyToMoveChanged(...)` is called every time the “ready to move” signal changes. The reaction to the change can be programmed in the body of the method `onIsReadyToMoveChanged(...)`.

Syntax

```
kuka_Sunrise_Cabinet.addControllerListener(new
IControllerStateListener() {
```

```

...
@Override
public void onIsReadyToMoveChanged(Device device,
boolean isReadyToMove) {
    // Reaction to change
}
...
} );

```

Explanation of the syntax

Element	Description
<i>kuka_Sunrise_Cabinet</i>	Type: Controller Controller attribute of the robot application (= name of the robot controller in the application)

15.16.4 Checking the robot activity

Description

A robot is active if a motion command is active. This affects both motion commands from the application and jogging commands.

The `hasActiveMotionCommand()` method is available for checking whether the robot is active. The method belongs to the `Robot` class.

The request does not provide any information as to whether the robot is currently in motion:

- If the request returns the value “false” (no motion command active), this does not necessarily mean that the robot is stationary. For example, robot activity may be checked directly after a synchronous motion command with a break condition. If the break condition occurs, the check supplies the value “false” if the robot is braked and moving.
- If the request returns the value “true” (motion command active), this does not necessarily mean that the robot is moving. For example, the request returns the value “true” if a robot executes the motion command `positionHold(...)` and is stationary.

Syntax

```
boolean result = robot.hasActiveMotionCommand();
```

Explanation of the syntax

Element	Description
<i>robot</i>	Type: Robot Identifier for the robot whose activity is checked
<i>result</i>	Type: boolean Variable for the return value <ul style="list-style-type: none"> • true: A motion command is active. • false: No motion command is active.

15.16.5 Requesting the state of safety signals

Description

The state of the following safety signals can be requested and evaluated in an application:

- Active operating mode
- Enabling
- Local EMERGENCY STOP
- External EMERGENCY STOP
- “Operator safety” signal
- Stop request (safety stop)
- Referencing state of position and joint torque sensors

The state of the different safety signals is first requested via the method `getSafetyState()` and grouped in an object of type `ISafetyState`.

From this object, the states of individual safety signals can then be requested. The interface `ISafetyState` contains the methods required for this.

Syntax

```
ISafetyState currentState = kinematics.getSafetyState();
```

Explanation of the syntax

Element	Description
<code>currentState</code>	Type: <code>ISafetyState</code> Variable for the return value. The return value contains the state of the safety signals at the time of requesting with <code>getSafetyState()</code> . Note: This does not apply to the referencing states. Referencing states are not requested until the corresponding methods of the <code>ISafetyState</code> object are called.
<code>kinematics</code>	Type: <code>MovableDevice</code> Kinematic system for which the state of the safety signals is requested

Precondition

The EMERGENCY STOP signal and the “Operator Safety” signal can only be evaluated if the following conditions are met in the safety configuration:

- The selected category matches the safety function:
 - Category *Local EMERGENCY STOP* for local EMERGENCY STOP
 - Category *External EMERGENCY STOP* for external EMERGENCY STOP
 - Category *Operator safety* for operator safety
- The configured reaction is a safety stop (no output).

Overview

Methods of the `ISafetyState` interface

The implementing class of the interface is `SunriseSafetyState` (package: `com.kuka.roboticsAPI.controllerModel.sunrise`).

Method	Description
getEmergencyStopInt()	<p>Return value type: Enum of type EmergencyStop</p> <p>Checks whether a local E-STOP is activated.</p> <ul style="list-style-type: none"> • ACTIVE: Local E-STOP is activated. • INACTIVE: Local E-STOP is not activated. • NOT_CONFIGURED: Not relevant, as a local E-STOP is always configured.
getEmergencyStopEx()	<p>Return value type: Enum of type EmergencyStop</p> <p>Checks whether an external E-STOP is activated.</p> <ul style="list-style-type: none"> • ACTIVE: External E-STOP is activated. • INACTIVE: External E-STOP is not activated. • NOT_CONFIGURED: No external EMERGENCY STOP is configured.
getEnablingDeviceState()	<p>Return value type: Enum of type EnablingDeviceState</p> <p>Checks whether an enabling switch is pressed.</p> <ul style="list-style-type: none"> • HANDGUIDING: Enabling switch on the hand guiding device is pressed. • NORMAL: Enabling switch on the smartPAD is pressed. • NONE: No enabling switch is pressed or a safety function has been violated and is blocking motion enable.
getOperationMode()	<p>Return value type: Enum of type OperationMode (package: com.kuka.roboticsAPI.deviceModel)</p> <p>Checks which operating mode is active.</p> <ul style="list-style-type: none"> • T1, T2, AUT, CRR
getOperatorSafetyState()	<p>Return value type: Enum of type OperatorSafety</p> <p>Checks the “Operator safety” signal.</p> <ul style="list-style-type: none"> • OPERATOR_SAFETY_OPEN: Operator safety is violated (e.g. safety gate is open). • OPERATOR_SAFETY_CLOSED: Operator safety is not violated. • NOT_CONFIGURED: No operator safety is configured.
getSafetyStopSignal()	<p>Return value type: Enum of type SafetyStopType</p> <p>Checks whether a safety stop is activated.</p> <ul style="list-style-type: none"> • NOSTOP: No safety stop is activated. • STOP0: A safety stop 0 or a safety stop 1 is activated. • STOP1: A safety stop 1 (path-maintaining) is activated. • STOP2: This value is currently not returned.

The methods for requesting the referencing state are described here:
 (>>> [15.16.5.1 "Requesting the referencing state" Page 435](#))

Example

The system checks whether a safety stop is activated. If this is the case, the operator safety is then checked. If this is violated, a message is displayed on the smartHMI. For output purposes, a logger object has been integrated with dependency injection.

```

ISafetyState safetyState = robot.getSafetyState();

SafetyStopType safetyStop = safetyState.getSafetyStopSignal();

if (safetyStop != SafetyStopType.NOSTOP) {
    OperatorSafety operatorSafety = safetyState.getOperatorSafetyState();
    if (operatorSafety == OperatorSafety.OPERATOR_SAFETY_OPEN) {
        logger.warn("The safety gate is open!");
    }
}

```

15.16.5.1 Requesting the referencing state

Description

An LBR has position and joint torque sensors that can be referenced. The referencing state of these sensors can be requested by the robot, e.g. to check whether referencing needs to be carried out again.

If a robot has no position or joint torque sensors that can be referenced, the request returns the value “false”.

Method	Description
isAxisGMSReferenced(...)	<p>Return type: Boolean</p> <p>Checks whether the joint torque sensor of a specific robot axis is referenced. The axis to be checked is transferred as a parameter (type: JointEnum).</p> <ul style="list-style-type: none"> • true: Joint torque sensor of the axis is referenced. • false: Joint torque sensor of the axis is not referenced or the robot has no joint torque sensors that can be referenced. <p>If an invalid axis is transferred, i.e. an axis that is not present on the robot, an Illegal Argument Exception is triggered.</p>
areAllAxesGMSReferenced()	<p>Return type: Boolean</p> <p>Checks whether all joint torque sensors of the robot are referenced.</p> <ul style="list-style-type: none"> • true: All joint torque sensors are referenced. • false: At least 1 joint torque sensor is not referenced or the robot has no joint torque sensors that can be referenced.
isAxisPositionReferenced(...)	<p>Return type: Boolean</p> <p>Checks whether the position sensor of a specific robot axis is referenced. The axis to be checked is transferred as a parameter (type: JointEnum).</p> <ul style="list-style-type: none"> • true: Position sensor of the axis is referenced. • false: Position sensor of the axis is not referenced or the robot has no position sensors that can be referenced. <p>If an invalid axis is transferred, i.e. an axis that is not present on the robot, an Illegal Argument Exception is triggered.</p>

Method	Description
areAllAxesPositionReferenced()	<p>Return type: Boolean</p> <p>Checks whether all position sensors of the robot are referenced.</p> <ul style="list-style-type: none"> • true: All position sensors are referenced. • false: At least 1 position sensor is not referenced or the robot has no position sensors that can be referenced.

Example

Checking whether the position sensor of axis A1 is referenced

```
boolean isReferencedJ1 =
robot.getSafetyState().isAxisPositionReferenced(JointEnum.J1)
;
```

15.16.5.2 Reacting to a change in state of safety signals

Description

There is a notification service of the Controller class in RoboticsAPI which reports changes in the state of safety signals. This service enables a direct reaction to the change in a signal state.

To register for the service, transfer an ISunriseControllerStateListener object to the Controller attribute in the robot application. The method addControllerListener(...) is used for this purpose.

The method onSafetyStateChanged(...) is called every time the state of a safety signal changes. The reaction to the change can be programmed in the body of the method onSafetyStateChanged(...).

Syntax

```
kuka_Sunrise_Cabinet.addControllerListener(new
ISunriseControllerStateListener() {
// ...
@Override
public void onSafetyStateChanged(Device device,
SunriseSafetyState safetyState) {
// Reaction to change in state
}
});
```

Explanation of the syntax

Element	Description
kuka_Sunrise_Cabinet	Type: Controller Controller attribute of the robot application (= name of the robot controller in the application)

Example

If the state of a safety signal changes, the operator safety is checked via the method onSafetyStateChanged()(...). If this is violated, a message is displayed on the smartHMI. For output purposes, a logger object has been integrated with dependency injection.

```

kuka_Sunrise_Cabinet.addControllerListener(new
ISunriseControllerStateListener() {

    // ...

    @Override
    public void onSafetyStateChanged(Device device, SunriseSafetyState
        safetyState) {
        OperatorSafety operatorSafety = safetyState.getOperatorSafetyState();
        if (operatorSafety == OperatorSafety.OPERATOR_SAFETY_OPEN) {
            logger.warn("The saftey gate is open!");
        }
    }
});

```

15.17 Changing and requesting the program run mode

Description

The program run mode can be changed and requested via the methods `setExecutionMode(...)` and `getExecutionMode()` of the `SunriseExecutionService`. The `SunriseExecutionService` itself is requested by the Controller.

Preparation

1. Variable of type `SunriseExecutionService`.
2. Request the `SunriseExecutionService` via the method `getExecutionService()` and save in the variable.

Syntax

To change the program run mode:

```
service.setExecutionMode(ExecutionMode.newMode);
```

To request the current program run mode:

```
currentMode = service.getExecutionMode();
```

Explanation of the syntax

Element	Description
<code>service:</code>	Type: <code>SunriseExecutionService</code> Variable for the return value (contains the <code>SunriseExecutionService</code> rerequested by the Controller)
<code>newMode</code>	Type: <code>Enum</code> of type <code>ExecutionMode</code> New program run mode <ul style="list-style-type: none"> • ExecutionMode.Step: Step mode (program sequence with a stop after each motion command) • ExecutionMode.Continuous: Standard mode (continuous program sequence without stops)
<code>currentMode</code>	Type: <code>ExecutionMode</code> Variable for the return value (contains the program run mode requested by the <code>SunriseExecutionService</code>)

Example

The `SunriseExecutionService` is requested by the Controller and saved in the variable "serv".

```

@Inject
private Controller controller;
private SunriseExecutionService serv;
// ...

@Override
public void initialize() {
    // ...
    serv = (SunriseExecutionService) controller
        .getExecutionService();
    // ...
}

```

The system first switches to Step mode and then back to standard mode.

```

@Override
public void run() {
    // ...
    serv.setExecutionMode(ExecutionMode.Step);
    // ...
    serv.setExecutionMode(ExecutionMode.Continuous);
    // ...
}

```

The current program run mode is requested.

```

@Override
public void run() {
    // ...
    ExecutionMode currentMode;
    currentMode = serv.getExecutionMode();
    // ...
}

```

15.18 Changing and requesting the override

The interface IApplicationOverrideControl provides methods with which the current override can be requested or changed in the application. For this, the interface IApplicationControl must be accessed in the first step using the method getApplicationControl().

The following override types are distinguished:

- Manual override: Override which can be adjusted manually by the user via the smartPAD
(>>> [6.19.3 "Setting the manual override" Page 118](#))
- Application override: Programmed override set by the application
- Effective program override: Product of the manual override and application override

Overview

Methods used for requesting the current override:

Method	Description
getApplicationOverride()	Return value type: double Requests the application override
getManualOverride()	Return value type: double Requests the manual override

Method	Description
getEffectiveOverride()	Return value type: double Requests the effective program override

Methods used for changing the override:

Method	Description
setApplicationOverride(...)	Sets the application override to the specified value (type: double) • 0 ... 1
clipApplicationOverride(...)	Reduces the application override to the specified value (type: double) • 0 ... 1 If a value is specified that is higher than the value currently programmed for the application override, the statement clipApplicationOverride(...) is ignored.
clipManualOverride(...)	Reduces the manual override to the specified value (type: double) • 0 ... 1 If a value is specified that is higher than the currently programmed manual override, the statement clipManualOverride(...) is ignored.

Example

```
getApplicationControl().setApplicationOverride(0.5);
// ...
double actualOverride =
getApplicationControl().getEffectiveOverride();
```

15.18.1 Reacting to an override change

Description

It is possible for an application to inform itself when an override changes. A listener of type IApplicationOverrideListener must be defined and registered for this purpose.

When changing an override, the method onOverrideChanged(...) is called. The reaction to the change can be programmed in the body of the method onOverrideChanged(...).

Syntax

Defining a listener:

```
IApplicationOverrideListener overrideListener =
new IApplicationOverrideListener() {
    @Override
    public void onOverrideChanged(double effectiveOverride,
        double manualOverride, double applicationOverride) {
        // Reaction to override change
    }
};
```

Registering a listener:

```
getApplicationControl() .  
addOverrideListener(overrideListener) ;
```

Removing a listener:

```
getApplicationControl() .  
removeOverrideListener(overrideListener) ;
```

Explanation of the syntax

Element	Description
<i>override Listener</i>	Type: IApplicationOverrideListener Name of the listener

15.19 Overview of conditions

Often, values are to be monitored in applications and if definable limits are exceeded or not reached, specific reactions are to be triggered. Possible sources for these values include the sensors of the robot or configured inputs. The progress of a motion can also be monitored. Possible reactions are the termination of a motion being executed or the execution of a handling routine.

A condition can have 2 states: it is met (state = TRUE) or not met (state = FALSE). To define a condition, an expression is formulated. In this expression, data, such as measurements provided by the system, are compared with a permissible limit value. The result of the evaluation of the expression defines the state of the condition.

Since different system data can be used for formulating conditions, there are different kinds of conditions. Each condition type is made available as its own class in the RoboticsAPI. They belong to the com.kuka.roboticsAPI.conditionModel package and implement the ICondition interface.

Some system data, e.g. axis torques or Cartesian forces and torques on the robot flange, are only available for sensitive robot types equipped with corresponding sensor systems. These sensitive robot types include, for example, the LBR. Condition types using forces or torques are only supported by these sensitive robot types. If these condition types are applied to robots that do not provide information about forces or torques, this results in a runtime error (Exception).

Categories

The various condition types can be subdivided into the following categories:

- Sensor-related conditions
- Path-related conditions
- Distance-related conditions
- I/O-related conditions

Sensor-related conditions:

Data type	Description
JointTorqueCondition	The axis torque condition is met if the torque measured in an axis lies outside of a defined range of values. (>>> 15.19.2 "Axis torque condition" Page 443)
ForceCondition	The force condition is met if the Cartesian force exerted on a frame below the robot flange (e.g. at the TCP) exceeds a defined magnitude. (>>> 15.19.3 "Force condition" Page 444)
ForceComponentCondition	The force component condition is met if the Cartesian force exerted along an axis of a frame below the robot flange (e.g. along an axis of the TCP) exceeds a defined range. (>>> 15.19.4 "Force component condition" Page 450)
CartesianTorqueCondition	The conditions for the Cartesian torque is met if the Cartesian torque acting about the axis of a frame below the robot flange (e.g. about the axis of the TCP) exceeds a defined value. (>>> 15.19.5 "Condition for Cartesian torque" Page 452)
TorqueComponentCondition	The torque component condition is met if the Cartesian torque exerted about an axis of a frame below the robot flange (e.g. about an axis of the TCP) is outside a defined range. (>>> 15.19.6 "Torque component condition" Page 457)

Path-related conditions:

Data type	Description
MotionPathCondition	The path-related condition is met if a defined distance on the planned path, from the start or end point of the motion, is reached. In addition, it is possible to define a time delay which must be met. (>>> 15.19.7 "Path-related condition" Page 458)

Distance-related conditions:

Data type	Description
FrameDistanceCondition	The distance condition is met if the Cartesian distance between 2 frames is less than a defined distance, e.g. the distance between a TCP or gripped workpiece and a defined, fixed reference point. (>>> 15.19.8 "Distance condition" Page 461)
FrameDistanceComponent Condition	The distance component condition is met if the Cartesian distance between 2 frames relative to the specified axes of an orientation frame is less than a defined distance. (>>> 15.19.8.1 "Distance component condition" Page 462)

I/O-related conditions:

Data type	Description
BooleanIOCondition	The condition for Boolean signals is met if a Boolean digital input or output has a specific state. (>>> 15.19.9 "Condition for Boolean signals" Page 463)
IORangeCondition	The condition for the value range of a signal is met if the value of an analog or digital input or output lies within a defined range. (>>> 15.19.10 "Condition for the range of values of a signal" Page 463)

Areas of application

- Abortion of motions

A motion is terminated as soon as a specific event occurs. The event occurs if the condition already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

(>>> [15.20 "Break conditions for motion commands" Page 464](#))

- Path-related switching actions (trigger)

An action is triggered as soon as a specific event occurs. The event occurs if the condition already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

(>>> [15.21 "Path-related switching actions \(trigger\)" Page 469](#))

- Monitoring of processes

The state of a condition is checked cyclically using a listener. If the state of the condition changes, it is possible to react.

(>>> [15.22 "Monitoring of processes" Page 473](#))

- Blocking wait for condition

An application is stopped until a certain condition is met or a certain wait time has expired.

(>>> [15.23 "Blocking wait for condition" Page 478](#))

15.19.1 Complex conditions

Conditions can be logically linked to one another so that it is possible to define complex conditions. The logic operators required for this are available as ICondition methods. The calling ICondition object is linked to one or more conditions, which are transferred as parameters.

The operators can be called several times in a row and in this way, parentheses and nesting of operations can be realized. The evaluation is thus dependent on the order of calling.

Operators

Operator	Description/syntax
NOT	Inversion of the calling ICondition object <code>ICondition invert();</code>
XOR	EITHER/OR operation linking the calling ICondition object with a further condition <code>ICondition xor(ICondition other);</code> <code>other: further condition</code>

Operator	Description/syntax
AND	AND operation linking the calling ICondition object with one or more additional conditions <code>ICondition and(ICondition <i>other1</i>, ICondition <i>other2</i>, ...);</code> <i>other1, other2, ...</i> : further conditions
OR	OR operation linking the calling ICondition object with one or more additional conditions <code>ICondition or(ICondition <i>other1</i>, ICondition <i>other2</i>, ...);</code> <i>other1, other2, ...</i> : further conditions

Example

```

JointTorqueCondition condA = ...;
JointTorqueCondition condB = ...;
JointTorqueCondition condC = ...;
JointTorqueCondition condD = ...;

ICondition combi1, combi2, combi3, combi4;

// NOT A
combi1 = condA.invert();

// A AND B AND C
combi2 = condA.and(condB, condC);

// (A OR B) AND C
combi3 = condA.or(condB).and(condC);

// (A OR B) AND (C OR D)
combi4 = condA.or(condB).and(condC.or(condD));

```

15.19.2 Axis torque condition

Description

The axis torque condition is used to check whether the external torque determined in an axis lies outside of a defined range of values.

(>>> [15.12 "Requesting axis torques" Page 419](#))



The load data must be specified correctly during programming. Only then is the condition usefully applicable.



This condition is only supported by sensitive robots, e.g. a robot of type LBR. If the condition is applied to a different robot that does not provide the required sensor information, this results in a runtime error (Exception).

Constructor syntax

```
JointTorqueCondition(JointEnum joint, double minTorque,  
double maxTorque)
```

Explanation of the syntax

Element	Description
<i>joint</i>	Axis whose torque value is to be checked
<i>minTorque</i>	Lower limit value for the axis torque (unit: Nm) The condition is met if the torque is less than or equal to <i>minTorque</i> .
<i>maxTorque</i>	Upper limit value for the axis torque (unit: Nm) The condition is met if the torque is greater than or equal to <i>minTorque</i> .

The following must apply when determining the upper and lower limit values for the torque: $\text{minTorque} \leq \text{maxTorque}$.

Example

The condition is met if a torque value of ≤ -2.5 Nm or ≥ 4.0 Nm is measured in axis J3.

```
JointTorqueCondition torqueCondJ3 =
    new JointTorqueCondition(JointEnum.J3, -2.5, 4.0);
```

15.19.3 Force condition

Description

The force condition can be used to check whether a Cartesian force exerted on a frame below the robot flange exceeds a defined limit value.

For example, it is possible to react to the force generated when the robot presses on a surface using a tool mounted on the flange. For the force condition, the projections of the force vector exerted on a frame below the flange are considered. The position of this frame is defined by the point of application of the force (here the tool tip). The orientation of the frame should correspond to the orientation of the surface.

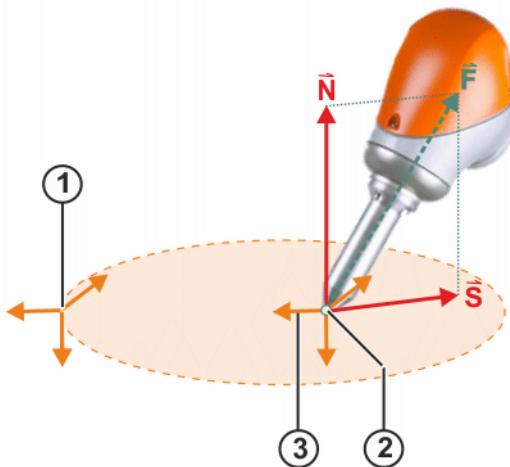


Fig. 15-18: Force vectors

- 1 Frame which specifies the orientation of the reference frame (here: orientation of the surface)
- 2 Point of application of the force, here the tip of the tool

- 3 Reference frame below the flange onto which the force vector is projected. The position of the frame corresponds to the point of application of the force. The orientation corresponds to the orientation of the surface.

The following force vectors are relevant:

- Normal force N:

The normal force is the projection of the force exerted on the surface normal (= vector which is perpendicular to the surface). This results in the part of the force exerted vertically on the surface. For example, pressure is exerted via the normal force in order to fit a component.

- Shear force S:

The shear force is the projection of the force exerted on the surface. This results in the part of the force exerted parallel to the surface. The shear force is generated by friction.



The load data must be specified correctly during programming. Only then is the condition usefully applicable.



The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the force component condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.



This condition is only supported by sensitive robots, e.g. a robot of type LBR. If the condition is applied to a different robot that does not provide the required sensor information, this results in a runtime error (Exception).

Methods

Force conditions are of the data type ForceCondition. ForceCondition contains the following static methods for programming conditions:

- `createSpatialForceCondition(...)`: Condition for Cartesian force from all directions
- `createNormalForceCondition(...)`: Condition for normal force
- `createShearForceCondition(...)`: Condition for shear force

To formulate the condition, a frame below the flange coordinate system (e.g. the tip of a tool) is defined as a reference system. The forces which are exerted relative to this frame are determined. The orientation of the reference system can be optionally defined via an orientation frame. This can be used, for example, to define the position of the surface on which the force is exerted.

A limit value is defined to determine the minimum force magnitude which meets the condition.

The Cartesian force is calculated from the measured values of the axis torque sensors. The reliability of the calculated force values varies depending on the axis configuration. If the quality of the force calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the force condition is also met.

15.19.3.1 Condition for Cartesian force from all directions

Description

The static method `createSpatialForceCondition(...)` is used to define a condition which is valid regardless of the direction from which the Cartesian force is exerted on a frame below the flange.

Syntax

```
ForceCondition.createSpatialForceCondition(
AbstractFrame measureFrame<, AbstractFrame
orientationFrame>, double threshold<, double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measureFrame</i>	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
<i>orientationFrame</i>	Optional. The orientation of the reference system is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.
<i>threshold</i>	Maximum magnitude of force which may act on the reference system (unit: N). <ul style="list-style-type: none"> • ≥ 0.0 The condition is met if the magnitude of force exerted on the reference system from any direction exceeds the value specified here.
<i>tolerance</i>	Optional. Maximum permissible inaccuracy of the calculated values (unit: N). <ul style="list-style-type: none"> • > 0.0 Default: 10.0 The condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

Example

The condition is met as soon as the magnitude of the force acting from any direction on the TCP of a tool exceeds 30 N.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private Tool gripper;
    // ...

    @Override
    public void initialize() {
        // ...
    }
}
```

```

        gripper.attachTo(robot.getFlange());
        // ...
    }

@Override
public void run() {
    // ...
    ForceCondition spatialForce_tcp = ForceCondition
        .createSpatialForceCondition(
            gripper.getFrame("/TCP"),
            30.0);
    // ...
}
}

```

15.19.3.2 Condition for normal force

Description

A condition for the normal force can be defined via the static method `createNormalForceCondition(...)`. The component of the force exerted along a definable axis of a frame below the flange (e.g. along an axis of the TCP) is considered here. This axis is generally defined so that it is perpendicular to the surface on which the force is exerted (surface normal).

Syntax

```
ForceCondition.createNormalForceCondition(AbstractFrame
measureFrame<, AbstractFrame orientationFrame>,
CoordinateAxis direction, double threshold<,
double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measure Frame</i>	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
<i>orientation Frame</i>	Optional. The orientation of the reference system is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.
<i>direction</i>	Coordinate axis of the reference system. The force component acting on the axis specified here is checked with the condition. <ul style="list-style-type: none"> • CoordinateAxis.X • CoordinateAxis.Y • CoordinateAxis.Z
<i>threshold</i>	Maximum magnitude of force which may act along the axis of the reference system (unit: N). <ul style="list-style-type: none"> • ≥ 0.0 The condition is met if the magnitude of force exceeds the value specified here.

Element	Description
<i>tolerance</i>	<p>Optional. Maximum permissible inaccuracy of the calculated values (unit: N).</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 10.0</p> <p>The condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here.</p> <p>If the parameter is not specified, the default value is automatically used.</p>

Example

A gripper mounted on the flange presses on a table plate. The robot is to react to that part of the force exerted at the TCP of the gripper which acts vertically on the table plate. The reference system is therefore defined such that its Z axis runs along the surface normal of the table plate.

The condition is met as soon as the normal force exceeds a magnitude of 45 N. The condition is also to be considered met if the inaccuracy value of the calculated data exceeds 8.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private Tool gripper;
    // ...

    @Override
    public void initialize() {
        // ...
        gripper.attachTo(robot.getFlange());
        // ...
    }

    @Override
    public void run() {
        // ...
        ForceCondition normalForce_z = ForceCondition
            .createNormalForceCondition(
                gripper.getFrame("/TCP"),
                getFrame("/Table/Edge/Tabletop"),
                CoordinateAxis.Z,
                45.0,
                8.0);
        // ...
    }
}
```

15.19.3.3 Condition for shear force

Description

A condition for the shear force can be defined via the static method `createShearForceCondition(...)`. The component of the force acting parallel to a plane is considered here. The position of the plane is determined by specifying the axis which is vertical to the plane.

Syntax

```
ForceCondition.createShearForceCondition(AbstractFrame
measureFrame<, AbstractFrame orientationFrame>,
CoordinateAxis normalDirection, double threshold<,
double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measureFrame</i>	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
<i>orientationFrame</i>	Optional. The orientation of the reference system is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.
<i>normalDirection</i>	Coordinate axis of the reference system. The axis specified here defines the surface normal of a plane. The force component acting parallel to this plane is checked. <ul style="list-style-type: none"> • CoordinateAxis.X • CoordinateAxis.Y • CoordinateAxis.Z
<i>threshold</i>	Maximum magnitude of force which may be exerted parallel to the reference system plane defined by its surface normal (unit: N). <ul style="list-style-type: none"> • ≥ 0.0 The condition is met if the magnitude of force exceeds the value specified here.
<i>tolerance</i>	Optional. Maximum permissible inaccuracy of the calculated values (unit: N). <ul style="list-style-type: none"> • > 0.0 Default: 10.0 The condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

Example

A gripper mounted on the flange presses on a table plate. The force at the TCP of the gripper is to be determined using the orientation of the table plate. This process considers the shear force which acts parallel to the XY plane of the measurement point, defined by the TCP and the position of the table.

To define the XY plane, the axis perpendicular to this plane must be specified as a parameter. This is the Z axis.

The condition is met as soon as the shear force exceeds a magnitude of 25 N. The condition is also to be considered met if the inaccuracy value of the calculated data exceeds 5.

```
public class ExampleApplication extends  
RoboticsAPIApplication {  
    @Inject  
    private LBR robot;  
    @Inject  
    private Tool gripper;  
    // ...  
  
    @Override  
    public void initialize() {  
        // ...  
        gripper.attachTo(robot.getFlange());  
        // ...  
    }  
  
    @Override  
    public void run() {  
        // ...  
        ForceCondition shearForce_xyPlane = ForceCondition  
            .createShearForceCondition(  
                gripper.getFrame("/TCP"),  
                getFrame("/Table/Edge/Tabletop"),  
                CoordinateAxis.Z,  
                25.0,  
                5.0);  
        // ...  
    }  
}
```

15.19.4 Force component condition

Description

The force component condition can be used to check whether the Cartesian force exerted on a frame below the robot flange (e.g. at the TCP) in the X, Y or Z direction exceeds a defined range.



The load data must be specified correctly during programming. Only then is the condition usefully applicable.



The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the force component condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.



This condition is only supported by sensitive robots, e.g. a robot of type LBR. If the condition is applied to a different robot that does not provide the required sensor information, this results in a runtime error (Exception).

The force component condition belongs to the ForceComponentCondition class. For the force component condition, a frame below the flange coordinate system is defined as a reference system. The force is determined at this frame, e.g. at the tip of a tool. The orientation of the reference system can be optionally defined via an orientation frame.

The direction from which the force is checked is defined with one of the coordinate axes of the reference system. The force component condition is met if the Cartesian force along the defined coordinate axis of the reference system lies outside of a definable range of values.

The Cartesian force is calculated from the values of the joint torque sensors. The reliability of the calculated force values varies depending on the axis configuration. If the quality of the force calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the force component condition is also met.

Constructor syntax

The ForceComponentCondition class has several constructors which differ in their number of input parameters:

```
ForceComponentCondition(AbstractFrame measureFrame
<, AbstractFrame orientationFramecoordinateAxis, double min, double max<, double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measureFrame</i>	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
<i>orientationFrame</i>	Optional: The orientation of the reference system is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.
<i>coordinateAxis</i>	Coordinate axis of the frame relative to which the exerted force is determined. Defines the direction from which the acting force is checked. <ul style="list-style-type: none"> • CoordinateAxis.X • CoordinateAxis.Y • CoordinateAxis.Z
<i>min</i>	Lower limit of the range of values for the force exerted along the coordinate axis of the reference system (unit: N). The force component condition is met if the force falls below the value specified here.
<i>max</i>	Upper limit of the range of values for the force exerted along the coordinate axis of the reference system (unit: N). The force component condition is met if the force exceeds the value specified here. Note: The upper limit value must be greater than the lower limit value: <i>max > min</i> .

Element	Description
<i>tolerance</i>	<p>Optional: Maximum permissible inaccuracy of the calculated values.</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 10.0</p> <p>The force component condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here.</p> <p>If the parameter is not specified, the default value is automatically used.</p>

Example

A joining process is ideally executed with a force of between 20 N and 25 N. A force component condition is to be defined, and is met if the force acting in the Z direction at the free end of a gripped workpiece is between 20 N and 25 N.

To this end, a force component condition is first defined which has the status FALSE in this range of values. The desired result is then realized by inversion.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private Tool gripper;
    @Inject
    @Named("Bolt")
    private Workpiece bolt;
    // ...

    @Override
    public void run() {
        // ...
        bolt.attachTo(gripper.getFrame("/Root"));

        ForceComponentCondition assemblyForce_inverted =
            new ForceComponentCondition(
                bolt.getFrame("/Assembly"),
                CoordinateAxis.Z,
                20.0,
                25.0);

        ICondition assemblyForce =
            assemblyForce_inverted.invert();
        // ...
    }
}
```

15.19.5 Condition for Cartesian torque

Description

The condition can be used to check whether a Cartesian torque exerted on a frame below the robot flange exceeds a defined limit value. The

point of application of the torque is specified by means of a frame below the robot flange coordinate system.

One application for this condition is the monitoring of torques that occur in a screw fastening process.

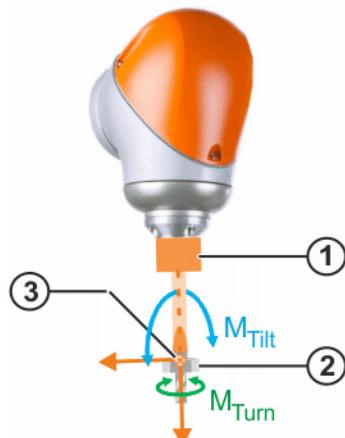


Fig. 15-19: Torque vectors in the screw fastening process

- 1 Power wrench
- 2 Screw
- 3 Reference frame, here the tip of the power wrench

The condition for the Cartesian torque can be used to check different projections of the torque vector acting on the axes of the reference frame:

- Torque M_{Turn}
The torque exerted about an axis arises from the projection of the torque vector on this axis.
- Tilting torque M_{Tilt}
The tilting torque arises from the projection of the torque vector on a plane.

The torque is applied about the longitudinal axis of the power wrench during a screw fastening process in order to screw in the screw. If the condition for the torque is used, it is possible to ensure that the maximum permissible values are not exceeded when fastening screws.

The tilting torque arises during a screw fastening process as a result of undesired tilting of the power wrench about the longitudinal axis, forwards or to the side. If the condition for the tilting torque is configured, it is possible to check whether the tilting torque is within an acceptable range of values.



The load data must be specified correctly during programming. Only then is the condition usefully applicable.



The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the force component condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.



This condition is only supported by sensitive robots, e.g. a robot of type LBR. If the condition is applied to a different robot that does not provide the required sensor information, this results in a runtime error (Exception).

Methods

Conditions for the Cartesian torque are of data type `CartesianTorqueCondition`. `CartesianTorqueCondition` contains the following static methods for programming conditions:

- `createSpatialTorqueCondition(...)`: Condition for Cartesian torque from all directions
- `createTurningTorqueCondition(...)`: Condition for torque
- `createTiltingTorqueCondition(...)`: Condition for tilting torque

To formulate the condition, a frame is defined as a reference system below the flange coordinate system. The torque is determined at this frame, e.g. at the tip of a power wrench. The orientation of the reference system can be optionally defined via an orientation frame. In this way, the desired orientation of the screw can be specified, for example.

A limit value is defined to determine the minimum Cartesian torque magnitude which meets the condition.

The Cartesian torque is calculated from the measured values of the axis torque sensors. The reliability of the calculated Cartesian torques varies depending on the axis configuration. If the quality of the calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the condition for the Cartesian torque is also met.

15.19.5.1 Condition for Cartesian torque from all directions

Description

The static method `createSpatialTorqueCondition(...)` is used to define a condition which is valid regardless of the direction from which the Cartesian torque is exerted on a frame below the flange.

Syntax

```
CartesianTorqueCondition.createSpatialTorqueCondition(  
AbstractFrame measureFrame<, AbstractFrame  
orientationFrame>, double threshold<, double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measureFrame</i>	Frame below the robot flange at which the exerted torque is determined. The position of the point of application of the torque is defined using this parameter.
<i>orientationFrame</i>	Optional: The orientation of the reference system is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.
<i>threshold</i>	Maximum magnitude of the torque which may act on the reference system (unit: Nm). <ul style="list-style-type: none">• ≥ 0.0 The condition is met if the magnitude of the torque exerted on the reference system from any direction exceeds the value specified here.

Element	Description
<i>tolerance</i>	<p>Optional: Maximum permissible inaccuracy of the calculated values (unit: Nm).</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 10.0</p> <p>The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here.</p> <p>If the parameter is not specified, the default value is automatically used.</p>

15.19.5.2 Condition for torque

Description

A condition for the torque can be defined via the static method `createTurningTorqueCondition(...)`. The component of the overall torque applied about a definable axis of a frame below the flange (e.g. about an axis of the TCP) is considered here.

Syntax

```
CartesianTorqueCondition.createTurningTorqueCondition(
    AbstractFrame measureFrame<, AbstractFrame
    orientationFrame>, CoordinateAxis direction,
    double threshold<, double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measure Frame</i>	<p>Frame below the robot flange at which the exerted torque is determined.</p> <p>The position of the point of application of the torque is defined using this parameter.</p>
<i>orientation Frame</i>	<p>Optional: This parameter defines the orientation of the frame relative to which the torque is determined.</p> <p>If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.</p>
<i>direction</i>	<p>Coordinate axis of the reference system</p> <p>The component of the overall torque acting on the axis specified here of the reference system is checked using this condition.</p> <ul style="list-style-type: none"> • CoordinateAxis.X • CoordinateAxis.Y • CoordinateAxis.Z
<i>threshold</i>	<p>Maximum magnitude of the torque that may be applied to the axis of the reference system (unit: Nm).</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>The condition is met if the magnitude of the torque exceeds the value specified here.</p>

Element	Description
<i>tolerance</i>	<p>Optional: Maximum permissible inaccuracy of the calculated values (unit: Nm).</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 10.0</p> <p>The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here.</p> <p>If the parameter is not specified, the default value is automatically used.</p>

15.19.5.3 Condition for tilting torque

Description

A condition for the tilting torque can be defined via the static method `createTiltingTorqueCondition(...)`. The component of the overall torque applied to a plane of the reference system is considered here. The position of the plane is determined by specifying the axis which is vertical to the plane (surface normal).

Syntax

```
CartesianTorqueCondition.createTiltingTorqueCondition(
AbstractFrame measureFrame<, AbstractFrame
orientationFrame>, CoordinateAxis normalDirection,
double threshold<, double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measure Frame</i>	<p>Frame below the robot flange at which the exerted torque is determined.</p> <p>The position of the point of application of the torque is defined using this parameter.</p>
<i>orientation Frame</i>	<p>Optional: This parameter defines the orientation of the frame relative to which the torque is determined.</p> <p>If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.</p>
<i>normal Direction</i>	<p>Coordinate axis of the reference system</p> <p>The axis specified here defines the surface normal of a plane. The component of the overall torque applied to the plane is checked.</p> <ul style="list-style-type: none"> • CoordinateAxis.X • CoordinateAxis.Y • CoordinateAxis.Z
<i>threshold</i>	<p>Maximum magnitude of the tilting torque that may be applied to the plane of the reference system defined by its surface normal (unit: Nm).</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>The condition is met if the magnitude of the torque exceeds the value specified here.</p>

Element	Description
<i>tolerance</i>	<p>Optional: Maximum permissible inaccuracy of the calculated values (unit: Nm).</p> <ul style="list-style-type: none"> • > 0.0 <p>Default: 10.0</p> <p>The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here.</p> <p>If the parameter is not specified, the default value is automatically used.</p>

15.19.6 Torque component condition

Description

The torque component condition can be used to check whether the Cartesian torque exerted about the X, Y or Z axis of a frame below the robot flange (e.g. about an axis of the TCP) is outside a defined range. It is used for monitoring the Cartesian torque in a specific direction, e.g. for monitoring screw fastening processes.



The load data must be specified correctly during programming. Only then is the condition usefully applicable.



The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the force component condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.



This condition is only supported by sensitive robots, e.g. a robot of type LBR. If the condition is applied to a different robot that does not provide the required sensor information, this results in a runtime error (Exception).

The torque component condition is represented by the `TorqueComponentCondition` class. For the torque component condition, a frame below the flange coordinate system is defined as a reference system. The torque is determined at this frame, e.g. at the tip of a power wrench. The orientation of the reference system can be optionally defined via an orientation frame.

The direction in which the torque is checked is defined with one of the coordinate axes of the reference system. The torque component condition is met if the Cartesian torque about the defined coordinate axis of the reference system lies outside a definable range of values.

The Cartesian torque is calculated from the measured values of the axis torque sensors. The reliability of the calculated Cartesian torques varies depending on the axis configuration. If the quality of the calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the torque component condition is also met.

Constructor syntax

The `TorqueComponentCondition` class has several constructors which differ in their number of input parameters:

```
TorqueComponentCondition(AbstractFrame measureFrame
<, AbstractFrame orientationFramecomponent, double min, double max<, double tolerance>)
```

Explanation of the syntax

Element	Description
<i>measure Frame</i>	Frame below the robot flange relative to which the exerted torque is determined. The position of the point of application of the torque is defined using this parameter.
<i>orientation Frame</i>	Optional: This parameter defines the orientation of the frame relative to which the torque is determined. If the <i>orientationFrame</i> parameter is not specified, <i>measureFrame</i> defines the orientation of the reference system.
<i>coordinate Axis</i>	Coordinate axis of the frame relative to which the exerted torque is determined. Defines the direction in which the acting torque is checked. <ul style="list-style-type: none"> • CoordinateAxis.X • CoordinateAxis.Y • CoordinateAxis.Z
<i>min</i>	Lower limit of the range of values for the torque exerted about the coordinate axis of the reference system (unit: Nm). The torque component condition is met if the torque falls below the value specified here.
<i>max</i>	Upper limit of the range of values for the torque exerted about the coordinate axis of the reference system (unit: Nm). The torque component condition is met if the torque exceeds the value specified here. Note: The upper limit value must be greater than the lower limit value: <i>max > min</i> .
<i>tolerance</i>	Optional: Maximum permissible inaccuracy of the calculated values (unit: Nm). <ul style="list-style-type: none"> • > 0.0 Default: 10.0 The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

15.19.7 Path-related condition

Description

Path-related conditions are always used in conjunction with a motion command. They serve as break conditions or triggers for path-related switching actions.

The condition defines a point on the planned path (switching point) on which a motion is to be terminated or a desired action is to be triggered. If the switching point is reached, the condition is met.



The braking process or the defined action is only triggered when the switching point is reached. When using a path-related condition as a break condition, this results in the robot coming to a standstill after the switching point rather than directly at it.

The switching point can be defined by a shift in space and/or time. The shift can optionally refer to the start or end point of a motion.



If a time offset is defined, a change to the override influences the switching point. The action linked to a path-related condition is therefore only triggered with an effective program override of 100% and at the defined switching point in T2 or Automatic modes.

Path-related conditions are of data type MotionPathCondition.

Constructor syntax

The MotionPathCondition class has the following constructor:

```
MotionPathCondition(ReferenceType reference,  
double distance, long delay)
```

Static methods

A MotionPathCondition object can also be created via one of the following static methods:

```
MotionPathCondition.createFromDelay(  
ReferenceType reference, long delay)  
MotionPathCondition.createFromDistance(  
ReferenceType reference, double distance)
```

Explanation of the syntax

Element	Description
<i>reference</i>	Reference point of the condition (type: Enum ReferenceType) <ul style="list-style-type: none"> • ReferenceType.START: Start point • ReferenceType.DEST: End point
<i>distance</i>	Offset in space relative to the reference point of the condition. For CP motions, <i>distance</i> specifies the Cartesian distance between the switching point and the reference point, i.e. the distance along the path which connects the switching point and the reference point, and not the shortest distance between these points. (Unit: mm) For PTP motions, <i>distance</i> does not specify a Cartesian distance but rather a path parameter without a unit. <ul style="list-style-type: none"> • Negative value: Offset contrary to the direction of motion • Positive value: Offset in the direction of motion <p>(>>> "Maximum offset" Page 460)</p>

Element	Description
<i>delay</i>	<p>Offset in time relative to the path point defined by <i>distance</i>. Or if <i>distance</i> is not defined, to the reference point of the condition. (Unit: ms)</p> <ul style="list-style-type: none"> • Negative value: Offset contrary to the direction of motion • Positive value: Offset in the direction of motion <p>Phases in which the application is paused are not included in the time measurement.</p> <p>(>>> "Maximum offset" Page 460)</p>

Maximum offset

The switching point can only be offset within certain limits. The limits apply to the entire offset, comprising the shift in space and time.

- Negative offset, at most to the start point of the motion
- Positive offset, at most to the end point of the motion

The following parameterizations may not be used, as they will inevitably lead to an offset beyond the permissible limits and thus to a runtime error:

Value combination	Effect
reference = ReferenceType.START distance < 0	The switching point is before the start of the motion.
reference = ReferenceType.START distance = 0 delay < 0	
reference = ReferenceType.DEST distance > 0	The switching point is after the end of the motion.
reference = ReferenceType.DEST distance = 0 delay > 0	

Even if a valid value combination has been used, the switching point can nevertheless be offset beyond the permissible limits. In these cases, the response is as follows:

- A condition which is met before the start of the motion triggers the motion at the start point.
- A condition which is met after the end of the motion is never a trigger.

Example

A path-related condition is to be formulated for an adhesive bonding application. The adhesive bead is to end 5 cm before the end point of the motion. In order for the flow of adhesive to end in time, the condition must be met 700 ms before this distance to the end is reached.

```
MotionPathCondition glueStop = new
MotionPathCondition(ReferenceType.DEST, -50.0, -700);
```

15.19.8 Distance condition

Description

The distance condition can be used to check whether the Cartesian distance between 2 frames is less than a defined distance.

- One of the frames must be a movable frame that is located beneath the robot flange, e.g. a TCP on the tool or the frame of a gripped workpiece.
- The other frame must be a static frame.
- The movable frame must be linked to the robot flange if the condition is to be used in a motion command or monitored with a listener.

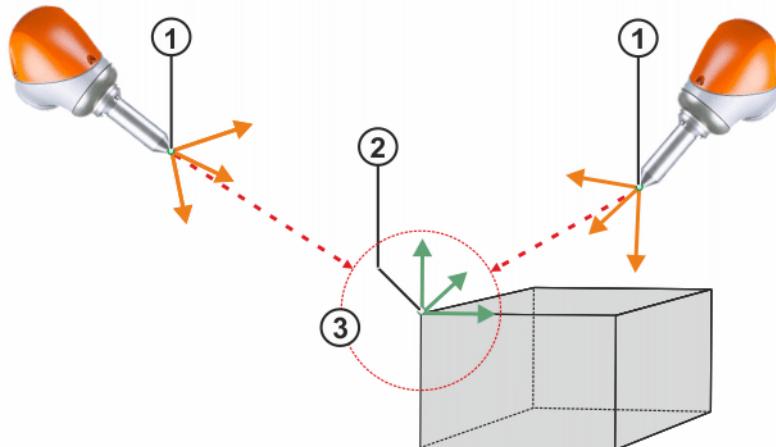


Fig. 15-20: Distance condition

- 1 Movable frame
- 2 Static frame
- 3 Minimum distance between the 2 frames

Constructor syntax

```
FrameDistanceCondition(AbstractFrame frameA,  
AbstractFrame frameB, double distanceThreshold)
```

Explanation of the syntax

Element	Description
<i>frameA</i>	One of the frames whose distance from another frame is being checked
<i>frameB</i>	One of the frames whose distance from another frame is being checked
<i>distance Threshold</i>	<p>Minimum distance between the 2 frames (unit: mm)</p> <ul style="list-style-type: none"> • > 0.0 <p>The distance condition is met if the distance between the 2 frames is less than the specified minimum distance.</p>

15.19.8.1 Distance component condition

Description

The distance component condition, like the distance condition, can be used to check the Cartesian distance between 2 frames. Additionally, different projections of the distance vector in the coordinate system of an orientation frame can be considered:

- If only one coordinate axis of the orientation frame is specified, the distance vector is projected onto this coordinate axis and the distance in the direction of this axis is measured.
- If 2 coordinate axes of the orientation frame are specified, the distance vector is projected onto the surface formed by these 2 coordinate axes and the distance is measured on this surface. This condition can be pictured as a virtual cylinder about the static frame with the movable frame being checked for entry into the cylinder radius (= minimum distance).

Constructor syntax

```
FrameDistanceComponentCondition(AbstractFrame frameA,  
AbstractFrame frameB, double distanceThreshold,  
AbstractFrame orientationFrame,  
EnumSet<CoordinateAxis> coordinateAxes)
```

Explanation of the syntax

Element	Description
<i>frameA</i>	One of the frames whose distance from another frame is being checked
<i>frameB</i>	One of the frames whose distance from another frame is being checked
<i>distance</i>	Minimum distance between the 2 frames (unit: mm) <ul style="list-style-type: none">• > 0.0 The distance component condition is met if the distance between the 2 frames is less than the specified minimum distance.
<i>orientation-Frame</i>	Frame which specifies the orientation of the distance vector Note: The frame must be a static frame and must not be connected to the robot flange.
<i>coordinate Axes</i>	Coordinate axes of the orientation frame to be considered <ul style="list-style-type: none">• CoordinateAxis.X• CoordinateAxis.Y• CoordinateAxis.Z At least one coordinate axis must be specified.

15.19.9 Condition for Boolean signals

Description

The Boolean signal condition can be used to check Boolean digital inputs or outputs. The condition is met if a Boolean input or output has a specific state.

Boolean signal conditions are of data type BooleanIOCondition.

Constructor syntax

```
BooleanIOCondition(AbstractIO booleanSignal,  
boolean booleanIOValue)
```

Explanation of the syntax

Element	Description
<i>boolean Signal</i>	Boolean input/output signal that is checked
<i>boolean IOValue</i>	State of the input/output signal with which the condition is met <ul style="list-style-type: none"> • true, false

Example

A boolean digital input signal is returned via a switch. In order to react to the signal in an application, a boolean signal condition is to be formulated. The condition must be fulfilled as soon as a high level (state TRUE) is present when the switch is activated.

```
public class ExampleApplication extends  
RoboticsAPIApplication {  
    // ...  
    @Inject  
    private SwitchesIOGroup switches;  
    // ...  
  
    @Override  
    public void run() {  
        // ...  
        AbstractIO switch1 = switches.getInput("Switch1");  
        BooleanIOCondition switch1_active =  
            new BooleanIOCondition(switch1, true);  
    }  
}
```

15.19.10 Condition for the range of values of a signal

Description

The value of a digital or analog input or output can be checked with the condition for the range of values of a signal. The condition is met if the value of the signal lies within a defined range.

Conditions for ranges of values are of data type ForceComponentCondition.

Constructor syntax

```
IORangeCondition(AbstractIO signal, Number minValue, Number maxValue)
```

Explanation of the syntax

Element	Description
<i>signal</i>	Analog or digital input/output signal that is checked
<i>minValue</i>	Lower limit of the range of values in which the condition is met The value returned by the signal must be greater than or equal to <i>minValue</i> .
<i>maxValue</i>	Upper limit of the range of values in which the condition is met The value returned by the signal must be less than or equal to <i>maxValue</i> .

Example

A temperature sensor returns an analog input signal whose value can lie in the range between 0 °C and 2000 °C. As soon as a threshold of 35 °C is exceeded, a condition for monitoring the sensor signal should be met.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    // ...
    @Inject
    private SensorIOGroup sensors;
    // ...

    @Override
    public void run() {
        // ...
        AbstractIO temperatureSensor =
            sensors.getInput("TemperatureSensor2");
        IORangeCondition tempHigher35 =
            new IORangeCondition(temperatureSensor, 35.0, 2000.0);
    }
}
```

15.20 Break conditions for motion commands

For certain processes a planned motion must not be fully executed but rather terminated when definable events occur. For example, in joining processes, the robot must stop if a force threshold is reached.

15.20.1 Defining break conditions

Description

Break conditions are conditions which cause a motion to be terminated. A break condition is met if it already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

Conditions are defined as objects of type `ICondition`. The available condition types belong to the package `com.kuka.roboticsAPI.conditionModel`.

An overview of the available condition types can be found here:

(>>> [15.19 "Overview of conditions" Page 440](#))

To define a break condition for a motion, an object of the desired condition type is transferred to the motion command via the method `breakWhen()`.

`breakWhen(...)` can be called several times when programming a motion command to define different break conditions for a motion. The individual break conditions are then linked by a logic OR operation.

The following points must be taken into consideration when programming break conditions:

- For a spline block, break conditions can only be programmed for the entire spline block. Break conditions for individual splines segments are not permissible.
- If a break condition defined for a motion within a MotionBatch is triggered, this is terminated, and then the next motion command in the batch is executed. If a break condition defined for the entire MotionBatch occurs, the entire MotionBatch is terminated.
- A break condition causes the motion currently being executed to be terminated. If no appropriate reaction strategy is programmed in the application, subsequent motions are carried out immediately after the terminated motion.
- In the case of approximated motions, the approximate positioning arc is part of the path of the subsequent motion. For this reason, only the break conditions for the subsequent motion affect the approximate positioning arc.
- If the break condition in an approximated motion occurs just before the approximate positioning point is reached, and if this does not cause the robot to come to a standstill until it is on the approximate positioning arc, the robot is accelerated again when the approximate positioning arc is reached in order to execute the subsequent motion.

Syntax

```
motion.breakWhen(condition_1, condition_2, ...);
```

Explanation of the syntax

Element	Description
<i>motion</i>	Type: Motion Motion for which a break condition is to be defined Example: <ul style="list-style-type: none"> • <code>ptp(getApplicationData().getFrame("/P1"))</code>
<i>condition</i>	Type: ICondition Parameterized ICondition object which describes a break condition

Example

A LIN motion is terminated if the torque in axis A3 is less than or equal to -12 Nm or greater than or equal to 0 Nm.

```
JointTorqueCondition cond_1 =
  new JointTorqueCondition(JointEnum.J3, -12.0, 0.0);
robot.move(lingetApplicationData().getFrame("/P10"))
  .breakWhen(cond_1);
```

15.20.2 Evaluating the break conditions

Description

If break conditions have been defined for a motion command, it is possible to view different information on the termination of a motion; for this purpose, the motion command is temporarily stored in an IMotionContainer variable. Via the method `getFiredBreakConditionInfo()`, this variable can be requested for an object of type IFiredConditionInfo, which contains the information about termination of the motion. If no break condition occurs during the motion, `getFiredBreakConditionInfo()` returns zero.

Syntax

```
IMotionContainer motionCmd = motion.breakWhen(...);
IFiredConditionInfo firedCondInfo =
    motionCmd.getFiredBreakConditionInfo();
```

Explanation of the syntax

Element	Description
<i>motion</i>	Motion instruction Example: <ul style="list-style-type: none"> • <code>lbr.move(ptpgetApplicationData().getFrame("/P1"))</code>
<i>motionCmd</i>	Type: IMotionContainer Temporary memory for the motion command
<i>firedCondInfo</i>	Type: IFiredConditionInfo Information about termination of the motion

Overview

The following methods are available in the IFiredConditionInfo interface:

Method	Description
<code>getFiredCondition()</code>	Return value type: ICondition Requests the condition which caused a motion to be terminated
<code>getPositionInfo()</code>	Return value type: PositionInformation Requests for robot position valid at the time when the break condition was triggered.
<code>getStoppedMotion()</code>	Return value type: IMotion Requests the segment of a spline block or the motion of a MotionBatch which was terminated

15.20.2.1 Requesting a break condition

Description

The condition which caused the termination of a motion can be requested via the method `getFiredCondition()`. The return value is of type ICondition and can be compared to the transferred break conditions via the `equals(...)` method.

The request is particularly useful if several break conditions for a motion have been defined by repeatedly calling the `breakWhen(...)` method.

Syntax

```
ICondition firedCondition = firedCondInfo.getFiredCondition();
```

Explanation of the syntax

Element	Description
<i>firedCondition</i>	Type: ICondition Variable for the return value. The variable contains the condition which caused the motion to be terminated.
<i>firedCondInfo</i>	Type: IFiredConditionInfo Information about termination of the motion

Example

The break conditions "cond1" and "cond2" are generated.

```
ICondition cond1;
ICondition cond2;
cond1 = new ...;
cond2 = new ...;
```

The break conditions "cond1" and "cond2" are transferred to a LIN motion with `breakWhen(...)`. The "motionCmd" variable of type IMotionContainer can be used to evaluate the motion command.

```
IMotionContainer motionCmd =
    robot.move(lingetApplicationData().getFrame("P10"))
        .breakWhen(cond1)
        .breakWhen(cond2);
```

The information about the termination of the motion is requested by "motionCmd". If the requested information is not equal to `null`, the motion has been terminated. The system only requests the triggered break condition in this case.

```
IFiredConditionInfo firedInfo =
    motionCmd.getFiredConditionInfo();

if (firedInfo != null) {
    ICondition firedCond = firedInfo.getFiredCondition();
    if (firedCond.equals(cond1)) {
        // ...
    }
    // ...
}
```

15.20.2.2 Requesting the robot position at the time of termination

Description

The robot position at the time when the break condition was triggered can be requested via the method `getPositionInfo()`.

The following position information can be accessed via the return value of type PositionInformation.

- Axis-specific actual position
- Cartesian actual position

- Axis-specific setpoint position
- Cartesian setpoint position
- Setpoint/actual value difference (translational)
- Setpoint/actual value difference (rotational)

Syntax

```
PositionInformation firedPosInfo =
firedCondInfo.getPositionInfo();
```

Explanation of the syntax

Element	Description
<i>firedPosInfo</i>	Type: PositionInformation Variable for the return value. The return value contains the position information at the time when the break condition was triggered.
<i>firedCondInfo</i>	Type: IFiredConditionInfo Information about termination of the motion

Example

The Cartesian actual position of the robot at the time when the break condition was triggered is requested via the method `getCurrentCartesianPosition()`.

```
PositionInformation firedPosInfo =
firedInfo.getPositionInfo();
Frame firedCurrPos =
firedPosInfo.getCurrentCartesianPosition();
```

15.20.2.3 Requesting a terminated motion (spline block, MotionBatch)

Description

Break conditions can be defined for an entire spline block or MotionBatch. If a break condition occurs, the entire spline block or MotionBatch is terminated.

The method `getStoppedMotion()` can be used to check which spline segment or which motion of a MotionBatch has been terminated. The return value is of type `IMotion`.

Syntax

```
IMotion stoppedMotion = firedCondInfo.getStoppedMotion();
```

Explanation of the syntax

Element	Description
<i>stoppedMotion</i>	Type: IMotion Variable for the return value. The variable contains the terminated motion.
<i>firedCondInfo</i>	Type: IFiredConditionInfo Information about termination of the motion

Example

Request using the example of a spline block:

```
ICondition stopCondition = new ...;
// ...
Spline splineMotion = new Spline(
    splgetApplicationData().getFrame("/P1"),
    circ(getApplicationData().getFrame("/P2"),
        getApplicationData().getFrame("/P3")),
    spl(getApplicationData().getFrame("/P4"))
    .setCartVelocity(150),
    lin(getApplicationData().getFrame("/P5"))
    .setCartVelocity(250).breakWhen(stopCondition);

IMotionContainer splineCont = robot.move(splineMotion);

IFiredConditionInfo firedInfoSpline =
    splineCont.getFiredConditionInfo();

if (firedInfoSpline != null) {
    IMotion stoppedMotion = firedInfoSpline.getStoppedMotion();
    // ...
}
```

15.21 Path-related switching actions (trigger)

A trigger is an event which is used to activate user-defined, path-related actions. If a specific event occurs while a motion is being executed, the action is triggered. The action is performed in parallel with the robot motion. For example, during a positioning motion, the gripper must be opened at the right time in order to be open when the pickup position for the workpiece to be transported is reached.

15.21.1 Programming triggers

Description

Events which activate path-related switching actions are called triggers. Events are defined using conditions. An event occurs if the defined condition already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

Conditions are defined as objects of type ICondition. The available condition types belong to the package com.kuka.roboticsAPI.conditionModel.

An overview of the available condition types can be found here:

(>>> [15.19 "Overview of conditions" Page 440](#))

To program a trigger, an object of the desired condition type and an ITriggerAction object which describes the action to be executed are transferred to the motion command via the method triggerWhen(...).

triggerWhen(...) can be called several times when programming a motion command to define different triggers for a motion. The execution of the corresponding switching actions is only dependent on whether the triggering event occurs, and is not influenced by the order of calling via triggerWhen(...).



The triggering event cannot re-trigger an action while it is being executed. The trigger is not effective again until the method triggerWhen(...) has ended. It is possible to request the number of events missed while the method was being executed.
(>>> [15.21.3 "Evaluating trigger information" Page 471](#))

Syntax

```
motion.triggerWhen(condition, action) ;
```

Explanation of the syntax

Element	Description
<i>motion</i>	Type: Motion Motion for which a trigger must be defined Example: <ul style="list-style-type: none"> • ptp(getApplicationContext().getFrame("/P1"))
<i>condition</i>	Type: ICondition Parameterized ICondition object which describes the condition for the trigger
<i>action</i>	Type: ITriggerAction ITriggerAction object which describes the action to be executed (>>> 15.21.2 "Programming a path-related switching action" Page 470)

15.21.2 Programming a path-related switching action

Description

The path-related action to be executed when an event occurs is defined via an ITriggerAction object. ITriggerAction is an interface from the com.kuka.roboticsAPI.conditionModel package. This interface currently does not provide any methods.

The ICallbackAction interface, which is derived from ITriggerAction, can be used for programming actions. The interface has the method onTriggerFired(...). The action to be carried out when the trigger is activated can be programmed in the body of the method onTriggerFired(...).

An ICallbackAction object can be used in any number of triggers.



The onTriggerFired(...) method is not called in real time. It is therefore not possible to guarantee specific time behavior. This can lead to delayed execution of the action.

Syntax

```
ICallbackAction action = new ICallbackAction() {
    @Override
    public void onTriggerFired(IFiredTriggerInfo triggerInformation) {
        //Action to be executed
    }
};
```

Explanation of the syntax

Element	Description
<i>action</i>	Type: ICallbackAction ICallbackAction object which describes the action transferred with triggerWhen(...)
<i>onTrigger Fired(...)</i>	Method whose execution is fired by the trigger
<i>trigger Information</i>	Type: IFiredTriggerInfo Contains information about the firing trigger (>>> 15.21.3 "Evaluating trigger information" Page 471)

Example

During motion to point “P1”, output “DO1” is always switched at the moment when input “DI1” is TRUE.

```
//set trigger action
ICallbackAction toggleOut_1 = new ICallbackAction() {
    @Override
    public void onTriggerFired(IFiredTriggerInfo
        triggerInformation) {
        //toggle output state when trigger fired
        if (IOs.getDO1())
        {
            IOs.setDO1(false);
        }
        else
        {
            IOs.setDO1(true);
        }
    };
};

//set trigger condition
BooleanIOCondition buttonPressed = new BooleanIOCondition(
    IOs.getInput("DI1"), true);

//motion with trigger
robot.move(ptp(P1)).triggerWhen(buttonPressed, toggleOut_1));
robot.move(ptp(P2));
```

15.21.3 Evaluating trigger information

The onTriggerFired(...) method is called when a trigger is activated. The object triggerInformation of type IFiredTriggerInfo, which contains various information about the activating trigger, is transferred to the onTriggerFired(...) method. This trigger information can be requested.

Overview

The following methods of the IFiredTriggerInfo class are available:

Method	Description
getFiredCondition()	Return value type: ICondition Requests the condition which fired the trigger
getMissedEvents()	Return value type: int Checks how many times the event which fired the trigger still occurred while the triggered action was being executed Note: The triggering event cannot re-trigger an action while it is being executed.
getMotionContainer()	Return value type: IMotionContainer Requests the motion command, during the execution of which the trigger was fired
getPositionInformation()	Return value type: PositionInformation Requests the position information valid at the time when the trigger was fired. The return value contains the following position information: <ul style="list-style-type: none"> • Axis-specific actual position • Cartesian actual position • Axis-specific setpoint position • Cartesian setpoint position • Setpoint/actual value difference (translational) • Setpoint/actual value difference (rotational)
getTriggerTime()	Return value type: java.util.Date Requests the time at which the trigger was fired

To request the position information obtained with getPositionInformation(), the following methods of the PositionInformation class are available:

Method	Description
getCommandedCartesianPosition()	Return value type: Frame Requests the Cartesian setpoint position at triggering time
getCommandedJointPosition()	Return value type: JointPosition Requests the axis-specific setpoint position at triggering time
getCurrentCartesianPosition()	Return value type: Frame Requests the Cartesian actual position at triggering time
getCurrentJointPosition()	Return value type: JointPosition Requests the axis-specific actual position at triggering time

Example 1

When the trigger is fired, the triggering time and condition are displayed on the smartHMI. For output purposes, a logger object has been integrated with dependency injection.

```

BooleanIOCondition in1 = new BooleanIOCondition(_input_1,
true);

ICallbackAction ica = new ICallbackAction() {
  @Override
  public void onTriggerFired(IFiredTriggerInfo
    triggerInformation) {

```

```

        logger.info("TriggerTime: " + triggerInformation
            .getTriggerTime().toString());
        logger.info("TriggerCondition: " + triggerInformation
            .getFiredCondition().toString());
    }
};

robot.move(ptpgetApplicationData().getFrame("/P1"))
.triggerWhen(in1, ica));

```

Example 2

The axis-specific and Cartesian robot position at triggering time are requested.

```

BooleanIOCondition in1 = new BooleanIOCondition(_input_1,
true);

ICallbackAction ica = new ICallbackAction() {
@Override
public void onTriggerFired(IFiredTriggerInfo
triggerInformation) {
    PositionInformation posInfo = triggerInformation
        .getPositionInformation();
    posInfo.getCommandedCartesianPosition();
    posInfo.getCommandedJointPosition();
    posInfo.getCurrentCartesianPosition();
    posInfo.getCommandedJointPosition();
}
};

robot.move(ptpgetApplicationData().getFrame("/P1"))
.triggerWhen(in1, ica));

```

15.22 Monitoring of processes

Processes can be monitored using a listener so that it is possible to react to certain events while an application is running.

These events are changes in state of defined conditions. The listener monitors the state of the condition. If the state of the condition changes, the listener is notified and the predetermined handling routine is triggered as a reaction.

During execution of a handling routine, the listener is not informed if further events occur. Once the handling routine has been completed, these events are only transferred to the listener and handled if the appropriate notification type has been defined.

15.22.1 Listener for monitoring conditions

Various listener interfaces are available from the package com.kuka.robotsAPI.conditionModel for monitoring a condition. The listeners differ in type in that they are each notified of a certain change in state of the monitored condition.

Each listener type declares a method which is executed when the listener is notified. The desired handling routine is programmed in the body of this method.

Data type	Description
IRisingEdgeListener	<p>Notification when the monitored condition is met (rising edge):</p> <ul style="list-style-type: none"> Rising edge: change in state from FALSE to TRUE <p>Method for the handling routine:</p> <ul style="list-style-type: none"> onRisingEdge(...)
IFallingEdgeListener	<p>Notification when the monitored condition is no longer met (falling edge):</p> <ul style="list-style-type: none"> Falling edge: change in state from TRUE to FALSE <p>Method for the handling routine:</p> <ul style="list-style-type: none"> onFallingEdge(...)
IAnyEdgeListener	<p>Notification for every change in state of the condition (rising or falling edge):</p> <ul style="list-style-type: none"> Rising edge: change in state from FALSE to TRUE Falling edge: change in state from TRUE to FALSE <p>Method for the handling routine:</p> <ul style="list-style-type: none"> onAnyEdge(...)

Overview

The following programming steps are required in order to be able to react to the change in state of a condition:

Step	Description
1	<p>Create a listener object to monitor the condition.</p> <p>(>>> 15.22.2 "Creating a listener object to monitor the condition" Page 474)</p>
2	Program the desired handling routine in the listener method.
3	<p>Register the listener for notification in case of a change in state of the condition.</p> <p>(>>> 15.22.3 "Registering a listener for notification of change in state" Page 475)</p>
4	<p>Activate the notification service for the listener.</p> <p>(>>> 15.22.4 "Activating or deactivating the notification service for listeners" Page 477)</p>

15.22.2 Creating a listener object to monitor the condition

Description

The syntax of a listener object is described here using the listener IAnyEdgeListener as an example. The listener method onAnyEdge(...) is automatically declared when the object is created. The method has input parameters containing information about the event that triggered execution of the method. This information can be requested and evaluated.

The listener objects of the other listener types are created in the same way and are structured analogously.

Syntax

```
IAnyEdgeListener condListener = new IAnyEdgeListener() {
    @Override
    public void onAnyEdge(ConditionObserver conditionObserver, Date time, int missedEvents, boolean conditionValue) {
        // Reaction to change in state
    }
};
```

Explanation of the syntax

Element	Description
condListener	Type: IAnyEdgeListener Name of the listener object
condition Observer	Type: ConditionObserver Object notified by the listener
time	Type: Date Date and time the listener was notified
missed Events	Type: int Number of changes in state which have occurred but not been handled. Possible causes of non-handled events: <ul style="list-style-type: none">• The notification service was deactivated when the triggering event occurred.• The handling routine was being executed when the triggering event occurred again. These events can be handled using the notification type NotificationType.MissedEvents . (>>> "NotificationType" Page 476)
condition Value	Type: Boolean Only present with the listener method onAnyEdge(...). Specifies the edge via which the method was called. <ul style="list-style-type: none">• true = rising edge: change in state from FALSE to TRUE• false = falling edge: change in state from TRUE to FALSE

15.22.3 Registering a listener for notification of change in state

Description

An object of type ConditionObserver is required to register a listener for notification in case of a change in state.

To create an object of type ConditionObserver, the ObserverManager of the application must first be requested via the method `getObserverManager()`. The `ObserverManager` class provides various methods for creating the required object.

- `createAndEnableConditionObserver(...)`
The notification service for the listener is active immediately.
- `createConditionObserver(...)`
The notification service for the listener is not active immediately, but rather must be explicitly activated.
(>>> 15.22.4 "Activating or deactivating the notification service for listeners" Page 477)

The transferred parameters in each case are identical for both methods.

Syntax

```
ConditionObserver myObserver =
getObserverManager().createAndEnableConditionObserver
(condition, notificationType, listener)
```

Explanation of the syntax

Element	Description
<i>myObserver</i>	Type: ConditionObserver Object which monitors the defined condition
<i>condition</i>	Type: ICondition Condition which is monitored
<i>notification Type</i>	Type: Enum of type NotificationType Notification type Defines the events at which the listener is to be notified in order to execute the desired handling routine. <i>(>>> "NotificationType" Page 476)</i>
<i>listener</i>	Type: IRisingEdgeListener, IFallingEdgeListener or IAnyEdgeListener Listener object which is registered

NotificationType

The Enum of type NotificationType has the following values:

Value	Description
EdgesOnly	The listener is only notified in the event of an edge change (according to the listener type used).
OnEnable	The listener is notified in the event of an edge change (according to the listener type used). In addition, the state of the monitored condition is checked upon activation of the listener. Depending on the listener type, the listener is notified when the following events occur: <ul style="list-style-type: none"> • IRisingEdgeListener: only if the condition is met upon activation • IFallingEdgeListener: only if the condition is not met upon activation • IAnyEdgeListener: if the condition is met or not met upon activation

Value	Description
MissedEvents	The listener is notified in the event of an edge change (according to the listener type used). In addition, following the execution of the handling routine, the listener is notified if triggering events were missed. This means that if the triggering edge change again occurs during execution of the handling routine, the listener is also notified again, and the handling routine is executed a second time.
All	Combination of OnEnable and MissedEvents The listener is notified in the case of all events described under OnEnable and MissedEvents.

15.22.4 Activating or deactivating the notification service for listeners

Description

The methods for activating or deactivating the notification service belong to the ConditionObserver class.

The notification service must only be activated if the method createConditionObserver(...) was used to register the listener.

Syntax

To activate the notification service:

myObserver.enable()

To deactivate the notification service:

myObserver.disable()

Explanation of the syntax

Element	Description
<i>myObserver</i>	Type: ConditionObserver Object which monitors the defined condition

15.22.5 Programming example for monitoring

A listener of type IRisingEdgeListener is defined for monitoring a force condition. As soon as a force of 35 N on the TCP is exceeded, this is considered a collision. The listener is notified and a warning lamp lights up.

NotificationType.MissedEvents is defined as the notification type. If the permissible force at the TCP is repeatedly exceeded while the warning lamp is switched on, the listener will be informed promptly.

```
ForceCondition collision = ForceCondition
    .createSpatialForceCondition(tool.getDefaultMotionFrame(), 35);

IRisingEdgeListener collisionListener = new IRisingEdgeListener() {
    @Override
    public void onRisingEdge(ConditionObserver conditionObserver,
        Date time, int missedEvents) {
        signals.setWarningLED(true);
    }
};
```

```
ConditionObserver collisionObserver = getObserverManager()  
.createConditionObserver(collision, NotificationType.MissedEvents,  
collisionListener);  
  
collisionObserver.enable();
```

15.23 Blocking wait for condition

Description

With `waitFor(...)`, an application is stopped until a certain condition is met or a certain wait time has expired. The application is then resumed.



Latency times may occur while the wait command is being processed. It is not possible to guarantee that the programmed wait time will be maintained exactly.

`waitFor(...)` must access the `ObserverManager` of the application. This is called with `getObserverManager()`.

All condition types are supported with the exception of `MotionPathCondition`.

An overview of the available condition types can be found here:
(>>> [15.19 "Overview of conditions" Page 440](#))

Syntax

Blocking wait with no time limit:

```
getObserverManager().waitFor(condition)
```

Blocking wait with a time limit:

```
boolean result = getObserverManager().waitFor(condition,  
timeout, timeUnit)
```

Explanation of the syntax

Element	Description
<i>condition</i>	Type: <code>ICondition</code> Condition which is waited for If the condition is already met when <code>waitFor(...)</code> is called, the application is immediately resumed.
<i>timeout</i>	Type: <code>long</code> Maximum wait time If the condition of the defined wait time does not occur, the application is also resumed without the occurrence of the condition.
<i>timeUnit</i>	Type: <code>Enum</code> of type <code>TimeUnit</code> Unit of the specified wait time The <code>Enum TimeUnit</code> is an integral part of the standard Java library.

Element	Description
<i>result</i>	Type: boolean Variable for the return value of <code>waitFor(...)</code> . The return value is true if the condition occurs within the specified wait time. Note: If no wait time is defined, <code>waitFor(...)</code> does not supply a return value.

Example

A wait for a boolean input signal is required in the application. The application is to be blocked for a maximum of 30 seconds. If the input signal is not supplied within this time, a defined handling routine is then to be executed.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    // ...
    @Inject
    private SwitchIOGroup inputs;
    // ...

    @Override
    public void run() {
        // ...
        Input input = inputs.getInput ("Input");

        BooleanIOCondition inputCondition =
            new BooleanIOCondition(input, true);

        boolean result = getObserverManager()
            .waitFor(inputCondition, 30, TimeUnit.SECONDS);

        if (!result) {
            //do something
        }
        else {
            //continue program
        }
    }
}
```

15.24 Recording and evaluating data

While an application is being executed, specific data, for example external forces and torques, can be recorded for later evaluation. The `DataRecorder` class (package: `com.kuka.roboticsAPI.sensorModel`) is available for programming the data recording.

The recorded data are saved in a file and stored on the robot controller in the directory `C:\KRC\Roboter\Log\DataRecorder`.

The file name is defined with the `DataRecorder` object to be created. If an error has occurred during recording, the file name begins with “FaultyDataRecorder...”.

The file can be opened with a text editor or read into an Excel table.

15.24.1 Creating an object for data recording

Description

For data recording, an object of type DataRecorder must first be created and parameterized. The following default parameters are set if the standard constructor is used for this purpose:

- The file name under which the recorded data are saved is created automatically. The name also contains an ID which is internally assigned by the system: `DataRecorderID.log`
- No recording duration is defined. Data are recorded until the buffer (currently 16 MB) is full or the maximum number of data sets (currently 30,000) is reached.
- The recording rate, i.e. the minimum time between 2 recordings, is 1 ms.

Constructor syntax

The DataRecorder class has the following constructors:

`DataRecorder()` (standard constructor)

`DataRecorder(String fileName, long timeout, TimeUnit timeUnit, int sampleRate)`

Explanation of the syntax

Element	Description
<code>fileName</code>	File name (with extension) under which the recorded data are saved Example: "Recording_1.log"
<code>timeout</code>	Recording duration <ul style="list-style-type: none"> • -1: No recording duration is defined. • ≥ 1 Default: -1 The time unit is defined with <code>timeUnit</code> .
<code>timeUnit</code>	Time unit for the recording duration Example: <code>TimeUnit.SECONDS</code> The Enum <code>TimeUnit</code> is an integral part of the standard Java library.
<code>sampleRate</code>	Recording rate (unit: ms) <ul style="list-style-type: none"> • ≥ 1 Default: 1



The DataRecorder class offers "set" methods which can be used to adapt the parameter values, in particular when using the standard constructor.

- `setFileName(...)`, `setSampleRate(...)`, `setTimeout(..., ...)`

In `setTimeout(..., ...)`, the first parameter defines the recording duration and the second parameter defines the corresponding time unit.

Example 1

Data are to be recorded every 100 ms for a duration of 5 s and written to the file Recording_1.log.

```
DataRecorder rec_1 = new DataRecorder("Recording_1.log", 5,
TimeUnit.SECONDS, 100);
```

Example 2

The DataRecorder object is generated using the standard constructor. This only specifies that data are recorded every 1 ms for an indefinite duration. The recorded data are to be written to the file Recording_2.log. The file name is defined with the corresponding "set" method.

```
DataRecorder rec_2 = new DataRecorder();
rec_2.setFileName("Recording_2.log");
```

15.24.2 Specifying data to be recorded

Using dot operators and the corresponding "add" method, the data to be recorded are added to the DataRecorder object created for this purpose. The simultaneous recording of various data is possible.

Overview

The following "add" methods of the DataRecorder class are available:

Method	Description
addInternalJointTorque(...)	Return value type: DataRecorder Recording of the measured axis torques of the robot which is transferred as a parameter (type: Robot)
addExternalJointTorque(...)	Return value type: DataRecorder Recording of the external axis torques (adjusted to the model) of the robot which is transferred as a parameter (type: robot)
addCartesianForce(...)	Return value type: DataRecorder Recording of the Cartesian forces along the X, Y and Z axes of the frame which is transferred as a parameter (unit: N). A second frame can be transferred as a parameter in order to define the orientation for the force measurement. If no separate frame is specified for the orientation, null must be transferred.

Method	Description
addCartesianTorque(...)	<p>Return value type: DataRecorder</p> <p>Recording of the Cartesian torques along the X, Y and Z axes of the frame transferred as a parameter (unit: Nm).</p> <p>A second frame can be transferred as a parameter in order to define the orientation for the torque measurement. If no separate frame is specified for the orientation, <code>null</code> must be transferred.</p>
	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>AbstractFrame measureFrame</code> Frame attached to the robot flange, e.g. the TCP of a tool. Defines the position of the measurement point. • <code>AbstractFrame orientationFrame</code> Defines the orientation of the measurement point. <p>Note: Both parameters must always be transferred together. The orientation may be <code>null</code>.</p>
addCommandedJointPosition(...)	<p>Return value type: DataRecorder</p> <p>Recording of the axis-specific setpoint position of the robot which is transferred as a parameter (type: <code>robot</code>). As a second parameter, the unit in which the axis angles are recorded must be transferred (Enum of type: <code>AngleUnit</code>).</p>
addCurrentJointPosition(...)	<p>Return value type: DataRecorder</p> <p>Recording of the axis-specific actual position of the robot which is transferred as a parameter (type: <code>Robot</code>). As a second parameter, the unit in which the axis angles are recorded must be transferred (Enum of type: <code>AngleUnit</code>).</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>Robot robot</code> • <code>AngleUnit angleUnit</code> <ul style="list-style-type: none"> – AngleUnit.Degree: Axis angle in degrees – AngleUnit.Radian: Axis angle in rad
addCommandedCartesianPositionXYZ(...)	<p>Return value type: DataRecorder</p> <p>Recording of the Cartesian setpoint position (translational section)</p> <p>The measurement point and reference coordinate system relative to which the position is recorded are transferred as parameters.</p>

Method	Description
addCurrentCartesianPosition-XYZ(...)	<p>Return value type: DataRecorder</p> <p>Recording of the Cartesian actual position (translational section)</p> <p>The measurement point and reference coordinate system relative to which the position is recorded are transferred as parameters.</p> <p>Parameters:</p> <ul style="list-style-type: none"> AbstractFrame <i>measureFrame</i> Frame attached to the robot flange, e.g. the TCP of a tool. Defines the position of the measurement point. AbstractFrame <i>referenceFrame</i> Defines the reference coordinate system. <p>Note: Both parameters must always be transferred together. None of the parameters may be null.</p>

Example

For an LBR, the following data are to be recorded using a DataRecorder object:

- Axis torques which are measured on the robot
- Force on the TCP of a gripper mounted on the robot with the orientation of a base frame

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private Tool gripper;
    // ...

    @Override
    public void run() {
        // ...
        gripper.attachTo(robot.getFlange());
        // ...
        DataRecorder rec = new DataRecorder();
        rec.addInternalJointTorque(robot);
        rec.addCartesianForce(gripper.getFrame("TCP"),
           getApplicationData().getFrame("/Base"));
        // ...
    }
}
```

15.24.3 Starting data recording

Data recording can be started independently of robot motion (possible at any point in the application) or synchronously with robot motion by means of a trigger.

Independent of robot motion

Before motion-independent recording is started, the DataRecorder object must be activated via the enable() method. Recording is started via the startRecording() method.

When recording has ended, the DataRecorder object is automatically deactivated. If data are to be recorded again with the same DataRecorder object, the DataRecorder must be re-activated.



It is not possible for more than one DataRecorder object to be activated at any one time.

Synchronous via a trigger

A condition of type ICondition and an action must be formulated for a trigger. When this condition is met, the trigger is fired, causing the action to be carried out.

(>>> *15.21.1 "Programming triggers" Page 469*)

This action starts the data recording. An object of type StartRecordingAction must be transferred for this purpose. When the object is created, the DataRecorder object to be used for data recording must be specified.

Constructor syntax:

StartRecordingAction(DataRecorder *recorder*)

The ICondition object and the StartRecordingAction object are subsequently linked to a motion command with triggerWhen(...).

Example 1

Data recording is to start when the robot has carried out the approach motion to a pre-position. The DataRecorder object is activated before the pre-position is addressed so as to reduce the delay when starting the recording.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    // ...

    @Override
    public void run() {
        // ...
        DataRecorder rec = new DataRecorder();
        // ...
        rec.enable();
        // ...
        robot.move(lingetApplicationData()
            .getFrame("/PrePosition")));
        rec.startRecording();
        // ...
    }
}
```

Example 2

Data recording is to begin 2 s after the start of a motion. A MotionPath-Condition object is parameterized for this.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
```

```

private LBR robot;
// ...

@Override
public void run() {
// ...
DataRecorder rec = new DataRecorder();
// ...
StartRecordingAction startAct =
new StartRecordingAction(rec);
MotionPathCondition startCond = new MotionPathCondition(
ReferenceType.START, 0.0, 2000);
robot.move(lin(getApplicationContext()
.getFrame("/Destination"))
.triggerWhen(startCond, startAct));
// ...
}
}

```

15.24.4 Ending data recording

Data recording can be ended independent of robot motion (possible at any point in the application), or synchronous with robot motion by means of a trigger.

In addition, recording is automatically ended when the application ends or when the recording duration specified in the DataRecorder object used has been reached.

Independent of robot motion

Recording can be stopped at any time via the stopRecording() method.

Synchronous via a trigger

A condition of type ICondition and an action must be formulated for a trigger. When this condition is met, the trigger is fired, causing the action to be carried out.

(>>> *15.21.1 "Programming triggers" Page 469*)

This action ends the data recording. An object of type StopRecordingAction must be transferred for this purpose. When the object is created, the DataRecorder object to be used for data recording must be specified.

Constructor syntax:

StopRecordingAction(DataRecorder recorder)

The ICondition object and the StopRecordingAction object are linked to a motion command with triggerWhen(...).

15.24.5 Requesting states from the DataRecorder object

Overview

The following methods of the DataRecorder class are available:

Method	Description
isEnabled()	<p>Return value type: Boolean</p> <p>The system checks whether the DataRecorder object is activated (= true).</p>
isRecording()	<p>Return value type: Boolean</p> <p>The system checks whether data recording is running (= true).</p>
isFileAvailable()	<p>Return value type: Boolean</p> <p>The system checks whether the file with the recorded data is already saved on the robot controller and whether it is available for evaluation (= true).</p>
awaitFileAvailable(...)	<p>Return value type: Boolean</p> <p>Blocks the calling application until the defined blocking duration has expired or until the file with the recorded data is saved on the robot controller and is available for evaluation (= true).</p> <p>The blocking statement returns the value “false” if the file is not available within the maximum blocking duration.</p> <p>Syntax:</p> <ul style="list-style-type: none"> • <code>awaitFileAvailable(long <i>timeout</i>, java.util.concurrent.TimeUnit <i>timeUnit</i>)</code> <p>Parameters:</p> <ul style="list-style-type: none"> • <i>timeout</i>: maximum blocking duration • <i>timeUnit</i>: time unit for the maximum blocking time

15.24.6 Example program for data recording

The following are to be recorded during an assembly process: the torques acting externally on the axes of an LBR and the Cartesian forces acting on the TCP of a gripper on the robot flange. The data are to be recorded every 10 ms.

Recording is to begin synchronously with robot motion when the force acting from any direction on the TCP of the gripper exceeds 20 N. When the assembly process ends, recording is to end as well.

The file is then to be evaluated if it is available after a maximum of 5 s.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    @Inject
    private Tool gripper;
    // ...

    @Override
    public void run() {
        // ...
        gripper.attachTo(robot.getFlange());
        // ...
        DataRecorder rec = new DataRecorder();

        rec.setFileName("Recording.log");
        rec.setSampleRate(10);
```

```

rec.addExternalJointTorque(robot);
rec.addCartesianForce(gripper.getFrame("/TCP"), null);

StartRecordingAction startAction =
new StartRecordingAction(rec);
ForceCondition startCondition = ForceCondition
.createSpatialForceCondition(
gripper.getFrame("/TCP"), 20.0);

robot.move(ptpgetApplicationData()
.getFrame("/StartPosition")));
robot.move(lin(getApplicationData()
.getFrame("/MountingPosition"))
.triggerWhen(startCondition, startAction));
robot.move(lin(getApplicationData()
.getFrame("/DonePosition")));

rec.stopRecording();

if (rec.awaitFileEnable(5, TimeUnit.SECONDS)) {
    // Evaluation of the file if available
}
// ...
}
}
}

```

15.25 Defining user keys

Description

Functions can be freely assigned to the 4 user keys on the smartPAD. For this purpose, various user key bars can be defined in the source code of the robot or background applications.

The user keys are assigned functions using the user key bar. One user key on the bar must be assigned a function, but it is not necessary for all of the keys to be assigned. In addition, graphical or text elements illustrating the function of each user key are located on the side panel of the smartHMI screen next to the user keys.

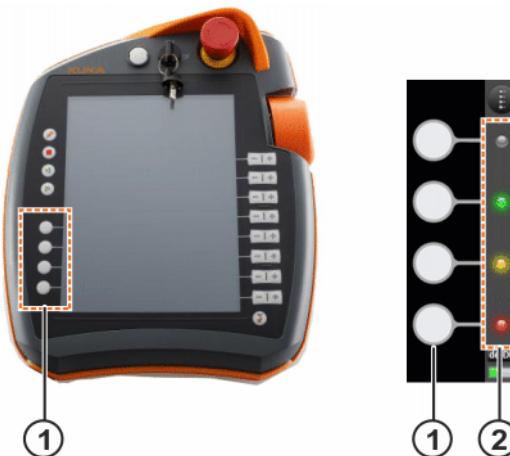


Fig. 15-21: User keys on the smartPAD (example)

1 User keys

2 Bar with LED icons

All the user key bars defined in the running robot application or a running background application are available to the operator. For example, one

user key bar can be used for controlling a gripper, and in another bar the same keys can be used to switch status signals.

User key bars are available until the task which created them has ended.

Overview

The following steps are required in order to program a user key bar:

Step	Description
1	Create a user key bar. (>>> 15.25.1 "Creating a user key bar" Page 488)
2	Add user keys to the bar (at least one). (>>> 15.25.2 "Adding user keys to the bar" Page 489)
3	Define the function which is to be executed if the user key is actuated. (>>> 15.25.3 "Defining the function of a user key" Page 491)
4	Assign at least one graphical or text element to the area along the left side panel of the smartHMI next to the user key. (>>> 15.25.4 "Labeling and graphical assignment of the user key bar" Page 493)
5	For user keys which trigger functions associated with a risk: Define the warning message to be displayed when the user key is actuated. The message appears before the function can be triggered. (>>> 15.25.5 "Identifying safety-critical user keys" Page 496)
6	Publish a user key bar. (>>> 15.25.6 "Publishing a user key bar" Page 497)

15.25.1 Creating a user key bar

Description

The following methods are required in order to create a user key bar:

- `getApplicationUI()`

This method is used to access the interface to the smartHMI graphical user interface from a robot application or a background application.
Return value type: `ITaskUI`

- `createUserKeyBar(...)`

This method is used to create the user key bar. It is part of the `ITaskUI` interface.

Syntax

```
IUserKeyBar keybar =
getApplicationUI().createUserKeyBar("name");
```

Explanation of the syntax

Element	Description
<i>keybar</i>	Type: IUserKeyBar Name of the user key bar created with <code>createUserKeyBar(...)</code>
<i>name</i>	Type: String Name under which the user key bar is displayed on the smartHMI (>>> <i>Fig. 6-14</i>) The number of characters which can be displayed is limited. <ul style="list-style-type: none">• A maximum of 12 to 15 characters is recommended.

Example

A user key bar for controlling a gripper is created.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");
```

15.25.2 Adding user keys to the bar

Description

A newly created user key bar does not have any user keys to start with. The user keys to be used must be added to the bar.

The IUserKeyBar interface provides the following methods for this purpose:

- `addUserKey(...)`
Adds a single user key to the bar.
- `addDoubleUserKey(...)`
Combines 2 neighboring user keys to a double key and adds this to the bar. The corresponding areas on the side panel of the smartHMI screen are also combined into a larger area.

When adding a user key to a bar, the user defines the function to be executed when the user key is actuated (e.g. opening a gripper, changing a parameter, etc.). Depending on the programming, both pressing and releasing the user key can be interpreted as actuation and linked to a function.

A user key bar must have at least one user key. Each user key is assigned a unique number. This number is transferred when a user key is added.

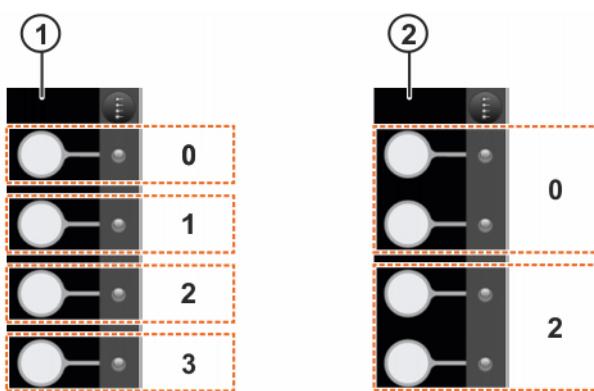


Fig. 15-22: Numbering of the user keys

1 Single keys

2 Double keys

Syntax

Adding a single key:

```
IUserKey key = keybar.addUserKey(slot, listener, ignoreEvents);
```

Adding a double key:

```
IUserKey doubleKey = keybar.addDoubleUserKey(slot, listener, ignoreEvents);
```

Explanation of the syntax

Element	Description
<i>keybar</i>	Type: IUserKeyBar Name of the user key bar to which a user key is added
<i>key</i>	Type: IUserKey Name of the single key added to the bar
<i>doubleKey</i>	Type: IUserKey Name of the double key added to the bar
<i>slot</i>	Type: int Number of the user key which is added. Single keys: • 0 ... 3 Double keys: • 0, 2
<i>listener</i>	Type: IUserKeyListener Name of the listener used to define the function to be executed when the user key is actuated (>>> <i>15.25.3 "Defining the function of a user key"</i> <i>Page 491</i>)

Element	Description
<i>ignore Events</i>	<p>Type: Boolean</p> <p>Defines whether there is a reaction if the user key is re-actuated while the key function is being executed</p> <ul style="list-style-type: none"> • true: If the key is actuated while the function is being executed, it has no effect. • false: It is counted how many times the key is actuated while the function is being executed. The function is repeated this many times.

Example

The user keys are assigned the following functions for controlling a gripper:

- The top user key is to be used to open the gripper, and the key below it is to close the gripper.
- The two lower user keys are combined in a double key. This is to be used to increase and decrease the velocity of the gripper.
- The functions for opening and closing the gripper are not to be called again until the respective function has ended.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");

IUserKeyListener openGripperListener = ...;
IUserKeyListener closeGripperListener = ...;
IUserKeyListener gripperVelocityListener = ...;

IUserKey openKey = gripperBar.addUserKey(0,
    openGripperListener, true);
IUserKey closeKey = gripperBar.addUserKey(1,
    closeGripperListener, true);
IUserKey velocityKey = gripperBar.addDoubleUserKey(2,
    gripperVelocityListener, false);
```

15.25.3 Defining the function of a user key

Description

In order to define which function is to be executed when a user key is actuated, a listener object of type `IUserKeyListener` must be created. The `onKeyEvent(...)` method is automatically declared when the object is created.

The listener method `onKeyEvent(...)` is called when the following events occur:

- The user key is pressed.
- The user key is released.



Only one `OnKeyEvent(...)` can be carried out even if different listeners are used. For example, if the user triggers the `OnKeyEvent(...)` of user key 2 while the `OnKeyEvent(...)` of user key 1 is being executed, the second `OnKeyEvent(...)` will not start until the first has been completed.

Syntax

```
IUserKeyListener listener = new IUserKeyListener() {
```

```

@Override
public void onKeyEvent(IUserKey key, UserKeyEvent
event) {
// Reaction to event
}
;

```

Explanation of the syntax

Element	Description
listener	Type: IUserKeyListener Name of the listener object
Input parameters of the listener method onKeyEvent(...):	
key	Type: IUserKey User key which has been actuated The parameter can be used to directly access the user key, for example to change the corresponding labelling or graphical assignment. In addition, it is possible to determine which user key has been actuated, especially when the same reaction is used for different user keys.
event	Type: Enum of type UserKeyEvent Event called by the listener method onKeyEvent(...) Enum values for single keys: <ul style="list-style-type: none"> • UserKeyEvent.KeyDown: Key has been pressed. • UserKeyEvent.KeyUp: Key has been released. Enum values for double keys: <ul style="list-style-type: none"> • UserKeyEvent.FirstKeyDown: Of the two keys, the upper one has been pressed. • UserKeyEvent.SecondKeyDown: Of the two keys, the lower one has been pressed. • UserKeyEvent.FirstKeyUp: Of the two keys, the upper one has been released. • UserKeyEvent.SecondKeyUp: Of the two keys, the lower one has been released.

Example

The user key bar for controlling a gripper is expanded by a method which can be used to adapt the velocity of the gripper. The two lower user keys combined in a double key are used for this purpose.

The attribute `velocity` is declared for setting the velocity. The attribute specifies the current velocity as a proportion of the maximum velocity (range of values: 0.1 ... 1.0). Pressing the upper user key increases the value by 0.1 and pressing the lower user key decreases it by 0.1.

```

double velocity = 0.1;
// ...
IUserKeyBar gripperBar = ...;
// ...
IUserKeyListener gripperVelocityListener = new IUserKeyListener() {
@Override
public void onKeyEvent(IUserKey key, IUserKeyEvent event){
if (event == UserKeyEvent.FirstKeyDown && velocity <= 0.9) {

```

```

        velocity = velocity + 0.1;
    }
    else if (event == UserKeyEvent.SecondKeyDown && velocity >= 0.2) {
        velocity = velocity - 0.1;
    }
}
};

// ...
IUserKey velocityKey = gripperBar.addDoubleUserKey(2, gripperVelocityListener,
    false);

```

15.25.4 Labeling and graphical assignment of the user key bar

Description

At least one graphical or text element must be assigned to the area along the left side panel of the smartHMI next to the user key. LED icons of various colors and sizes are available as graphical elements. These elements can be adapted during the runtime of the robot application or the background task.

In order to clearly position the individual elements, the area next to the user key is divided into a grid with 3x3 spaces. This also applies for user keys that have been grouped together as a double key. In the case of double keys, the grid stretches over both fields.

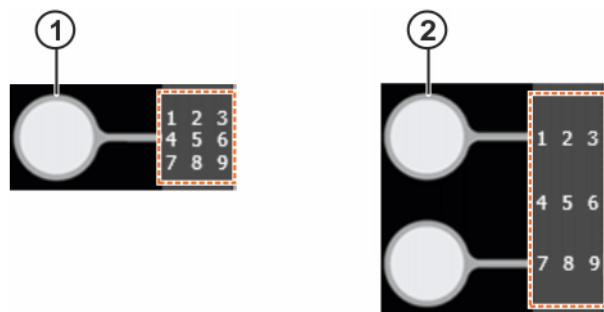


Fig. 15-23: Division of the grid

1 Single keys

2 Double keys

One element can be set in each grid space. This grid space is defined by the value of the enum UserKeyAlignment. If a new element is allocated to a grid space which has already been assigned, the existing element is deleted.

UserKey alignment

Grid space no.	Value
1	UserKeyAlignment.TopLeft
2	UserKeyAlignment.TopMiddle
3	UserKeyAlignment.TopRight
4	UserKeyAlignment.MiddleLeft
5	UserKeyAlignment.Middle
6	UserKeyAlignment.MiddleRight
7	UserKeyAlignment.BottomLeft
8	UserKeyAlignment.BottomMiddle

Grid space no.	Value
9	UserKeyAlignment.BottomRight

15.25.4.1 Assigning a text element

Description

Each grid space can be assigned a text element. The `setText(...)` method is used for this purpose. The method belongs to the `IUserKey` interface.

Syntax

```
key.setText(position, "text");
```

Explanation of the syntax

Element	Description
<code>key</code>	Type: <code>IUserKey</code> User key to which a text element is assigned
<code>position</code>	Type: Enum of type <code>UserKeyAlignment</code> Position of the element (grid space) (>>> "UserKey alignment" Page 493)
<code>text</code>	Type: <code>String</code> Text to be displayed Often, a text length of 2 or more characters will exceed the size of the grid space. The text display area is then expanded. However, it is only practical to use a limited number of characters. The possible number of characters depends on the text elements of the neighboring grid spaces and the characters used.

Example

The user key bar for controlling a gripper is to be expanded. A suitable label should be displayed continuously next to each of the user keys.

- Label for the user keys for opening and closing the gripper: OPEN and CLOSE
 - Label for the user keys for increasing and decreasing the gripper velocity: plus sign and minus sign
- In addition, the current velocity is to be displayed and automatically updated each time a change is made.

```
double velocity = 0.1;
// ...
IUserKeyBar gripperBar = ...;
// ...
IUserKeyListener gripperVelocityListener = new IUserKeyListener() {
    @Override
    public void onKeyEvent(IUserKey key, IUserKeyEvent event) {
        if (event == UserKeyEvent.FirstKeyDown && velocity <= 0.9) {
            velocity = velocity + 0.1;
        }
        else if (event == UserKeyEvent.SecondKeyDown && velocity >= 0.2) {
            velocity = velocity - 0.1;
        }
    }
}
```

```

    }
    // The following line formats the velocity display
    // The first three characters are displayed
    String value = String.valueOf(velocity).substring(0, 3);
    key.setText(UserKeyAlignment.Middle, value);
}
};

IUserKey openKey = ...;
openKey.setText(UserKeyAlignment.TopLeft, "OPEN");

IUserKey closeKey = ...;
closeKey.setText(UserKeyAlignment.TopLeft, "CLOSE");

IUserKey velocityKey = ...;
velocityKey.setText(UserKeyAlignment.TopMiddle, "+");
velocityKey.setText(UserKeyAlignment.Middle, Double.toString(velocity));
velocityKey.setText(UserKeyAlignment.BottomMiddle, "-");

```

15.25.4.2 Assigning an LED icon

Description

Each grid space can be assigned an LED icon. The `setLED(...)` method is used for this purpose. The method belongs to the `IUserKey` interface.

Syntax

```
key.setLED(position, led, size);
```

Explanation of the syntax

Element	Description
<code>key</code>	Type: <code>IUserKey</code> User key to which a graphical element is assigned
<code>position</code>	Type: Enum of type <code>UserKeyAlignment</code> Position of the element (grid space) (>>> "UserKey alignment" Page 493)
<code>led</code>	Type: Enum of type <code>UserKeyLED</code> Color of the LED icon <ul style="list-style-type: none"> • <code>UserKeyLED.Grey</code>: gray • <code>UserKeyLED.Green</code>: green • <code>UserKeyLED.Yellow</code>: yellow • <code>UserKeyLED.Red</code>: red
<code>size</code>	Type: Enum of type <code>UserKeyLEDSIZE</code> Size of the LED icon <ul style="list-style-type: none"> • <code>UserKeyLEDSIZE.Small</code>: small • <code>UserKeyLEDSIZE.Normal</code>: large

Example

The user key bar for controlling a gripper is to be expanded. The user keys for opening and closing the gripper should each be assigned a small LED icon.

As long as the gripper is opening or closing, the LED icons should be displayed in green. If the gripper is stationary, the LED icons should be displayed in gray.

```
IUserKeyBar gripperBar = getApplicationUI().createUserKeyBar("Gripper");

IUserKeyListener openGripperListener = new IUserKeyListener() {
    @Override
    public void onKeyEvent(IUserKey key, UserKeyEvent event) {
        key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Green,
                  UserKeyLEDSIZE.Small);
        openGripper(); // Method for opening the gripper
        key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
                  UserKeyLEDSIZE.Small);
    }
};

IUserKeyListener closeGripperListener = new IUserKeyListener() {
    @Override
    public void onKeyEvent(IUserKey key, UserKeyEvent event) {
        key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Green,
                  UserKeyLEDSIZE.Small);
        closeGripper(); // Method for closing the gripper
        key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
                  UserKeyLEDSIZE.Small);
    }
};

IUserKeyListener gripperVelocityListener = ...;
// ...

IUserKey openKey = ...;
openKey.setText(...);
openKey.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
              UserKeyLEDSIZE.Small);

IUserKey closeKey = ...;
closeKey.setText(...);
closeKey.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
               UserKeyLEDSIZE.Small);

IUserKey velocityKey = ...;
```

15.25.5 Identifying safety-critical user keys

Description

User keys can trigger functions that are associated with a risk. In order to prevent damage caused by the unintentional actuation of such user keys, a warning message can be added identifying them as safety-critical. The `setCriticalText(...)` method is used for this purpose. The method belongs to the `IUserKey` interface.

If the operator actuates a user key designated as safety-critical, the message defined with `setCriticalText(...)` is displayed on the smartHMI in a

window with the name **Critical operation**. The user key is then deactivated for approx. 5 s. Once this time has elapsed, the operator can trigger the desired function by actuating the user key again within 5 s.

If the user key is not actuated within this time or if an area outside of the **Critical operation** window is touched, the window is closed and the user key is reset to its previous state.

Syntax

```
key.setCriticalText("text");
```

Explanation of the syntax

Element	Description
<code>key</code>	Type: IUserKey User key which is provided with a warning message
<code>text</code>	Type: String Message text displayed when the user key is actuated

Example

The user key bar for controlling a gripper is to be expanded. If the user key for opening the gripper is actuated, a warning message should appear. The operator is requested to ensure that no damage can result from workpieces falling out when the gripper is opened.

```
IUserKeyBar gripperBar = getApplicationUI().createUserKeyBar("Gripper");
// ...
IUserKey openKey = ...;
openKey.setText...;
openKey.setLED...;

openKey.setCriticalText("Gripper opens when key is actuated again. Ensure
that no damage can result from workpieces falling out!");
```

15.25.6 Publishing a user key bar

Description

Once a user key bar has been equipped with all the necessary user keys and functionalities, it must be published with the `publish()` method. Only then can the operator access it on the smartPAD.

Once a user key bar has been published, further user keys may not be added later in the program sequence. In other words, it is not possible to add an unassigned user key and assign a function to it at a later time. It is, however, possible to change the labeling or graphical element displayed next to the user key on the smartHMI at a later time.

Syntax

```
keybar.publish();
```

Explanation of the syntax

Element	Description
<code>keybar</code>	Type: IUserKeyBar Name of the user key bar created with <code>createUserKeyBar(...)</code> .

Example

The user key bar created for controlling a gripper is published.

```
IUserKeyBar gripperBar = getApplicationUI().createUserKeyBar("Gripper");
// ...
gripperBar.publish();
```

15.26 Message programming

15.26.1 Programming user messages

Description

It is possible to program notification, warning and error messages which are displayed on the smartHMI and written to the LOG file of the application while the application is running. In addition, it is possible to program messages which are not displayed on the smartHMI but are only written to the LOG file.

In order to program a user message, an object of the ITaskLogger class is integrated by means of dependency injection. At this object, the corresponding methods can be called in order to generate a message display with the appropriate LOG level.

Dependency injection makes it possible for messages to be displayed on the smartHMI from all classes of an application, including those which are not a task.



It is advisable to only display messages on the smartHMI which are absolutely essential. Over-intensive use of the message display (permanently > 10 messages/s) can have a negative effect on the runtime of the application and the operation of the smartHMI.



For message output, it is advisable to use only the commands described here and not other logging functionalities, e.g. the Java commands System.out.println(...) or System.err.println(...). If these commands are used, it is not possible to guarantee that the message will be displayed on the smartHMI.

Syntax

Integrating a logger object:

```
@Inject
private ITaskLogger logger;
```

Notification message:

```
logger.info("Message");
```

Warning message:

```
logger.warn("Message");
```

Error message:

```
logger.error("Message");
```

Message that is only written to the LOG file:

```
logger.fine("Message");
```

Explanation of the syntax

Element	Description
<i>logger</i>	Type: ITaskLogger Name of the logger object, as it is to be used in the application
<i>Message</i>	Type: String Message which is to be displayed on the smartHMI and/or written to the LOG file

Example

Once the robot has reached an end point, a notification message is to be displayed. If the motion ended with a collision, a warning notification is displayed instead.

```
public class ExampleApplication extends
RoboticsAPIApplication {
    @Inject
    private ITaskLogger logger;
    @Inject
    private IApplicationData data;
    @Inject
    private LBR robot;

    private ForceCondition collision

    @Override
    public void initialize() {
        // initialize your application here
        collision = ForceCondition
            .createSpatialForceCondition(robot.getFlange(), 15.0);
    }

    @Override
    public void run() {
        // ...
        IMotionContainer motion = robot.move(lin(getFrame("/P20"))
            .breakWhen(collision));

        if (motion.getFiredBreakConditionInfo() == null) {
            logger.info("End point reached.");
        }
        else {
            logger.warn("Motion canceled after collision!");
        }
        // ...
    }
}
```

15.26.2 Programming user dialogs

Description

User dialogs can be programmed in an application. These user dialogs are displayed in a dialog window on the smartHMI while the application is being run and require user action.

Various dialog types can be programmed via the method `displayModalDialog(...)`. The following icons are displayed on the smartHMI according to type:

Icon	Type
	INFORMATION Dialog with information of which the system integrator must take note
	QUESTION Dialog with a question which the system integrator must answer
	WARNING Dialog with a warning of which the system integrator must take note
	ERROR Dialog with an error message of which the system integrator must take note

The medical device manufacturer answers by selecting a button that can be labeled by the programmer. Up to 12 buttons can be defined.

The application from which the dialog was called is stopped until the system integrator reacts. How program execution continues can be made dependent on which button the system integrator selects. The method `displayModalDialog(...)` returns the index of the button which the system integrator selects on the smartHMI. The index begins at "0" (= index of the first button).

Syntax

```
getApplicationUI().displayModalDialog(Dialog type, "Dialog text", "Button_1"<, ... "Button_12">)
```

Explanation of the syntax

Element	Description
<i>Dialog type</i>	Type: Enum of type ApplicationDialogType <ul style="list-style-type: none"> INFORMATION: The dialog with the information icon is displayed. QUESTION: The dialog with the question icon is displayed. WARNING: The dialog with the warning icon is displayed. ERROR: The dialog with the error icon is displayed.
<i>Dialog text</i>	Type: String Text which is displayed in the dialog window on the smartHMI
<i>Button_1</i> ... <i>Button_12</i>	Type: String Labeling of buttons 1 ... 12 (proceeding from left to right on the smartHMI)

Example

The following user dialog of type QUESTION is to be displayed on the smartHMI:

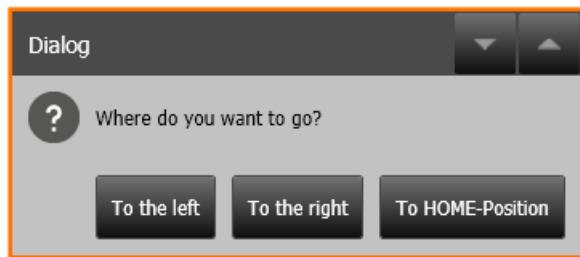


Fig. 15-24: Example of a user dialog

```

int direction = getApplicationUI().displayModalDialog(
    ApplicationDialogType.QUESTION,
    "Where do you want to go to?",
    "To the left", "To the right", "To HOME-Position");

switch (direction) {
    case 0:
        robot.move(ptpgetApplicationData().getFrame("/Left"));
        break;
    case 1:
        robot.move(ptpgetApplicationData().getFrame("/Right"));
        break;
    case 2:
        robot.move(ptpHome());
        break;
}

```

15.27 Program execution control

15.27.1 Pausing an application

Description

An application can be paused with the `halt()` method.

The `halt()` method pauses the motion currently being executed, and the application state on the smartHMI switches to **Motion paused**.

`halt()` causes a blocking stop of the calling thread. If further threads are running at the same time, these will continue to be executed. The application execution is only stopped if `halt()` is called in the application thread. It is therefore advisable not to call `halt()` in handling routines for path-related switching actions or in handling routines for monitoring processes. Instead, it is advisable to use the `pause()` method in these handling routines.

(>>> *15.27.2 "Pausing motion execution" Page 502*)

The motion and paused thread may only be resumed via the Start key on the smartPAD. Pressing the Start key causes the paused motion to resume. The paused thread is resumed with the instruction following `halt()` in the source code.

Syntax

```
getApplicationControl().halt();
```

15.27.2 Pausing motion execution

Description

Motion execution can be paused with the pause() method.

The behavior corresponds to pausing the application via the smartPAD. The pause() method pauses the motion currently being executed, and the application state on the smartHMI switches to **Motion paused**.

pause() does not cause a blocking wait. The application continues to be executed until a synchronous motion command is reached.

Motion execution may only be resumed via the Start key on the smart-PAD.

Syntax

```
getApplicationControl().pause();
```

15.27.3 Canceling a motion command

Description

Asynchronously executed motion commands can be canceled using the cancel() method while the motion is in progress.



Cancelling an active motion modifies the path of the subsequent motions compared with the path without cancellation.

Example

A motion is executed from P1 to P4. P2 and P3 are approximated with an absolute approximation distance of 20 mm.

```
IMotionContainer m1 =
    robot.moveAsync(lin(getApplicationData().getFrame("/P1")));
IMotionContainer m2 =
    robot.moveAsync(lin(getApplicationData().getFrame("/P2"))
        .setBlendingCart(20));
IMotionContainer m3 =
    robot.moveAsync(lin(getApplicationData().getFrame("/P3"))
        .setBlendingCart(20));
IMotionContainer m4 =
    robot.moveAsync(lin(getApplicationData().getFrame("/P4")));
```

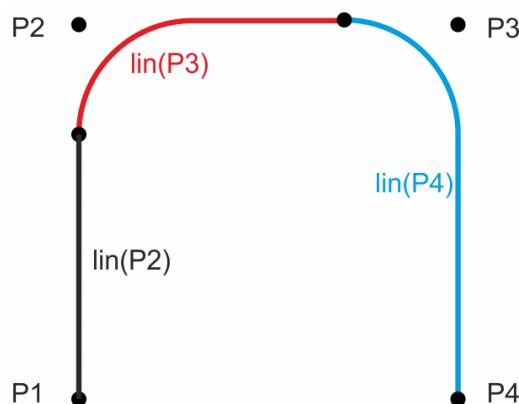


Fig. 15-25: Original path

During the motion, the linear motion command to point P3 is canceled with cancel().

```
IMotionContainer m1 =
    robot.moveAsync(lingetApplicationData().getFrame("/P1")));
IMotionContainer m2 =
    robot.moveAsync(lingetApplicationData().getFrame("/P2"))
    .setBlendingCart(20));
IMotionContainer m3 =
    robot.moveAsync(lingetApplicationData().getFrame("/P3"))
    .setBlendingCart(20));
IMotionContainer m4 =
    robot.moveAsync(lingetApplicationData().getFrame("/P4")));
m2.await();
ThreadUtil.milliSleep(500);
m3.cancel();
m4.await();
```

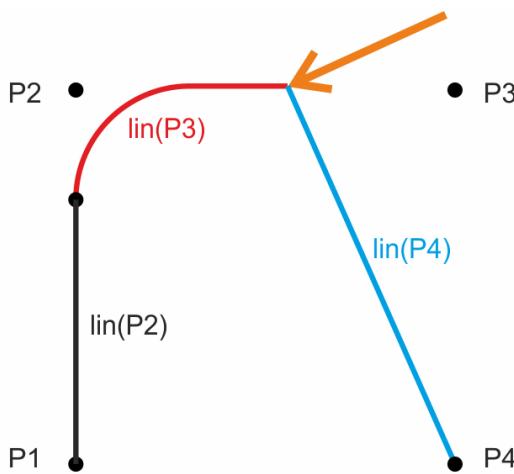


Fig. 15-26: cancel() during an active motion

15.27.4 FOR loop

Description

The FOR loop, also called counting loop, repeats a statement block as long as a defined condition is met.

A counter is defined, which is increased or decreased by a constant value with each execution of the loop. At the beginning of a loop execution, the system checks if a defined condition is met. This condition is generally formulated by comparing the counter with a limit value. If the condition is no longer met, the loop is no longer executed and the program is continued after the loop.

The FOR loop is generally used if it is known how often a loop must be executed.

FOR loops can be nested.

(>>> [15.27.9 "Examples of nested loops" Page 511](#))

Syntax

```
for (int Counter = Start value; Condition; Counting statement) {
    Statement_1;
    <...
    Statement_n;
```

}

Explanation of the syntax

Element	Description
<i>Counter</i>	Counter for the number of loops executed The counter is assigned a start value. With each execution of the loop, the counter is increased or decreased by a constant value.
<i>Start value</i>	Start value of the counter
<i>Condition</i>	Condition for the loop execution The counter is generally compared with a limit value. The result of the comparison is always of type Boolean. The loop is ended as soon as the comparison returns FALSE, meaning that the condition is no longer met.
<i>Counting statement</i>	The counting statement determines the amount by which the counter is changed with each execution of the loop. The increment and counting direction can be specified in different ways. Examples: <ul style="list-style-type: none"> • <i>Start value ++ --</i>: With each execution of the loop, the start value is increased or decreased by a value of 1. • <i>Start value + - Increment</i>: With each execution of the loop, the start value is increased or decreased by the specified increment.

Example

```
for (int i = 0; i < 10; i++) {
    logger.info("Variable i =" + i);
}
```

The value of the variable `i` is increased by 1 with every cycle. The current value of `i` is displayed on the smartHMI with every cycle. The loop is executed a total of 10 times. The values of 0 to 9 are displayed in the process. For output purposes, a logger object has been integrated with dependency injection.

15.27.5 WHILE loop

Description

The WHILE loop repeats a statement block for as long as a certain condition is fulfilled. It is also called a rejecting loop because the condition is checked before every loop execution.

If the condition is no longer met, the statement block of the loop is no longer executed and the program is resumed after the loop. If the condition is not already fulfilled before the first execution, the statement block is not executed at all.

The WHILE loop is generally used if it is unknown how often a loop must be executed, e.g. because the repetition condition is calculated or is a specific signal.

WHILE loops can be nested.

(>>> 15.27.9 "Examples of nested loops" Page 511)

Syntax

```
while (Repetition condition) {
    Statement_1;
    <...
    Statement_n;
}
```

Explanation of the syntax

Element	Description
<i>Repetition condition</i>	Type: Boolean Possible: <ul style="list-style-type: none"> • Variable of type Boolean • Logic operation, e.g. a comparison, with a result of type Boolean

Example 1

```
while (input1 == true) {
    logger.info("Input 1 is TRUE.");
}
logger.info("Input 1 is FALSE.");
```

Before the loop is executed the system checks whether an input signal is set. As long as this is the case, the loop will be executed again and again and the smartHMI will display the input as TRUE. If the input signal has been reset, the loop will not be executed (any longer) and the input will be displayed as FALSE. For output purposes, a logger object has been integrated with dependency injection.

Example 2

```
int w = 0;
Random num = new Random();
while (w <= 21) {
    w = w + (num.nextInt(6) + 1);
}
```

With every loop execution, the value of the variable `w` is increased by a random number between 1 and 6. As long as the sum of all random numbers is less than 21, the loop will be executed. It is not possible to predict the exact number of cycles. It is possible that the loop is ended after 4 cycles (3 x 6 and 1 x 3) or only after 21 cycles (21 x 1).

15.27.6 DO WHILE loop

Description

The DO WHILE loop repeats a statement block until a certain condition is fulfilled. It is also called a post-test loop because the condition is only checked after every loop execution.

The statement block is executed at least once. When the condition is met, the loop is terminated and the program is resumed.

The DO WHILE loop is generally used if a loop must be executed at least once, but it is unknown how often in total, e.g. because the break condition is being calculated or is a specific signal.

DO WHILE loops can be nested.

(>>> 15.27.9 "Examples of nested loops" Page 511)

Syntax

```
do {
    Statement_1;
    <...
    Statement_n;
} while (Break condition);
```

Explanation of the syntax

Element	Description
<i>Break condition</i>	Type: Boolean Possible: <ul style="list-style-type: none"> • Variable of type Boolean • Logic operation, e.g. a comparison, with a result of type Boolean

Example

```
int num;
do {
    num = (int) (Math.random() * 6 + 1);
} while (num != 6);
```

Random numbers between 1 and 6 are generated until the “dice” shows a 6. The dice must be thrown at least once.

15.27.7 IF ELSE branch

Description

The IF ELSE branch is also called a conditional branch. Depending on a condition, either the first statement block (IF block) or the second statement block (ELSE block) is executed.

The ELSE block is executed if the IF condition is not met. The ELSE block may be omitted. If the IF condition is not met, then no further statements are executed.

It is possible to check further conditions and to link them to statements after the IF block using `else if`. As soon as one of these conditions is met and the corresponding statements are executed, the subsequent branches are no longer checked.

Several IF statements can be nested in each other.

Syntax

```
if (Condition_1) {
    Statement_1;
    <...
    Statement_n;
```

```

    }
<else if (Condition_2) {
Statement_1;
<...
Statement_n;
}>
<else {
Statement_1;
<...
Statement_n;
}>

```

Explanation of the syntax

Element	Description
Condition	Type: Boolean Possible: <ul style="list-style-type: none"> • Variable of type Boolean • Logic operation, e.g. a comparison, with a result of type Boolean

Example 1

IF branch without else

```

int a;
int b;

if (a == 17) {
    b = 1;
}

```

If variable `a` has the value 17, variable `b` is assigned the value 1.

Example 2

IF branch within a FOR loop without else

```

for (int a = 1; a <= 10; a++) {
    if (a == 3) {
        a = a + 5;
    }
    logger.info("Variable a =" + a);
}

```

The loop is executed 5 times. If variable `a` has the value 3, the value of `a` is increased by 5 once only.

The values 1, 2, 8, 9 and 10 are displayed on the smartHMI. For output purposes, a logger object has been integrated with dependency injection.

Example 3

IF ELSE branch with else if

```

double velAct = 0.0;
double velDesired = 130.0;

```

```
// ...

if (velAct < velDesired) {
    accelerating();
} else if (velAct > velDesired) {
    braking();
} else {
    testrun();
}
```

In a program, a test run for a vehicle is to be carried out. This test run is only meaningful at a specific command velocity.

The IF statement checks whether the actual velocity `velAct` is lower than the command velocity `velDesired`. If this is the case, the vehicle accelerates. If this is not the case, it continues with `else if`.

The IF ELSE statement checks whether the actual velocity `velAct` is higher than the command velocity `velDesired`. If this is the case, the vehicle is braked. If this is not the case, the ELSE block is executed with the test run.

15.27.8 SWITCH branch

Description

The SWITCH branch is also called a multiple branch. Generally, a SWITCH branch corresponds to a multiply nested IF branch.

In a SWITCH block, different CASE blocks can be executed which are designated by CASE labels (jump labels). Depending on the result of an expression, the corresponding CASE block is selected and executed. The program jumps to the CASE label and is resumed at this point.

The keyword `break` at the end of a CASE block means that the SWITCH block is left. If no `break` follows at the end of an instruction block, all subsequent instructions (not only instructions with CASE labels) are executed until either a BREAK label is reached or all instructions have been executed.

A DEFAULT block can optionally be programmed. If no condition is met for jumping to a CASE label, the DEFAULT block is executed.

Syntax

```
switch (expression) {
    case Constant_1:
        Statement_1;
        <...
        Statement_n; >
        < break;>
        <...
    case Constant_n:
        Statement_1; >
        <...
        Statement_n; >
        < break;>
        < default:
            Statement_1; >
```

```
<...
Statement_n;>
< break;>
}
```

Explanation of the syntax

Element	Description
<i>Expression</i>	Type: int, byte, short, char, enum
<i>Constant</i>	<p>Type: int, byte, short, char, enum</p> <p>The data type of the constant must match the data type of the expression.</p> <p>Note: Constants of type char must be specified with ' , e.g. case 'a'</p>

Example 1

SWITCH branch with BREAK and DEFAULT instruction:

```
int a, b;
// ...
switch (a) {
    case 1:
        b = 10;
        break;
    case 2:
        b = 20;
        break;
    case 3:
        b = 30;
        break;
    default:
        b = 40;
        break;
}
// next command
```

If variable *a* has the value 1, the program jumps to the label *case 1*. The variable *b* is assigned the value 10. The BREAK instruction causes the SWITCH block to be left. Program execution is resumed with the next command after the closing bracket of the SWITCH block.

If variable *a* has the value 2 at the start, variable *b* is assigned the value 20. If *a* has the value 3 at the start, *b* is assigned the value 30.

The DEFAULT statement is optional. It is nonetheless advisable for it always to be set. If variable *a* has a value at the start that is not covered by a CASE statement (e.g. 0 or 5), the instructions in the DEFAULT block are executed. In this example, this means that variable *b* is assigned the value 40.

Example 2

The keyword *break* may be omitted in a CASE statement. Cases in which this is practically applied include the following:

- The identical statement is to be executed in multiple CASE instances (e.g. *a* = 1, 2 or 3). See SWITCH statement with fall-through (variant 1).

- For a CASE instance, specific statements and additional statements applicable to another instance are to be executed. See SWITCH statement with fall-through (variant 2).

SWITCH statement with fall-through (variant 1):

```
int a, b;
// ...
switch (a) {
    case 1:
        // fall-through
    case 2:
        // fall-through
    case 3:
        b = 20;
        break;
    case 4:
        b = 30;
        break;
    default:
        b = 40;
        break;
}
// next command
```

In variant 1, the statements to be executed are only written to the last of the grouped CASE blocks. Omission of the BREAK statement in `case 1` and `case 2` makes the assignment of variable `b` in these CASE blocks obsolete too, as variable `b` will be overwritten in `case 3` anyway. To make it evident that the BREAK statement has not been forgotten but intentionally omitted, fall-through is entered as a comment.

SWITCH statement with fall-through (variant 2):

```
int a, b, c;
// ...
switch (a) {
    case 1:
        b = 10;
        // fall-through
    case 2:
        c = 20;
        break;
    case 3:
        b = 30;
        break;
    case 4:
        c = 30;
        break;
    default:
        b = 40;
        c = 40;
        break;
}
// next command
```

The behavior in variant 2 is as follows:

- If `case 1` occurs, variable `b` is set to 10 and additionally variable `c` to 20.

In `case 1` fall-through is entered as a comment to indicate that further statements are to be executed, in this case those of `case 2`.

- If case 2 occurs, variable `c` is set to 20. Variable `b` is not changed.

15.27.9 Examples of nested loops

The outer loop is first executed until the inner loop is reached. The inner loop is then executed completely. The outer loop is then executed until the end, and the system checks whether the outer loop must be executed again. If this is the case, the inner loop must also be executed again.

There is no limit on the nesting depth of loops. The inner loops are always executed as often as the outer loop.

FOR in FOR loop

```
for (int i = 1; i < 4; i++) {
    logger.info(i + ".Cycle begins");

    for (int k = 10; k > 0; k--) {
        logger.info("..." + k);
    }
}
```

The outer loop determines that the inner loop is executed 3 times. The counter of the outer loop starts with the value `i = 1`.

Once the smartHMI has displayed the start of the 1st cycle, the counter of the inner loop starts with the value `k = 10`. The value of variable `k` is decreased by 1 with every cycle. The current value of `k` is displayed on the smartHMI with every cycle. If variable `k` has the value 1, the inner loop will be executed for the last time.

Then the outer loop is ended and the value of variable `i` is increased by 1. The 2nd cycle begins. For output purposes, a logger object has been integrated with dependency injection.

FOR in WHILE loop

```
int sum = 0;
int round = 1;
int diceRoll = 0;
Random num = new Random();

while (sum < 21) {
    round++;

    for (int i = 1; i <= 3; i++) {
        diceRoll = (num.nextInt(6) + 1);
        if (diceRoll % 2 == 0) {
            sum += diceRoll;
        }
    }
}
```

The following rules apply in a dice game:

- The total sum of all rolls must be at least 21 (check with WHILE loop).
- The dice are rolled 3 times in each round (FOR loop).
- Only even numbers (2, 4 and 6) are counted (IF check with modulo).

15.28 Continuing a paused application in Automatic mode (recovery)

Description

If a paused application is to be continued in Automatic mode, the higher-level controller must be able to determine whether the robot is still situated on its programmed path. If the robot is no longer situated on the path, e.g. following a non-path-maintaining stop or because it was jogged while the program was paused, there must be a suitable strategy for automatically repositioning the robot.

This return strategy may only be applied if it can be ensured that there is no risk of a collision while the robot is returning to the path. If this is not ensured, the robot must be manually repositioned by the user.

RoboticsAPI provides the `IRecovery` interface for automatic repositioning. It is possible to access the interface from robot and background applications:

- `IRecovery getRecovery()`

Overview

The `IRecovery` interface provides methods for requesting information about whether robots must be repositioned in order to resume a paused application and which return strategy is applied.

Method	Description
<code>isRecoveryRequired()</code>	<p>Return value type: Boolean</p> <p>Checks whether one or more robots used in the application must be repositioned in a paused application.</p> <p>true: At least one robot must be repositioned for the application to be resumed.</p> <p>false: The application can be resumed immediately.</p>
<code>isRecoveryRequired(...)</code>	<p>Return value type: Boolean</p> <p>Checks whether a specific robot must be repositioned in a paused application. The robot is transferred as a parameter (type: <code>Robot</code>).</p> <p>true: The robot must be repositioned for the application to be resumed.</p> <p>false: The application can be resumed immediately.</p>
<code>getRecoveryStrategy(...)</code>	<p>Return value type: <code>RecoveryStrategy</code></p> <p>Requests the strategy being applied in order to return a specific robot to the path. The robot is transferred as a parameter (type: <code>Robot</code>).</p> <ul style="list-style-type: none"> • PTPRecoveryStrategy: The robot is repositioned with a PTP motion. The robot is moved at 20% of the maximum possible axis velocity and the effective program override. No further strategies are available at this time. <p>The method returns <code>null</code> in the following cases:</p> <ul style="list-style-type: none"> • No return strategy is required or available. • The application is not paused.

PTPRecovery Strategy

The PTPRecoveryStrategy class provides “get” methods which are used to request the characteristics of the PTP motion. With these methods, it is possible to evaluate whether the return strategy may be carried out in Automatic mode.

Method	Description
getStartPosition()	<p>Return value type: JointPosition</p> <p>Requests the start position of the PTP motion (= axis position from which the robot can be repositioned)</p> <p>The start position is the currently commanded setpoint position of the robot and not the currently measured actual position.</p>
getMotion()	<p>Return value type: PTP</p> <p>Requests the PTP motion carried out on execution of the strategy</p> <p>Further information can be requested from the returned motion object:</p> <ul style="list-style-type: none"> • getDestination(): Target position of the PTP motion (= axis position at which the robot left the path) • getMode(): Controller mode of the motion which was interrupted

External controller

The robot controller must inform the higher-level controller whether the robot must be repositioned. The higher-level controller may only allow the return strategy to be carried out if this can be done without risk. Otherwise, the robot may only be manually repositioned.

The following system signals are available:

- Output AutExt_AppReadyToStart

With this output, the robot controller communicates to the higher-level controller whether or not the application may be resumed.

- If isRecoveryRequired(...) supplies the value **false** (= no repositioning required), the output can be set to TRUE.
 - If getRecoveryStrategy(...) supplies **null** (= no return strategy available), the output must be set to FALSE.
 - If the evaluation of the return strategy shows that it can be executed in Automatic mode, the output can be set to TRUE.
- If this is not the case, the output must be set to FALSE.

- Input App_Start

The higher-level controller informs the robot controller via a rising edge that the application should resume. (Precondition: AutExt_AppReadyToStart is TRUE)

The higher-level controller must send the start signal App_Start twice:

1. Start signal for repositioning
2. Start signal for resuming the application

15.29 Error treatment

15.29.1 Handling of failed motion commands

Motion commands that are communicated to the robot controller can fail for various reasons, e.g.:

- End point lies outside of a workspace
- End point cannot be reached with the given axis configuration
- The frame used is not present in the application data

As standard, a failed motion command results in termination of the application. Handling routines can be defined in order to prevent the application from terminating in case of error.

The following handling options are available depending on the error:

- Failed synchronous motion commands are handled using a try-catch block
- Failed asynchronous motion commands are handled using an event handler

15.29.2 Handling of failed synchronous motion commands

Description

Synchronously executed motion commands (`.move(...);`) are sent in steps to the real-time controller and executed. The further execution of the program is interrupted until the motion has been executed. Only then is the next command sent.

Using a try-catch block, predictable runtime errors or exceptions can be executed in the program sequence without the application being aborted.

A defined method for error treatment is triggered within a try-catch block. When the keyword `try` is called, an attempt is made to execute the listed command. If an error occurs during execution, the corresponding handling routine is started in the catch block.

Syntax

```
try{
    // Code in which a runtime error could occur when executed
}
catch(Exception e){
    // Code for treating the runtime error
}
< finally{
    // Final treatment (optional)
}>
```

Explanation of the syntax

Element	Description
try{...}	The try block contains a code which could result in a run-time error. If an error occurs, the execution of the try block is terminated and the catch block is executed.
catch (...) { ... }	The catch block contains the code for treating the run-time error. The catch block will only be executed if an error occurs in the try block.
Exception e	The error data type (here: Exception) can be used to define the error type to be handled in the catch block. The error type Exception is the superclass of most error data types. However, it is also possible to focus on more specific errors. Information about errors which have occurred can be polled using the parameter e. In particular, the error data type CommandInvalidException (package: com.kuka.roboticsAPI.executionModel) is important. It occurs, for example, when the end point of the motion cannot be reached.
finally { ... }	The finally block is optional. Here it is possible to specify a final treatment to be executed in all cases, whether or not an error occurs in the try block.

Example

A robot executes a motion under impedance control with very low stiffness. For this reason, it is not guaranteed to reach the end position. It is then to move relatively by 50 cm in the positive Z direction of the flange coordinate system. If the robot is in an unfavorable position following the motion under impedance control, the linear motion cannot be executed and a runtime error will occur. In order to prevent the application from aborting in this case, the critical linear motion is programmed in a try-catch block. If the motion planning fails, the robot should be moved to an auxiliary point before the application is resumed.

```
CartesianImpedanceControlMode softMode = new
CartesianImpedanceControlMode();

softMode.parametrize(CartDOF.ALL).setStiffness(10.0);
exampleRobot.move(ptp(getFrame("/Start"))
    .setMode(softMode).setJointVelocityRel(0.3));

try{
    getLogger().info("1: Try to execute linear motion");
    exampleRobot.move(linRel(0.0, 0.0, 500.0)
        .setJointVelocityRel(0.5));
}

catch(CommandInvalidException e){
    getLogger().info("2: Bewegung nicht möglich");
    exampleRobot.move(ptp(getFrame("/Auxiliary point"))
        .setJointVelocityRel(0.5));
}
```

```

finally{
    getLogger().info(
        "3: Final treatment in finally block is executed");
}

getLogger().info("4: Further in the program");

```

15.29.3 Handling of failed asynchronous motion commands

Description

In the case of asynchronously executed motion commands (`.moveA-sync(...);`), the next program line is executed directly after the motion command is sent.

An event handler is used in order to react to a failed asynchronous motion command.

This event handler is an object of type `IErrorHandler` and defines the method `handleError(...)`. The transfer of further motion commands to the real-time controller is blocked during execution of the method `handleError(...)`. The application remains at a standstill.

The handling routine is defined with `handleError(...)`. Information on the failed motion command can be accessed via the input parameters of the method. The method returns a parameter of type `ErrorHandlingAction`. The final reaction to the error is selected via this parameter.

The following reactions are available:

- The application is terminated with an error.
- The motion execution is paused and can only be resumed by the user pressing the Start key on the smartPAD.
- The error is ignored and the application is resumed.

The defined event handler must be registered before it can be used in the application. The method `getApplicationControl().registerMoveAsyncErrorHandler(...)` is used for this purpose. The method belongs to the `IApplicationControl` interface.

Syntax

Defining the event handler:

```

IErrorHandler errorHandler = new IErrorHandler() {
    @Override
    public ErrorHandlingAction handleError
        (Device device, IMotionContainer failedContainer,
         List<IMotionContainer> canceledContainers) {
        // Code which is executed in case of error
        return ErrorHandlingAction.reaction;
    }
};

```

Registering the event handler:

```

getApplicationControl().registerMoveAsyncErrorHandler(errorHandler);

```

Explanation of the syntax

Element	Description
<i>errorHandler</i>	Type: IErrorHandler Name of the event handler responsible for handling failed asynchronous motion commands
Input parameters of the handleError(...) method:	
<i>device</i>	Type: device The parameter can be used to access the robot for which the failed motion command is commanded.
<i>failed Container</i>	Type: IMotionContainer The parameter can be used to access the failed motion command.
<i>canceled Contain-ers</i>	Type: List<IMotionContainer> The parameter can be used to access a list of all deleted motion commands. It contains all motion commands which have already been sent to the real-time controller when the method handleError(...) is called.
<i>reaction</i>	Type: Enum of type ErrorHandlingAction Return value of the handleError(...) method by means of which the final reaction to the error is defined: <ul style="list-style-type: none"> • ErrorHandlingAction.EndApplication: The application is terminated with an error. • ErrorHandlingAction.PauseMotion: The motion execution is paused until the user resumes the application via the smartPAD. • ErrorHandlingAction.Ignore: The error is ignored and the application is resumed.

Example

Several asynchronous motion commands are to be executed in an application. By registering an event handler of type IErrorHandler, a handling routine is defined using the method handleError(...) for the event that one of the asynchronous motion commands fails:

- The smartHMI displays which motion command has failed.
- The smartHMI displays which motion commands are no longer executed.

The handleError(...) method is ended with the return of the value ErrorHandlingAction.Ignore.

```
public class ErrorHandler extends RoboticsAPIApplication {
    // fields which need to be injected
    @Inject
    private ITaskLogger logger;
    @Inject
    private LBR robot;

    // not injected fields
    private IErrorHandler errorHandler;

    @Override
    public void initialize() {
```

```

    errorHandler = new IErrorHandler() {
        @Override
        public ErrorHandlingAction handleError(Device device,
        IMotionContainer failedContainer,
        List<IMotionContainer> canceledContainers) {
            logger.warn("Execution of the following motion
failed: " + failedContainer.getCommand().toString());
            logger.info("The following motions will not be
executed:");
            for (int i = 0; i < canceledContainers.size(); i++) {
                logger.info(canceledContainers.get(i)
                .getCommand().toString());
            }
            return ErrorHandlingAction.Ignore;
        };
    };

    getApplicationControl()
        .registerMoveAsyncErrorHandler(errorHandler);
}

@Override
public void run(){
    robot.move(ptpHome());
    robot.move(ptp(getFrame("/PrePos")));

    // ...

    robot.moveAsync(ptp(getFrame("/P1")));
    robot.moveAsync(ptp(getFrame("/P2")));

    robot.moveAsync(lin(getFrame("/P3")));

    robot.moveAsync(ptp(getFrame("/P4")));
    robot.moveAsync(ptp(getFrame("/P5")));
    robot.moveAsync(ptp(getFrame("/P6")));

    robot.moveAsync(ptp(getFrame("/P7")));
    robot.moveAsync(ptp(getFrame("/P8")));
    robot.moveAsync(ptp(getFrame("/P9")));

    // ...

    robot.move(ptpHome());
}
}

```

To explain the system behavior, it is assumed that the linear motion to P3 cannot be planned. This means that the method `handleError(...)` is called. In our example, the robot is situated at end point P2 at this time.

If, for example, the motion commands to P4, P5, P6 are already in the real-time controller at the same time, these motion commands will be deleted and no longer executed.

Calling the method `handleError(...)` will block further motion commands from being sent to the real-time controller. In this case, the application will be stopped before the motion command to P7. If the `handleError(...)` method is ended with the return of the value `ErrorHandlingAction.Ignore`, the application is resumed. The robot then moves directly from its current position P2 to P7.

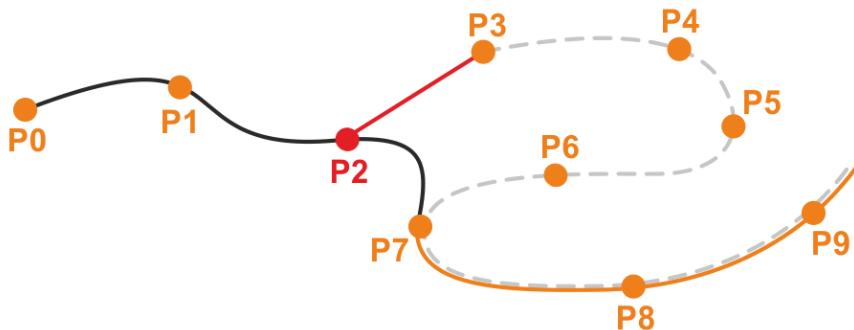


Fig. 15-27: Failed motion to P3 (example of path)

15.30 Overview of additional Med functions

When the LBR Med software package is installed, the following additional functions are installed for Sunrise.OS Med:

- ([>>> 15.30.1 "Creating a Sunrise Med project using a template" Page 519](#))
- ([>>> 15.30.2 "Robot status overview" Page 520](#))
- ([>>> 15.30.3 "Mastering axes" Page 524](#))
- ([>>> 15.30.4 "Jog mode" Page 526](#))

15.30.1 Creating a Sunrise Med project using a template

The procedure for creating a Sunrise Med project differs only in the selection of the **Topology template** list.

([>>> 5.3 "Creating a Sunrise project with a template" Page 61](#))

Step	Description
1	If required, install the SunriseFramework Med option package in Sunrise.Workbench. (>>> 10.7.1 "Installing the option package" Page 204)
2	In order to select the catalog elements on the Software tab, the robot controller and then the template of the robot must be selected in the list. <ul style="list-style-type: none"> • Depending on the robot variant, a corresponding template must be selected: <ul style="list-style-type: none"> – LBR Med 7 R800 – LBR Med 14 R820
3	After selecting the robot variant, the following catalog elements are available for selection on the Software tab: <ul style="list-style-type: none"> • KUKA Sunrise.LBRMedExtension Med (automatically selected) • KUKA Sunrise.LBRMed Extension Med - Example Applications When manually selecting KUKA Sunrise.LBRMed Extension Med - Example Applications , the sample applications are copied into the selected project upon saving and can then be synchronized with the controller. (>>> 10.3 ""Software" tab" Page 194)

The following sample applications are available following installation of the catalog element:

- BrakeTestMonitorSampleApp:
Sample application that performs the cyclical brake test
- UnmasteredSampleApp:
Sample application for mastering axis 2 of the robot and for starting an application, even if the robot has not been mastered.
- RobotStateEventSampleApp:
Sample application for operation of the event mechanism for RobotStateEvents that represent the status of the LBR Med, in the same way as the display of the smartPAD.
- JoggingSampleApp:
Sample application for jogging

15.30.2 Robot status overview

15.30.2.1 Polling the robot state

Description

The RobotStateAggregator class can be used in the application to poll the robot state at any time. Additionally, it is possible to register in the application for RobotStateEvents that informs all listeners of any change of state. The medical device manufacturer can use this information to determine the readiness of the LBR Med for its intended use.

Overview

The RobotStateAggregator class contains the following methods:

Method	Description
RobotStateAggregator(<i>robot</i>)	Transfer parameter <i>robot</i> (type: Robot) Constructor for creating an instance of RobotStateAggregator for the robot <i>robot</i> .
isReadyForNormalUse()	Return value type: boolean Message indicating whether the robot is ready for normal operation <ul style="list-style-type: none"> • true: robot is ready for normal operation • false: robot is not ready for normal operation
getFailedConditionsForNormalUse()	Return value type: list of type FailedConditionForNormalUse Returns a list of all errors that prevent normal operation of the robot: (>>> "Changes in state" Page 520)

Changes in state

The list of type getFailedConditionsForNormalUse() has the following values:

Value	Description
NOT_READY_TO_MOVE	<p>The robot is not operational. Possible causes:</p> <ul style="list-style-type: none"> • The robot is stopped for safety reasons. • T1 or T2 mode is selected and the enabling switch is not pressed. <p>In this case, the message WRONG_OPERATION_MODE is returned.</p>
WRONG_OPERATION_MODE	The active operating mode is not AUT.
NOT_MASTERED	At least one axis of the robot is not mastered.
NOT_GMS_REFERENCED	At least one joint torque sensor of the robot is not referenced. Note: Only applies to torque-sensitive robots (e.g. LBR Med).
NOT_POSITION_REFERENCED	At least one position sensor is not mastered or referenced.
WRONG_BOOT_STATE	The boot state is incorrect.
WRONG_DEVICE_STATE	The device state is incorrect.
CONNECTION_LOST	The connection to the robot controller has been lost.



If the methods isReadyForNormalUse() and getFailedConditionsForNormalUse() are used, polling of the robot data may be delayed. The data are polled in sequence and not simultaneously. For this reason, the data are several milliseconds apart. This can result in inconsistency between the return values of these two methods if the data have changed before the result of the method has been calculated and returned.

Example

```
// Initialise a RobotStateAggregator
// and check if the robot is ready for normal use,
// if not get the failed conditions as a list.

RobotStateAggregator _robotStateAggregator = new
RobotStateAggregator (lbr);
if(!_robotStateAggregator.isReadyForNormalUse ()) {
    List<FailedConditionForNormalUse> failedConditions
    =
    robotStateAggregator.getFailedConditionsForNormalUse () ;
}
```

15.30.2.2 Robot status log

Description

The Java package com.kuka.med.medController provides the interface IR-robotStateListener. The interface can be used to register a listener that returns the changes in status of the robot as logged events.

The information provided by these events is comparable to the information provided by the status indicators of the smartHMI user interface on the smartPAD.

The following events are returned:

- A list of the unfulfilled conditions that must be met for operation of the robot.
- The trigger source of the event.
- List of the system components for which an error or warning is active together with the corresponding error or warning messages referring to the relevant tile.

The MedController class provides the following methods for registering listeners:

- `addRobotStateListener(IRobotStateListener listener, Robot robot)`:
Registers the listener for the change-of-state events of the robot.
 - If a parameter value = NULL, an `IllegalArgumentException` is triggered
- `removeRobotStateListener (IRobotStateListener listener)`:
Removes the listener.

The RobotStateEvent class provides the following methods:

Method	Description
<code>getTrigger()</code>	<p>Return value type: Enum of type <code>TriggerName</code></p> <p>Returns the trigger source of the event:</p> <ul style="list-style-type: none"> • INITIAL_STATE: Feedback of system status immediately after initialization • CONNECTION_LOST: Connection to the robot controller has been interrupted • SHUTDOWN: Shutdown of the overall system has been triggered • SAFETY_STATE_CHANGED: The safety monitoring state has changed, e.g.: <ul style="list-style-type: none"> – An external EMERGENCY STOP has been switched. – The operating mode has been changed to Manual Reduced Velocity mode (T1). • TILE_STATE_CHANGED: The status indicator (LED) on a tile of the smartPAD signals a change of state, e.g.: <ul style="list-style-type: none"> – If the robot is restarted and the current PDS firmware version does not match the PDS firmware version on the robot controller, the status indicator of the Device state tile turns red. – The status indicators of the tiles at Station level change if an EMERGENCY STOP button is pressed or released.
<code>getFailedConditions()</code>	<p>Return value type: list of type <code>FailedConditionForNormalUse</code></p> <p>Returns a list of all errors that prevent normal operation of the robot. The list is updated with every change of state:</p> <p>(>>> <i>"Changes in state" Page 520</i>)</p>

Method	Description
getInfoOfTiles InWarningErrorState()	<p>Return value type: map <string, TileInfo></p> <p>Returns a map with all tiles that are in the warning or error state. The information (type: TileInfo) is sorted by the name of the tile on the smartHMI. (>>> 15.30.2.3 "Status indicator on the smartHMI" Page 523)</p>

15.30.2.3 Status indicator on the smartHMI

The structure of the TileInfo class is comparable to the information provided by the status indicators on the user interface of the smartHMI.

(>>> [6.5 "KUKA smartHMI user interface" Page 85](#))

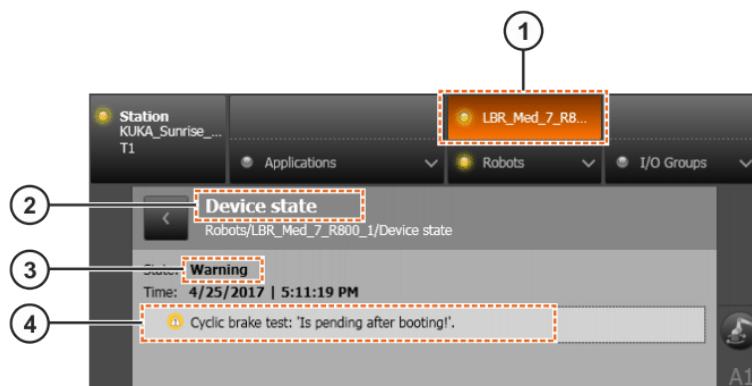


Fig. 15-28: TileInfo class compared with smartHMI

- 1 Indicator: getTileLevel()
- 2 Indicator: getTileName()
- 3 Indicator: getTileState()
- 4 Indicator: getErrorMsgTree()

The TileInfo class provides the following interfaces:

Method	Description
getTileName()	<p>Return value type: string</p> <p>Returns the name of the tile represented on the smartHMI.</p>
getTileLevel()	<p>Return value type: string</p> <p>Returns the aggregated level of the tile represented on the smartHMI.</p>
getTileState()	<p>Return value type: Enum of type TileState</p> <p>Returns the state of the tile represented on the smartHMI.</p>
getErrorMsgTree()	<p>Return value type: list with elements of the ErrorMsgTree class</p> <p>Returns the error message of the tile represented on the smartHMI. The list contains a separate element of the ErrorMsgTree class for each level of the hierarchy.</p>



It must be noted that every action of the system can trigger multiple events with different triggers. For example, pressing the EMERGENCY STOP triggers one event with the trigger SAFETY_STATE_CHANGED and one with the trigger TILE_STATE_CHANGED. Events with different triggers may nevertheless contain the same list of FailedConditionForNormalUse.

Example

```

public class RobotStateEventSampleApp extends
RoboticsAPIApplication
implements IRobotStateListener{

    private MedController _medController;
    @Inject
    private LBR _lbr;

    @Override
    public void initialize(){
        _medController = (MedController) _lbr.getController();
        // register the application as robot state listener
        _medController.addRobotStateListener(this, _lbr);
        // User specific initialization
    }

    @Override
    public void dispose(){
        // remove listener registration of this application
        _medController.removeRobotStateListener(this);
        super.dispose();
    }

    @Override
    public void run(){
        // User specific application
    }

    @Override
    public void onRobotStateChanged(RobotStateEvent event){
        // User specific behavior when robot state changes
    }
}

```

15.30.3 Mastering axes

Description

Using the additional Med functions, the robot can also be mastered from within the robot application, without the need to use the smartPAD. It is not normally possible to start an application with an unmastered robot. For this reason, the additional annotation `@MedApplicationCategory` is provided. The Mastering class provides the interface for mastering from within the application.

Overview

The following methods are available:

Method	Description
Mastering(robot)	Constructor for creating an instance for mastering the robot <i>robot</i> of type Robot in the robot application.
masterAxis(axis)	<p>Return value type: boolean</p> <p>Performs mastering of the axis <i>axis</i> (type: int). The specified axis must not be mastered.</p> <ul style="list-style-type: none"> • true: mastering of the axis was successful • false: mastering of the axis failed or the axis was already mastered
isAxisMastered(axis)	<p>Return value type: boolean</p> <p>Monitoring of the mastering status of the axis <i>axis</i> (type: int)</p> <ul style="list-style-type: none"> • true: the defined axis is mastered • false: the axis is not mastered
invalidateMastering(axis)	<p>Return value type: boolean</p> <p>Resets the mastering status of the axis <i>axis</i> (type: int) to unmastered. The specified axis must be mastered.</p> <ul style="list-style-type: none"> • true: the axis was unmastered successfully • false: the axis could not be unmastered or was already unmastered



If a robot application is paused and at least one axis has not been mastered, the motion commands of the application can only be resumed if it concerns mastering motions or axis-specific jogging.

(>>> [15.30.4 "Jog mode" Page 526](#))

Other motion commands (e.g. PTP, LIN, CIRC) lead to a CommandInvalidException since these motion commands cannot be safely executed without mastering.

If a robot application is paused because the robot has left the path, the robot must be repositioned. In such a case, the repositioning can only be carried out if all axes have been mastered.



If the robot application was provided with the annotation '@MedApplicationCategory' and one or more of the axes have been mastered, repositioning is deactivated. In such a case, the repositioning of the robot is thus not possible if the robot leaves the path and the application is paused.

Example

Using the annotation @MedApplicationCategory:

```
// when the parameter checkMastering is set to false,
// the user application is allowed
// to start even if not all axes of the robot are mastered.
@MedApplicationCategory(checkMastering = false)
public class UnmasteredAppSample extends
RoboticsAPIApplication {
...
}
```

Using the Mastering class:

```
LBR lbr = (LBR) getRobot(kuka_Sunrise_Cabinet_1, "LBR");
...
// instantiate the mastering
```

```

Mastering mastering = new Mastering(lbr);
// invalidate mastering of first axis
mastering.invalidateMastering(0);
// check whether the first axis is mastered
mastering.isAxisMastered(0);
// master the first axis
mastering.masterAxis(0);

```

15.30.4 Jog mode

Description

This functionality makes it possible to control robot jog commands from the application.

Jogging means that the robot is moved manually for a certain time without a predefined destination. Alternatively, the robot can only be moved manually using the smartPAD. (>>> [6.16 "Jogging the robot" Page 103](#))

The Med software package com.kuka.med.jogging enables the following functions:

- Configuration of the velocity before and during execution of the motion command
- Jogging of a freely selected number of robot axes
- Jogging of freely selectable Cartesian degrees of freedom, e.g.:
 - Jogging relative to the base coordinate system
 - Jogging relative to the tool coordinate system
 - Jogging relative to an external reference frame

The duration of the commands can either be limited over time, explicitly stopped or implicitly stopped on reaching axis limits or singularities.

Overview

The Jogging class provides the following methods:

Method	Description
Jogging(<i>robot</i>)	Constructor for creating an instance for jogging the robot <i>robot</i> of type Robot in the robot application.
startJointJogging(<i>axes</i> , <i>vel</i> , [<i>joggingTimeInMilliseconds</i>])	<p>Return value type: boolean</p> <p>The method is used to start axis-specific jogging with a specified velocity <i>vel</i> (type: double) for the axes <i>axes</i> (type: EnumSet of type JointEnum).</p> <p>The optional parameter <i>joggingTimeInMilliseconds</i> (type: long) defines a timeout for the motion; the default value is 5000 ms. If another motion is already active, jogging is not started and the value "false" is returned. If Cartesian jogging is already active, no axis-specific motion may be started and an IllegalStateException is generated.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Programmed velocity: <ul style="list-style-type: none"> – $-1.0 \leq \text{vel} \leq 1.0$ • Timeout for jogging (optional) (unit: ms) <ul style="list-style-type: none"> – ≤ 0: no timeout is defined Default: 5000 • true: axis-specific jogging has been started successfully • false: the robot was already executing a motion command and jogging is ignored
updateJointJogging(<i>axes</i> , <i>velocity</i>)	<p>Return value type: boolean</p> <p>The method updates the runtime parameters of an active, axis-specific jog command for the axes <i>axes</i> (type: EnumSet of type JointEnum).</p> <ul style="list-style-type: none"> • Programmed velocity: <ul style="list-style-type: none"> – $-1.0 \leq \text{velocity} \leq 1.0$ • true: the data were updated successfully • false: the data were not updated successfully, or no axis-specific jog command is active
stopJointJogging(<i>axes</i>)	<p>Return value type: boolean</p> <p>The motions of the specified axes <i>axes</i> are stopped. If no motion is active, the command is ignored.</p> <ul style="list-style-type: none"> • true: the motions of the specified axes have been successfully stopped • false: no motion is active or the stopping of the specified axes failed

Method	Description
<code>startCartesianJogging (coord, vel, cartJogMode, [joggingTimeInMilliseconds])</code>	<p>Return value type: boolean</p> <p>The method is used to start Cartesian jogging within defined degrees of freedom <i>coord</i> (type: EnumSet of type CartesianCoordinates) with a specified velocity <i>vel</i> (type: double).</p> <p>The parameter <i>cartJogMode</i> (type: Enum of type CartesianJoggingMode) defines the reference system relative to which the Cartesian motions are commanded (BASE, TOOL or TOOL-DIFF).</p> <p>The parameter <i>joggingTimeInMilliseconds</i> (type: long) defines a timeout for the motion; the default value is 5000 ms.</p> <p>If another motion is already active, jogging is not started and the value "false" is returned.</p> <p>If axis-specific jogging is already active, no Cartesian motion may be started and an IllegalStateException is generated.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Programmed velocity: <ul style="list-style-type: none"> – $-1.0 \leq \text{vel} \leq 1.0$ • Timeout for jogging (optional) (unit: ms) <ul style="list-style-type: none"> – ≤ 0: no timeout is defined Default: 5000 • true: Cartesian jogging has been started successfully • false: the robot was already executing a motion command and jogging is ignored
<code>updateCartesianJogging (coordinates, velocity)</code>	<p>Return value type: boolean</p> <p>The method updates the runtime parameters of an active, Cartesian jog command for the degrees of freedom <i>coordinates</i> (type: EnumSet of type CartesianCoordinates).</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Programmed velocity: <ul style="list-style-type: none"> – $-1.0 \leq \text{velocity} \leq 1.0$ • true: the parameters were updated successfully • false: the data were not updated successfully, or no Cartesian jog command is active
<code>stopCartesianJogging (coord)</code>	<p>Return value type: boolean</p> <p>The motions of the specified Cartesian degrees of freedom <i>coord</i> are stopped. If no motion is active, the command is ignored.</p> <ul style="list-style-type: none"> • true: the motions of the specified degrees of freedom have been successfully stopped • false: no motion is active or the stopping of the specified degrees of freedom failed

Method	Description
isJoggingActive()	<p>Return value type: boolean</p> <p>Returns a value indicating whether an axis-specific or Cartesian jog command is active.</p> <ul style="list-style-type: none"> • true: jogging is activated • false: jogging is deactivated
setTcp(<i>tcp</i>)	<p>Sets a freely selectable, relative reference frame <i>tcp</i> (type: ObjectFrame) for the tool of the Cartesian jog command. As standard, the DefaultMotionFrame of the robot is used. Setting the TCP during an active Cartesian motion is not permissible and triggers an IllegalStateException.</p>
setBase(<i>base</i>)	<p>Sets a freely selectable, relative reference frame <i>base</i> (type: ObjectFrame) for the base offset for Cartesian commands. By default, the base frame is located at the robot base. Setting the base during an active Cartesian motion is not permissible and triggers an IllegalStateException.</p>
stopJoggingAtAxisLimit OrSingularity (<i>activeForJointJogging</i> , <i>activeForCartesianJogging</i>)	<p>Method activates or deactivates the end of the jog command for the following causes:</p> <ul style="list-style-type: none"> • Axis limits exceeded • Singularity occurred <p>The response can be defined individually for axis-specific (<i>activeForJointJogging</i>) and Cartesian (<i>activeForCartesianJogging</i>) jog commands. By default, it is activated for both types.</p> <ul style="list-style-type: none"> • true: stop is activated • false: stop is deactivated <p>If the method is called during an active jog command, an IllegalStateException is triggered.</p>

Method	Description
getFiredStopCondition()	<p>Return value type: Enum of type JoggingStopCondition</p> <p>Once jogging has been completed, the method returns the condition which caused the motion to be terminated:</p> <ul style="list-style-type: none"> • TIMEOUT Timeout conditions have been exceeded • MOTION_IMPOSSIBLE Stop due to axis limits or singularity • COMMAND The stop command has been called • JOGGING_NEVER_STARTED No information can be provided about the break conditions, as jogging has not been started. • NONE Jogging is active. No break condition has occurred. Note: A motion stop for safety reasons is not detected. If jogging is stopped due to a safety stop, the method getFiredStopCondition() returns the value NONE. <p>To detect a motion stop:</p> <ul style="list-style-type: none"> – Poll status change messages (>>> 15.30.2.2 "Robot status log" Page 521) – Request the state of the safety signals (>>> 15.16.5 "Requesting the state of safety signals" Page 433) <p>The method returns the break conditions of the most recently called jogging (axis-specific or Cartesian). As soon as the motion command has been started and jogging is activated, the status is reset and NONE is returned.</p>

If the robot is already executing a motion (e.g. PTP, Spline, ...), jogging cannot be started. The methods for starting jogging are not taken into consideration and the value FALSE is returned.



Cartesian jogging from a singularity is only stopped with a MOTION_IMPOSSIBLE JoggingStopCondition if stopJoggingAtAxisLimitOrSingularity(...) has been set for Cartesian motions. Otherwise, Cartesian jogging remains active, even though the robot is not moving due to the singularity.



The commands for starting or stopping jogging require several milliseconds for implementation, as the commands have to be sent to the controller. If the time at which jogging is to be stopped is critical, the parameter joggingTimeInMilliseconds > 0 must be selected when starting jogging. Only then is a stop carried out immediately after the time has elapsed. The command for stopping jogging should only be used if the stopping of jogging is not time-critical.

Example

Various conditions are polled in this example:

```
LBR lbr = (LBR) getRobot(kuka_Sunrise_Cabinet_1, "LBR");
...
// instantiate the jogging
Jogging jogging = new Jogging(lbr);
```

```
// start joint jogging of the second axis
// with velocity of 0.3 for 3000ms
jogging.startJointJogging(EnumSet.of(JointEnum.J2), 0.3,
3000);
// wait for one second and then update the motion
parameters
ThreadUtil.milliSleep(1000);
jogging.updateJointJogging(EnumSet.of(JointEnum.J2), -0.3);
// wait two more seconds for the motion
// to be finished due to the timeout
ThreadUtil.milliSleep(2000);
// start Cartesian jogging motion
// with velocity 0.3 for 1000ms relative to the base
jogging.startCartesianJogging(EnumSet.of(CartesianCoordinates
.X),
0.3, CartesianJoggingMode.BASE, 1000);
// stop the motion even before the timeout expires
jogging.startCartesianJogging(EnumSet.of(CartesianCoordinates
.X));
```


16 Background tasks

16.1 Using background tasks

Activities

Background tasks are used in order to be able to perform tasks in the background, parallel to a running robot application, or to implement cyclical processes that are to be run continuously in the background. Multiple background tasks can run simultaneously and independently of the running robot application.

Background tasks are used, in particular, to control and monitor peripheral devices and to implement the corresponding higher-level logic. Examples:

- Switching signal lamps
- Monitoring and evaluating sensor information

This means that no higher-level controller, e.g. a PLC, is required for smaller applications, as the robot controller can perform such tasks by itself.



In the case of outputs that are switched by a background task, the following points must be observed:

- The outputs are switched, irrespective of whether a robot application is currently being executed.
- The outputs are also switched if the robot application is paused due to an EMERGENCY STOP or missing enabling signal.
- The outputs are also switched if a stop request from the safety controller is active (this also applies if outputs are switched by a robot application).



WARNING

Background tasks must not be used for moving the robot or influencing parameters that might affect motions. This is the task of the robot application. Calling motion commands or modifying motion-specific parameters in a background task can result in unspecified behavior of the robot and thus cause personal injury and damage to property.

Properties

Background tasks, like robot applications, are implemented as Java classes. They are similar in structure to robot applications: they have a run() method that contains the commands to be executed.

Background tasks are an integral feature of the Sunrise project. They are created in Sunrise.Workbench and transferred to the robot controller when the project is synchronized.

(>>> [5.4.5 "Creating a new background application" Page 65](#))

There are 2 types of background task that differ in terms of their duration:

- Cyclic background task
Executed cyclically. The cyclical behavior can be adapted by the programmer depending on the task to be performed.
- Non-cyclic background task
Executed once.

Background tasks also differ in terms of their start type:

- **Manual**

The task must be started manually via the smartPAD. (This function is not yet supported.)

- **Automatic**

The task is automatically started when the robot controller is booted and stopped when it is shut down.

In background tasks many objects of the application can be accessed by means of dependency injection.

(>>> *15.3.3 "Dependency injection" Page 379*)

If a background task requires access to information from the running robot application or other background tasks that are not accessible via dependency injection, a separate interface is available for data exchange.

(>>> *16.4 "Data exchange between tasks" Page 538*)

Synchronization behavior

When synchronizing the Sunrise project, the associated background tasks with **Automatic** start type exhibit the following behavior:

- Tasks not yet present

Both cyclic and non-cyclic tasks are transferred to the controller and subsequently started.

- Tasks already present

If the task to be synchronized (cyclic or non-cyclic) is already present on the controller, it will be terminated if it is still running. The synchronization is then executed and the task automatically restarted.

- Tasks no longer present

If a background task has been deleted from the associated project and synchronization is carried out, the task is terminated before synchronization on the controller. It is then no longer available after synchronization.

Runtime behavior

After a non-cyclic task has been started, it is executed fully in accordance with its programming. When it reaches the end of its run() method, it is terminated and not restarted until the next synchronization or the next reboot of the controller.

When started, a cyclic task is first instanced. The run() method of the task is then repeatedly called on a regular basis. These background tasks are therefore permanently executed as long as the controller is running.

If an error which cannot be intercepted and rectified occurs in a task (cyclic or non-cyclic), the task is automatically terminated.



If a background task has been terminated because of an unhandled error, the task can only be restarted by rebooting the robot controller or carrying out project synchronization of Sunrise.Workbench on the robot controller.

16.2 Cyclic background task

Structure

```

1 package backgroundTask;
2
3 public class BackgroundTask extends RoboticsAPICyclicBackgroundTask {
4     @Inject
5     Controller kuka_Sunrise_Cabinet_1;
6
7     @Override
8     public void initialize() {
9         // initialize your task here
10        initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
11                        CycleBehavior.BestEffort);
12    }
13
14    @Override
15    public void runCyclic() {
16        // your task execution starts here
17    }
18}

```

Fig. 16-1: Structure of a cyclic background task

Item	Description
1	This line contains the name of the package in which the task is located.
2	Import section The section contains the imported classes which are required for programming the task
3	Header of the task The cyclic background task is a subclass of RoboticsAPICyclicBackgroundTask.
4	Declaration section The data arrays of the task that are required for its execution are declared here. As an example, the controller is automatically integrated via dependency injection when the task is created.
5	initialize() method Initial values are assigned here to data arrays that are not integrated using dependency injection. The initializeCyclic(...) method is available as standard. This method is used to define the cyclical behavior of the task. (>>> <i>"Initialization" Page 535</i>) Note: The method must not be deleted or renamed.
6	runCyclic() method The code that is to be executed cyclically is programmed here. Note: The method must not be deleted or renamed.

Initialization

initializeCyclic(...) is used to define the cyclical behavior of the background task.

When a cyclical background task is created, the call for `initializeCyclic(...)` is automatically inserted. The input parameters of the method are assigned initial values, which can lead to the following cyclical behavior:

- Delay: 0 ms
- Period: 500 ms
- Behavior if the defined period is exceeded: execution of `runCyclic()` continues.

The initial values can be changed by the programmer.

```
initializeCyclic(long initialDelay, long period, TimeUnit
timeUnit, CycleBehavior behavior);
```

Element	Description
<i>initialDelay</i>	Delay after which the cyclical background task is executed for the first time after the start. All further cycles are executed without a delay. The time unit is defined with <i>timeUnit</i> .
<i>period</i>	Period (= time between 2 calls of <code>runCyclic()</code>) The period is maintained even if the execution time of <code>runCyclic()</code> is less than the defined period. The behavior in the event of <code>runCyclic()</code> exceeding the period is defined by <i>behavior</i> . The time unit is defined with <i>timeUnit</i> .
<i>timeUnit</i>	Time unit of <i>initialDelay</i> and <i>period</i> The Enum <code>TimeUnit</code> is an integral part of the standard Java library.
<i>behavior</i>	Timeout behavior The behavior of the background task if the period defined with <i>period</i> is exceeded by the runtime of <code>runCyclic()</code> is defined here. <ul style="list-style-type: none"> • CycleBehavior.BestEffort <code>runCyclic()</code> is executed completely and then called again. • CycleBehavior.Strict Execution of the background task is canceled with an error of type <code>CycleExceededException</code>.

Example

A robot is to assemble workpieces that it takes from a magazine. The magazine can contain a maximum of 100 workpieces and is loaded manually. If the remaining number of workpieces in the magazine falls below 20, this is signaled to the robot controller via a digital input. An LED is then to flash every 500 ms to signal to the operator that the magazine needs filling. Another LED is to flash if the force determined at the robot flange exceeds a limit of 150 N.

A cyclic background task is used for data evaluation and activation of the LEDs. The background task is executed every 500 ms.

```
public class LEDTask extends
RoboticsAPICyclicBackgroundTask {
    @Inject
    private LBR robot;
    @Inject
    private ProcessParametersIOGroup processParaIOs;
```

```

@Inject
private ProcessParametersLEDsIOGroup LED_IOS;

public void initialize() {
    initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
        CycleBehavior.BestEffort);
}

public void runCyclic() {
    /**
     * Check if refill is required (value true)
     */
    if (processParaIOs.getSensor_RefillRequired()) {
        /**
         * If refill is required, the appropriate LED changes
         * its state with every execution of runCyclic()
         */
        LED_IOS.setLED_RefillRequired(!LED_IOS
            .getLED_RefillRequired());
    } else {
        /**
         * If refill is not required, the LED remains off
         */
        LED_IOS.setLED_RefillRequired(false);
    }

    /**
     * Query the applied force
     */
    Vector forceVector = robot.getExternalForceTorque(
        robot.getFlange()).getForce();

    /**
     * Check if absolute force (length of vector)
     * exceeds 150 N
     */
    if (forceVector.length() > 150.0) {
        /**
         * If the force limit is exceeded, the appropriate LED
         * changes its state with every execution of runCyclic()
         */
        LED_IOS.setLED_ForceExceeded(!LED_IOS
            .getLED_ForceExceeded());
    } else {
        LED_IOS.setLED_ForceExceeded(false);
    }
}

```

16.3 Non-cyclic background task

Structure

```

1 package backgroundTask;
2
3 import javax.inject.Inject;
4
5 * Implementation of a background task.
6 public class BackgroundTask1 extends RoboticsAPIBackgroundTask {
7     @Inject
8     Controller kuka_Sunrise_Cabinet_1;
9
10    @Override
11    public void initialize() {
12        // initialize your task here
13    }
14
15    @Override
16    public void run() {
17        // your task execution starts here
18    }
19 }

```

Fig. 16-2: Structure of a non-cyclic background task

Item	Description
1	This line contains the name of the package in which the task is located.
2	Import section The section contains the imported classes which are required for programming the task
3	Header of the task The non-cyclic background task is a subclass of RoboticsAPI-BackgroundTask.
4	Declaration section The data arrays of the task that are required for its execution are declared here. As an example, the controller is automatically integrated via dependency injection when the task is created.
5	initialize() method Initial values are assigned here to data arrays that are not integrated using dependency injection. Note: The method must not be deleted or renamed.
6	run() method The code that is to be executed once is programmed here. The runtime is not limited. Note: The method must not be deleted or renamed.

16.4 Data exchange between tasks

Description

The mechanism described here can be used to exchange data between running tasks. One task can provide task functions (providing task) that can be accessed by other tasks (requesting tasks).

Example: Accessing and processing information from the running robot application in a background task.

It is not relevant for programming whether data are exchanged between a background task and a robot application or between 2 background tasks. The providing task may be either a robot application or a background task. For this reason, background tasks and robot applications are grouped together as tasks.

Overview

The following steps are required in order for the providing task and the requesting task to be able to communicate with one another:

Step	Description
1	Create an interface and declare the desired task functions. (>>> 16.4.1 "Declaring task functions" Page 540)
2	Implement the interface in which the task functions are declared. The interface can be implemented directly by the providing task or by a specially created class. The declared task functions must be programmed in the implementing class. (>>> 16.4.2 "Implementing task functions" Page 540)
3	Create the providing task. The providing task must contain a parameterless public method with the annotation @TaskFunctionProvider which returns the implementation of the interface. (>>> 16.4.3 "Creating the providing task" Page 542)
4	In the requesting task, use the getTaskFunction(...) method to request the interface in which the task functions are declared. The method is available in all task classes. The ITaskFunctionMonitor interface can be used to check whether the task functions are available. (>>> 16.4.4 "Using task functions" Page 544)

Example

The data exchange between tasks is described step by step in the sections below, using the following example:

An assembly process is to be implemented using the robot application “AssemblyApplication”. An LED is to flash during the assembly process. If the robot leaves the path during the application and has to be repositioned, a further LED is to flash.

The LEDs are activated by the background task “LEDTask”. In this example, the background task is the requesting task.

The robot application is the providing task. It must enable access to your Recovery interface, which is used to check whether repositioning of the robot is required. Furthermore, it must also signal the start and end of the assembly process.

16.4.1 Declaring task functions

Description

The desired task functions must be declared in a specially created interface.



The interface may only declare those methods that are to be made available to the requesting task. It is thus advisable not to use set methods for setting fields in the interface. Instead, such methods can be offered by the implementing class.

Example

Declaration of the task functions via the interface IApplicationInformationFunction

The following methods are to be available to the providing task (here the background task) and are declared by the IApplicationInformationFunction interface:

- `isAssemblyRunning()`: checks whether an assembly process is currently being executed
- `isManualRepositioningRequired()`: checks whether repositioning of the robot is required

```

1 public interface IApplicationInformationFunction {
2
3   /**
4    * Signifies whether assembly is currently executed
5    * @return true, if assembly is executed
6   */
7   public boolean isAssemblyRunning();
8
9   /**
10    * Returns whether the application requires
11    * repositioning of the robot
12    * @return true if repositioning is required
13   */
14  public boolean isManualRepositioningRequired();
15
16 }
```

Line	Description
1 ... 16	Interface IApplicationInformationFunction
7	Method <code>isAssemblyRunning()</code> Called by the requesting task to check whether the assembly process is currently running.
14	Method <code>isManualRepositioningRequired()</code> Called by the requesting task to check whether repositioning of the robot is required.

16.4.2 Implementing task functions

Description

A class must be made available that implements the interface and in which the declared task functions are programmed. The providing task or a specially created class can be used as the implementing class.

Example

Implementation of the `IApplicationInformationFunction` interface using the `ApplicationInformation` class

The following methods are only to be available to the providing task (here the robot application) and are declared and implemented by the `ApplicationInformation` class:

- `setAssemblyRunning(...)`
Announces the start and end of the assembly process
- `setApplicationRecoveryInterface(...)`
Enables access to the Recovery interface of the robot application

```

1  public class ApplicationInformation implements
2    IApplicationInformationFunction {
3
4    private boolean assembly;
5    private IRecovery recovery;
6
7    @Override
8    public boolean isAssemblyRunning() {
9      return assembly;
10 }
11
12 /**
13  * Called from application when assembly is
14  * started and finished
15  * @param assembly Set to true when assembly is
16  * started.
17  * Reset when assembly is stopped.
18  */
19 public void setAssemblyRunning(boolean isRunning) {
20   assembly = isRunning;
21 }
22
23 @Override
24 public boolean isManualRepositioningRequired() {
25   return recovery.isRecoveryRequired();
26 }
27
28 /**
29  * Called from application to give access to its
30  * recovery interface
31  * @param applicationRecoveryInterface Recovery
32  * interface of the application
33  */
34 public void setApplicationRecoveryInterface(
35   IRecovery recoveryInterface) {
36   recovery = recoveryInterface;
37 }
```

Line	Description
1 ... 37	Class ApplicationInformation The task functions are programmed in the class.
4, 5	Declaration of the data arrays <ul style="list-style-type: none"> • boolean assembly: Indicates whether a joining process is currently active. • IRecovery recovery: refers to the Recovery interface of the robot application
7 ... 10	Method isAssemblyRunning() Called by the requesting task to check whether the assembly process is currently running.
18 ... 20	Method setAssemblyRunning(...) Called by the robot application when the assembly process is started or ended.
22 ... 25	Method isManualRepositioningRequired() Called by the requesting task to check whether repositioning of the robot is required.
33 ... 36	Method setApplicationRecoveryInterface(...) Called by the robot application for making its Recovery interface available.

16.4.3 Creating the providing task

Description

A task can provide task functions of various interfaces. For each interface whose task functions are provided by the task, a parameterless public method with the annotation `@TaskFunctionProvider` must be inserted which returns the implementation of the interface.

Syntax

```
@TaskFunctionProvider
public Interface Method name()
    return Interface instance;
}
```

Explanation of the syntax

Element	Description
<i>Interface</i>	Interface whose task functions the task provides
<i>Method name</i>	Name of the method that returns the implementation of the interface (the name can be freely selected)
<i>Interface instance</i>	Instance of the implementing class



If the providing task does not, itself, implement the interface derived from `ITaskFunction`, it requires an instance of the implementing class. It is advisable to create this instance as an array.



If the providing task implements the interface itself, transfer the instance of the task for the *Interface instance* parameter:

```
return this;
```



Each interface may only be provided once. This means that there must not be 2 tasks that return the same interface in their `@TaskFunctionProvider` annotation.

Example

The robot application contains a data array of type `ApplicationInformation`. Its method `setApplicationRecoveryInterface(...)` provides the Recovery interface of the robot application. Calling the method `setAssembly(...)` announces that the assembly process is being carried out.

```
public class AssemblyApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    private ApplicationInformation appInfo;

    @Override
    public void initialize() {

        appInfo = new ApplicationInformation();
        // Gives access to recovery interface
        appInfo.setApplicationRecoveryInterface(getRecovery());

    }

    @Override
    public void run() {

        // Moves robot to initial pose
        robot.move(ptp(getFrame("/StartPos")));

        // Announces that assembly is running
        appInfo.setAssemblyRunning(true);
        assembly();

        // Announces that assembly is finished
        appInfo.setAssemblyRunning(false);

        // Moves robot to initial pose
        robot.move(ptp(getFrame("/StartPos")));

    }

    /**
     * Implements the assembly process
     */
    private void assembly() {
        // ...
    }

    /**
     * TaskFunctionProvider method that has to be
     * implemented by the task
     */
    @TaskFunctionProvider
```

```
public IApplicationInformationFunction getAppInfoFunction()
{
    return appInfo;
}
```

16.4.4 Using task functions

Task functions provided by a task can be used by other tasks.

Enabling access

The following steps are required in order to enable access to the task functions of an interface in the requesting task:

1. Create a data array of the type of the interface.

```
private Interface Interface instance;
```

2. Request the interface with the getTaskFunction(...) method. The task functions of the interface are saved in the data array just created.

```
Interface instance = getTaskFunction(Interface.class);
```

Explanation of the syntax:

- *Interface*: interface whose task functions the task wants to access
- *Interface instance*: instance of the interface in which the task functions are declared

Example:

In the requesting background task “LEDTask”, access to the functions defined by IApplicationInformationFunction is to be enabled. The interface instance required for this is created as a data array and generated in the initialize() method of the task:

```
public class LEDTask extends
RoboticsAPICyclicBackgroundTask {
// ...
private IApplicationInformationFunction appInfoFunction;

public void initialize() {
    initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
        CycleBehavior.BestEffort);

    // Get Task Function Interface
    appInfoFunction = getTaskFunction(
        IApplicationInformationFunction.class);
}
```

Checking availability

The task functions of the providing task are only available when the providing task is being executed or is paused.



If a task function is not available when it is called, a runtime error may occur in the requesting task (InstanceNotRunningException). If this error is not handled, execution of the task is aborted.

The methods of the ITaskFunctionMonitor interface can be used to check whether the task functions of the providing task are available.

The following steps are required in order to make the methods of the interface available in the requesting task:

1. Create a data array of the type of the interface.

```
private ITaskFunctionMonitor monitor;
```

2. Use the TaskFunctionMonitor.create(...) method to initialize the monitor for the task functions to be monitored. The instance of the interface in which the task functions are declared is transferred to the method as a parameter.

```
monitor = TaskFunctionMonitor.create(Interface instance);
```

Explanation of the syntax:

- *monitor*: instance of the ITaskFunctionMonitor interface
- *Interface instance*: instance of the interface in which the task functions are declared

The following methods are available in the ITaskFunctionMonitor interface:

Method	Description
isAvailable()	<p>Return value type: boolean</p> <p>Specifies whether the task functions of the providing task are available (true = available).</p>
await(<i>time</i> , <i>unit</i>)	<p>Return value type: boolean</p> <p>If the task functions are not available when the providing task is called, the system waits a defined time for them to become available (true = task functions available within the defined wait time).</p> <p>Parameter:</p> <ul style="list-style-type: none"> • <i>time</i> (type: long): maximum waiting time. The unit is defined by the parameter <i>unit</i>. • <i>unit</i> (type: TimeUnit): unit of <i>time</i>

Example:

In the method runCyclic() of the background task “LEDTask”, a check is to be carried out to ascertain whether the assembly process is currently being executed. For this, the interface IApplicationInformationFunction offers the method isAssemblyRunning().

The requesting background task “LEDTask” can only check whether the assembly process is being executed if the robot application is running or paused. For this reason, the availability of the function must be checked before isAssemblyRunning() is called:

```
public class LEDTask extends
RoboticsAPICyclicBackgroundTask {
    // ...
    private IApplicationInformationFunction appInfoFunction;
    private ITaskFunctionMonitor appInfoMonitor;

    public void initialize() {
        initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
            CycleBehavior.BestEffort);

        // Get Task Function Interface
        appInfoFunction = getTaskFunction(
            IApplicationInformationFunction.class);

        /**
         * Create ITaskFunctionMonitor for
         * IApplicationInformationFunction
         */
        appInfoMonitor = TaskFunctionMonitor.create(
            appInfoFunction);
```

```
}

public void runCyclic() {
    // Check if task functions are available
    if (appInfoMonitor.isAvailable()) {
        /**
         * Use task function to check if assembly is
         * currently executed
         */
        if (appInfoFunction.isAssemblyRunning()) {
            // ...
        }
    }
}
```

Overall example

The requesting task “LED Task” is executed cyclically every 500 ms. It first checks whether the required task functions of the robot application are available. If they are available, a check is carried out to ascertain whether the assembly process is running and the corresponding LED is activated. The system then checks whether repositioning of the robot is required. If this is the case, a further LED is activated.

```
public class LEDTask extends
RoboticsAPICyclicBackgroundTask {
    @Inject
    private ProcessParametersLEDsIOGroup LED_IOs;
    private IApplicationInformationFunction appInfoFunction;
    private ITaskFunctionMonitor appInfoMonitor;

    public void initialize() {
        initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
            CycleBehavior.BestEffort);

        // Get Task Function Interface
        appInfoFunction = getTaskFunction(
            IApplicationInformationFunction.class);

        /**
         * Create ITaskFunctionMonitor for
         * IApplicationInformationFunction
         */
        appInfoMonitor = TaskFunctionMonitor
            .create(appInfoFunction);
    }

    public void runCyclic() {
        // Check if task functions are available
        if (appInfoMonitor.isAvailable()) {
            /**
             * Use task function to check if assembly is
             * currently executed
             */
            if (appInfoFunction.isAssemblyRunning()) {
                /**
                 * If assembly is running, the appropriate LED
                 * changes
                 * its state with every execution of runCyclic()
                 */
                boolean currentStateAssemblyLED = LED_IOs
                    .getLED_Assembly();
```

```
LED_IOs.setLED_Assembly(!currentStateAssemblyLED);
} else {
    LED_IOs.setLED_Assembly(false);
}

/**
 * Use task function to check whether the application
 * requires repositioning
 */
boolean recoveryRequired =
    appInfoFunction.isManualRepositioningRequired();

if (recoveryRequired) {
    /**
     * If recovery is required,
     * the appropriate LED changes
     * its state with every execution of runCyclic()
     */
    boolean currentStateRecoveryLED = LED_IOs
        .getLED_RecoveryRequired();
    LED_IOs.setLED_ForceExceeded(
        !currentStateRecoveryLED);
} else {
    LED_IOs.setLED_RecoveryRequired(false);
}
} else {
    // If application is not running, LEDs remain off
    LED_IOs.setLED_Assembly(false);
    LED_IOs.setLED_RecoveryRequired(false);
}
}
```


17 KUKA Sunrise.EnhancedVelocityControl

17.1 Overview of KUKA Sunrise.EnhancedVelocityController

KUKA Sunrise.EnhancedVelocityController (EVC) is an add-on option for limiting the Cartesian robot velocity.

Description

EVC regulates the velocity of the following frames:

- Robot flange
- TCP

EVC automatically adapts the robot velocity so that the following Cartesian velocity limits are not exceeded by these frames:

- Cartesian velocity limits that are active on the safety controller
Cartesian velocity monitoring functions can be combined in the safety configuration with the “Brake” safety reaction.
(>>> [17.2 “Brake” safety reaction” Page 550](#))
- Mode-specific Cartesian velocity limitation of 250 mm/s that is active in T1 or CRR mode
- Device-specific Cartesian velocity limitation that is set by the application
(>>> [17.3 “Cartesian velocity limitation via application” Page 552](#))
- Aggregated Cartesian velocity limitation
If multiple Cartesian velocity limitations are active simultaneously, the velocity is reduced to the lowest of these limits.

Motion types

EVC limits Cartesian velocities in position-controlled spline motions:

- Impedance-controlled motions and motions that are not spline motions, e.g. manual guidance motions, are not compatible with EVC.

Functional principle

EVC takes safety-oriented velocity limits into account, thereby preventing the robot from being stopped with a safety stop:

- If a Cartesian velocity monitoring function is active, the velocity of the commanded TCP and the flange is automatically reduced so that the monitored velocity limit is not exceeded by these frames. The robot structure and the safety-oriented frames of active tools are not taken into consideration.
- If multiple Cartesian velocity monitoring functions are active simultaneously, the velocity is automatically reduced so that the lowest currently monitored velocity limit is not exceeded.

The velocity is always reduced to 90% of the lowest current velocity limit in order to maintain a buffer between the exact limit and the target velocity. If, for example, only the mode-specific Cartesian velocity limitation of 250 mm/s is active, EVC regulates the velocity to 225 mm/s.

17.2 “Brake” safety reaction

Description

With EVC, the “Brake” safety reaction is available for safety functions of the PSM mechanism used for monitoring the Cartesian velocity.

Brake can only be used as a reaction if the safety function (PSM row) contains the *Cartesian velocity monitoring* AMF. The PSM row can contain further AMFs. An exception is extended AMFs, such as the *Time delay* AMF. Extended AMFs cannot be used in conjunction with the “Brake” reaction.

Benefits

The “Brake” safety reaction can be used to prevent the robot being stopped with a safety stop if its velocity is higher than the configured limit when the velocity monitoring is activated.



With very high accelerations, it is possible, in rare cases, that the velocity limitation does not act quickly enough. This can result in the robot stopping with a safety stop 1. Possible remedy: Reduce the acceleration for the motions in which this occurs.



The “Brake” safety reaction is only compatible with position-controlled spline motions. In the case of incompatible motion and control types (e.g. manual guidance, impedance control), there is no automatic braking with the drives. If the velocity does not decrease, the safety controller triggers a safety stop 1. For this reason, it is advisable not to use the “Brake” safety reaction with incompatible motion and control types. Unlike safety stop 1 (path-maintaining), safety stop 1 leads to increased response times and the programmed path is left.



If the “Brake” safety reaction is used, it must be taken into consideration that the robot velocity is only reduced to the limit value of the Cartesian velocity monitoring configured in a PSM row for as long as all other AMFs of the PSM row are violated. If the violation state of these other AMFs is rapidly switched backwards and forwards, it is possible that the “Brake” safety reaction does not lead to a reduction in velocity. For every PSM row with the “Brake” safety reaction, a check must be made to see whether there could be an increased risk due to rapid switching to and from the violation state of the AMFs with which the Cartesian velocity monitoring is linked. This is the responsibility of the safety maintenance technician.

Example 1

Cartesian velocity monitoring is activated with an input signal (LOW), e.g. a laser scanner.

AMF1	AMF2	AMF3	Reaction
<i>Input signal</i>	<i>Cartesian velocity monitoring</i>	-	<i>Brake</i>

If the laser scanner detects a person and the robot is too fast, i.e. its velocity is above the configured limit, the “Brake” reaction is triggered. The resulting braking process is monitored and the velocity continues being reduced until it is below the configured limit. The *Cartesian velocity monitoring* AMF is then no longer violated and the “Brake” reaction is terminated. As long as the input signal has the LOW level, EVC keeps the velocity below this limit value.

Example 2

Cartesian velocity monitoring is combined with protected space monitoring.

AMF1	AMF2	AMF3	Reaction
<i>Cartesian protected space monitoring</i>	<i>Cartesian velocity monitoring</i>	-	<i>Brake</i>

In a collaboration zone (HRC zone), the Cartesian velocity is to be reduced in order to minimize the risk for the operator. This can be achieved by combining Cartesian velocity monitoring with protected space monitoring.

As soon as the robot enters the protected space, the velocity monitoring is active. If the velocity is too high on entering the protected space, the "Brake" reaction is triggered. The resulting braking process is monitored and the velocity continues being reduced until it is below the configured limit.

The Cartesian velocity monitoring AMF is then no longer violated and the "Brake" reaction is terminated. As long as the protected space is violated, EVC keeps the velocity below the configured limit. If the robot leaves the protected space again, the velocity monitoring is no longer active and EVC no longer limits the velocity.

Brake ramp monitoring

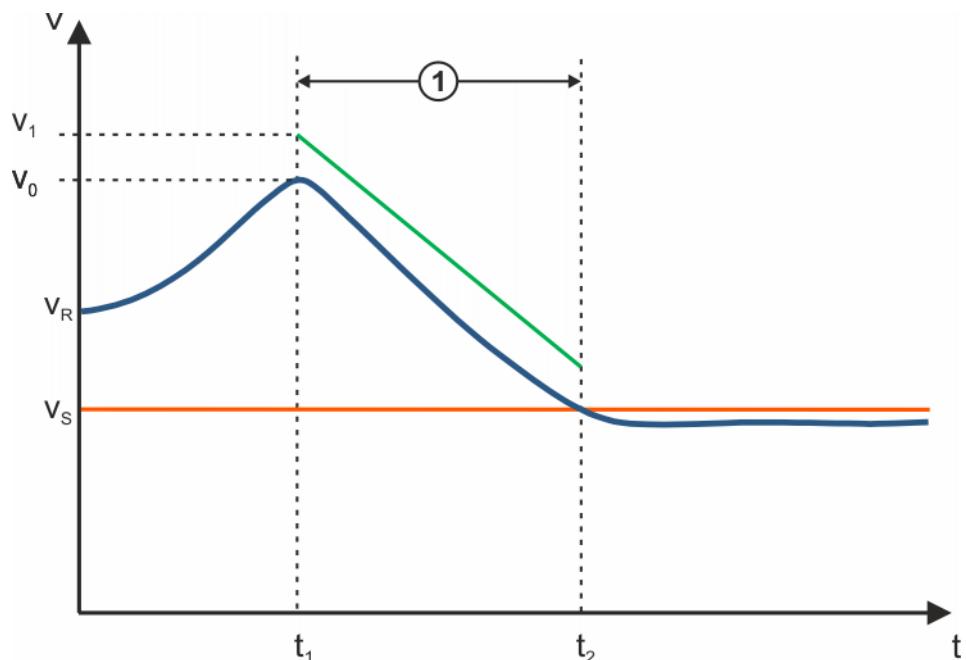


Fig. 17-1: Braking ramp monitoring

1 Monitoring time

v Velocity

t Time

t_1 Moment in time at which the "Brake" reaction is triggered (PSM row is violated)

t_2 Moment in time at which the "Brake" reaction is stopped (Cartesian velocity monitoring AMF is no longer violated)

v_R Robot velocity (blue curve)

v_S Velocity limit of the *Cartesian velocity monitoring* AMF

v_0 Velocity at time t_1 at which the "Brake" reaction is triggered

v_1 Start value of the monitoring ramp (green curve)
 $v_1 = v_0 + 250 \text{ mm/s}$

17.3 Cartesian velocity limitation via application

EVC can limit the Cartesian robot velocity via the application. Once set, this device-specific Cartesian velocity limitation can be deactivated again via the application. Additionally, information about all Cartesian velocity limitations carried out by EVC can be requested by the robot.

17.3.1 Setting and deactivating velocity limitation

Description

The robot class contains the methods that can be used to set device-specific Cartesian velocity monitoring in the application and then deactivate it again.

Syntax

Set velocity limitation:

```
robot.setDeviceCartesianVelocityLimit(limit);
```

Deactivate velocity limitation:

```
robot.deactivateDeviceCartesianVelocityLimit();
```

Explanation of the syntax

Element	Description
<i>robot</i>	Type: Robot Instance of the robot used in the application
<i>limit</i>	Type: int Value > 0 that is set as the Cartesian velocity limit for the robot (unit: mm/s)

Example

```
public class RobotApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    // ...

    @Override
    public void initialize() {
        // initialize your application here
    }

    @Override
    public void run() {
        // your application execution starts here
        int integerValue = 69;      //in mm/s

        robot.move(ptpHome());

        //Sets "integerValue" as the new device limit
    }
}
```

```

    robot.setDeviceCartesianVelocityLimit(integerValue);

    //Deactivates the device limit
    robot.deactivateDeviceCartesianVelocityLimit();
}
}

```

17.3.2 Requesting information about velocity limitation functions

Description

The method `getCartesianVelocityLimitInfo()` can be used to request information about the Cartesian velocity limitations from the robot and store it in a container of the `CartesianVelocityLimitInfo` class.

The following information is stored:

- The mode-specific Cartesian velocity limitation
- The device-specific Cartesian velocity limitation that is set by the application
- The Cartesian velocity limit that is active on the safety controller
- The aggregated Cartesian velocity limitation
- The sources from which the value for the aggregated Cartesian velocity limitation was formed

The information stored in the container can be read.

Syntax

```
CartesianVelocityLimitInfo infoObject =
robot.getCartesianVelocityLimitInfo();
```

Explanation of the syntax

Element	Description
<i>infoObject</i>	Type: <code>CartesianVelocityLimitInfo</code> Variable for the information requested from the robot using <code>getCartesianVelocityLimitInfo()</code>
<i>robot</i>	Type: <code>Robot</code> Instance of the robot used in the application

Overview

The information stored in the containers can be read using the following methods:

Method	Description
<code>getDeviceVelocityLimit()</code>	Return value type: Integer Returns the device-specific Cartesian velocity limitation that is currently set by an application (unit: mm/s). The return value -1 means that the device-specific Cartesian velocity limitation is deactivated.
<code>getOperationModeVelocityLimit()</code>	Return value type: Integer Returns the mode-specific Cartesian velocity limitation (unit: mm/s). The return value -1 means that the Cartesian velocity is not currently limited by an operating mode.

Method	Description
getSafetyVelocityLimit()	<p>Return value type: Integer</p> <p>Returns the Cartesian velocity limit that is active on the safety controller (unit: mm/s).</p> <p>The return value -1 means that there is currently no Cartesian velocity limit active on the safety controller.</p>
getAggregatedVelocityLimit()	<p>Return value type: Integer</p> <p>Returns the minimum of all currently active and valid Cartesian velocity limitations (unit: mm/s). This aggregated velocity value is the actual limitation that regulates the robot velocity.</p> <p>The return value -1 means that the Cartesian velocity is not currently limited. In other words, it is not currently limited by either the operating mode or the application and there is no active Cartesian velocity limit on the safety controller.</p>
getVelocityLimitSources()	<p>Return value type: Set<CartVelocityLimitSourceType></p> <p>Returns an Enum data set with the sources from which the value for the aggregated Cartesian velocity limitation was formed.</p> <p>The Enum CartVelocityLimitSourceType contains the following values:</p> <ul style="list-style-type: none"> • DEVICE Array for device-specific Cartesian velocity limitation • OPERATIONMODE Array for mode-specific Cartesian velocity limitation • SAFETY Array for Cartesian velocity limit that is active on the safety controller

Example

```

public class RobotApplication extends
RoboticsAPIApplication {
    @Inject
    private LBR robot;
    private Integer deviceVelocityLimit;
    private Integer safetyVelocityLimit;
    private Integer operationModeVelocityLimit;
    private Integer aggregatedVelocityLimit;
    private Set<CartVelocityLimitSourceType>
velocityLimitSources;

    @Override
    public void initialize() {
        // initialize your application here
    }

    @Override
    public void run() {
        // your application execution starts here
        int integerValue = 69;      //in mm/s

        robot.move(ptpHome());
    }
}

//All limit data is queried from the robot

```

```
CartesianVelocityLimitInfo infoObject = robot
    .getCartesianVelocityLimit();

deviceVelocityLimit = infoObject.getDeviceVelocityLimit();
safetyVelocityLimit = infoObject.getSafetyVelocityLimit();
operationModeVelocityLimit = infoObject
    .getOperationModeVelocityLimit();

aggregatedVelocityLimit = infoObject
    .getAggregatedVelocityLimit();
velocityLimitSources = infoObject
    .getVelocityLimitSources();

//Sets "integerValue" as the new device limit
robot.setDeviceCartesianVelocityLimit(integerValue);

//Deactivates the device limit
robot.deactivateDeviceCartesianVelocityLimit();
}
}
```


18 KUKA Sunrise.StatusController

18.1 Overview of KUKA Sunrise.StatusController

Description

KUKA Sunrise.StatusController is a programming interface that can be used to signal various system states. It can be used in robot and background applications.

The status controller distinguishes between status group and status. Various statuses are grouped together in a status group, e.g. the status group SAFETY_STOP contains the statuses that belong to a safety stop.

Examples of statuses of the status group SAFETY_STOP:

- E-STOP actuated on smartPAD
- Safety configuration not activated
- Safety stop 0 active
- etc.

Functional principle

Some statuses are automatically set by the station monitoring, e.g. whether a safety stop is active. Additionally, user-defined statuses can be set via the status monitor of a task. A status listener can then be used to respond to status changes, e.g. switching on of a red lamp in the case of an error state.

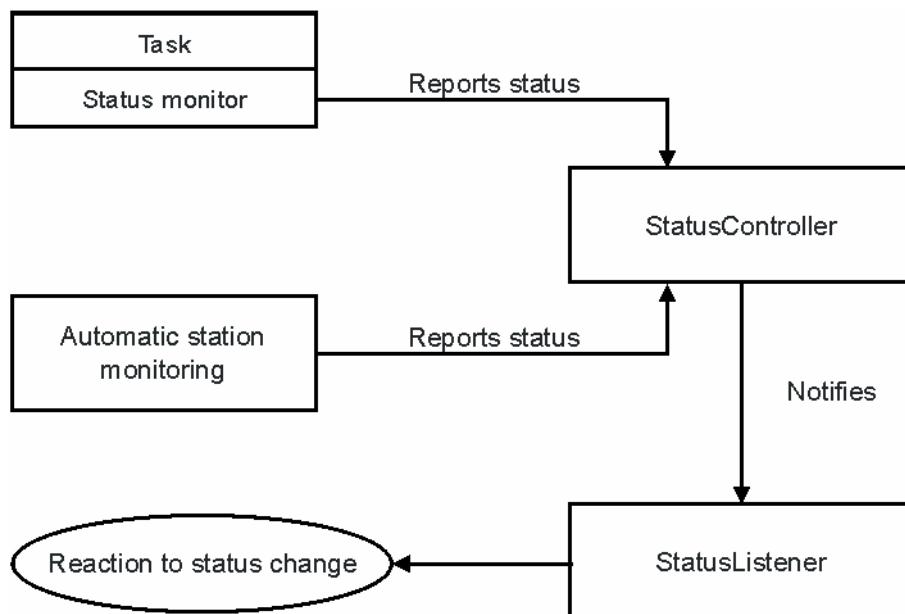


Fig. 18-1: Functional principle of status controller

Installation

KUKA Sunrise.StatusController is contained in the software catalog of Sunrise.Workbench and must be installed in the Sunrise project:

Procedure

1. Open the station configuration (file **StationSetup.cat**) in the project.
2. Set the check mark next to the entry **KUKA Sunrise.StatusController** in the **Install** column on the **Software** tab.

3. Save the station configuration and apply changes to the project.



The software expansions installed in the Sunrise project must be transferred to the robot controller. The System Software must be reinstalled on the robot controller for this.

Interfaces

KUKA Sunrise.StatusController provides the following interfaces:

- IStatusController
Interface with the methods of the status controller
 - Automatically set statuses and statuses set by the system integrator on a status monitor are signaled to the status controller.
 - If a status listener is registered on the status controller, the status listener is notified of status changes.
 - All currently active statuses and status groups can be requested via the status controller.
- IStatusMonitor
Interface for setting the status from robot and background applications
- IStatusListener
Interface for responding to status changes

Classes

KUKA Sunrise.StatusController provides the following classes:

- DefaultStatusGroups class
The class contains predefined status groups.
- StatusGroup class
Objects of the class each represent a status group.
- Status class
Objects of the class each represent a status of a status group.

18.1.1 Predefined status groups

The following status groups are defined in the class DefaultStatusGroups. For some of these groups, the statuses are automatically set if certain preconditions are met. For other groups, the statuses must be generated and set by the system integrator.

Status groups whose statuses are only set automatically (cannot be used by the system integrator):

Status group	Description
APPLICATION_READY	A robot application is present and can be started. The status is set if the following preconditions are met: <ul style="list-style-type: none"> • Motion enable signal has been received for all kinematic systems. • All robots are mastered. • AUT mode • There is at least one robot application on the robot controller. • No robot application running
APPLICATION_RUNNING	A robot application is running in AUT mode and is not paused.

Status group	Description
SAFETY_STOP	A safety stop has been triggered, e.g. by pressing an EMERGENCY STOP.

Status groups whose statuses are set automatically (can also be used by the system integrator):

Status group	Description
ERROR_GENERAL	A general, non-safety-oriented error is active, e.g. application has been terminated with an error.
WARNING_NON_CRITICAL	There is a non-critical warning which does not affect operability of the station.

Status groups whose statuses are only set by the system integrator:

Status group	Description
WARNING_CRITICAL	There is a critical warning which affects operability of the station.
INTERACTION_REQUIRED_HAPTIC	Manual intervention on the robot is required to be able to continue the program, e.g. jogging to a specific position.
INTERACTION_REQUIRED_HMI	Manual intervention on the smartHMI is required (e.g. acknowledging a dialog) to be able to continue the program.
HRC_ACTIVE	The robot is in HRC mode.

18.1.2 Creating status and status groups

Description

In order to be able to define a new status, a status group is necessary. If the new status matches one of the predefined status groups, this status group can be used.

If the new status does not match any of the predefined status groups, new status groups can be created. The designation of the status group must enable clear identification of the group.



If new status groups are created and used, these cannot be processed automatically by the supplied status handlers, e.g. the flexFELLOW status handler. In all cases, user-defined status groups must be handled by the system integrator.

Constructor syntax

New status:

```
Status(StatusGroup statusGroup<, String description>)
```

Explanation of the syntax

Element	Description
<i>statusGroup</i>	Status group for which the new status is being created
<i>description</i>	Description of the new status (optional) The description can be used in status listeners.

Constructor syntax

New status group:

```
StatusGroup(String id)
```

Explanation of the syntax

Element	Description
id	ID of the new status group If several status groups exist with the same ID, these are treated as one status group.

Examples

For the status group WARNING_CRITICAL, a new status lackOfParts is created:

```
Status lackOfParts = new Status(
    DefaultStatusGroups.WARNING_CRITICAL, "palette empty");
```

A new status group and a new status are created here:

```
StatusGroup myStatusGroup =
    new StatusGroup("myStatusGroup");
Status myStatus = new Status(
    myStatusGroup, "myStatus description");
```

18.1.3 Using the IStatusController interface

Description

The IStatusController interface can be used, for example, for requesting all active statuses and status groups or for registering to be notified of status changes. An instance of IStatusController can be integrated into tasks by means of dependency injection.

The methods for requesting statuses and status groups always return all requested statuses and status groups, irrespective of the IStatusMonitor instance on which they were set.

Overview

The IStatusController interface provides the following methods:

Method	Description
getActiveStatusGroups()	Return value type: List<StatusGroup> Returns the list of currently active status groups.
getActiveStatuses()	Return value type: List<Status> Returns the list of currently set statuses.
getActiveStatuses(StatusGroup)	Return value type: List<Status> Returns the list of currently set statuses belonging to the transferred status group.
isSet(Status)	Return value type: Boolean Checks whether the transferred status is set. <ul style="list-style-type: none">• true: Status is set.• false: Status is not set.

Method	Description
addStatusListener(IStatusListener, StatusGroup)	<p>Return value type: void</p> <p>Registers the transferred status listener so that it is notified of status changes.</p> <p>Any number of status groups can be transferred after the parameter IStatusListener. The listener is then only informed of changes in the transferred status groups. If no status groups are transferred, the listener is informed of all status changes.</p>
removeStatusListener(IStatusListener)	<p>Return value type: void</p> <p>Unregisters the transferred status listener so that it is no longer notified of status changes.</p>

18.1.4 Setting and deleting the status via the status monitor

Description

To set and delete a status in a task, a status monitor is required.

Instances of IStatusMonitor can be integrated into a task by means of dependency injection. A separate instance is generated for each task. The lifecycle of the status monitor depends on the task into which it is integrated. If a task is ended, this has the following effect on the status monitor injected for this task:

- All statuses set on the monitor are automatically deleted.
- The monitor can no longer be used.

Instances of IStatusMonitor have separate validity ranges. This means that a status that is set on one instance of IStatusMonitor cannot be deleted on a different instance of IStatusMonitor. This also means that a status which has been set in one task cannot be deleted from another task.

Overview

The interface IStatusMonitor provides the methods for setting and deleting a status:

Method	Description
clear(Status)	<p>Return value type: void</p> <p>Deletes the transferred status.</p> <p>The following exceptions can occur:</p> <ul style="list-style-type: none"> • <code>IllegalArgumentException</code>: Status is <code>null</code> • <code>StatusNotSetException</code>: Status is not currently set. • <code>StatusOutOfScopeException</code>: Status is set, but by a different status monitor.
set(Status)	<p>Return value type: void</p> <p>Sets the transferred status.</p> <p>Any number of statuses from any number of status groups can be set simultaneously on a status monitor.</p> <p>The following exceptions can occur:</p> <ul style="list-style-type: none"> • <code>IllegalArgumentException</code>: Status is <code>null</code> • <code>StatusAlreadySetException</code>: Status is already set. • <code>StatusOutOfScopeException</code>: Status is already set by a different status monitor.



To compare status objects, the status controller internally uses their object identity. If 2 new status objects are generated with identical status groups and description, the status controller handles them as different statuses.

Example

The status `lackOfParts` is set by a status monitor to signal that there is a lack of parts. The status is cleared again when new parts are available.

```

private Status lackOfParts = new Status(
    DefaultStatusGroups.WARNING_CRITICAL, "palette empty");
@.Inject private IStatusMonitor statusMonitor;
//...

@Override
public void run() {
    //...
    //set status if a lack of parts was detected
    statusMonitor.set(lackOfParts);
    //...
    //wait for new parts to be available
    statusMonitor.clear(lackOfParts);
    //...
}

```

18.1.5 Implementing a status listener

Description

In order to be able to respond to status changes in a task, the interface `IStatusListener` must be implemented in a cyclical background application.

The status listener itself must be registered on the status controller using the `addStatusListener(...)` method. Any number of status groups can be transferred during registration of the status listener. The listener is then

only informed of changes in the transferred status groups. If no status groups are transferred, the listener is informed of all status changes. If a status of a subscribed status group is set or deleted, the `onStatusSet(StatusEvent)` or `onStatusCleared(StatusEvent)` method of the status listener is called. The procedure for implementing these methods is illustrated in the example.

Syntax

Registering a status listener:

```
statusController.addStatusListener(statusListener<, statusGroup1,
... statusGroup1+n>);
```

Explanation of the syntax

Element	Description
<code>statusController</code>	Type: <code>IStatusController</code> Status controller on which the listener is registered
<code>statusListener</code>	Type: <code>IStatusListener</code> Listener that is registered
<code>statusGroup1 ... statusGroup1+n</code>	Type: <code>StatusGroup</code> Status groups to which the listener is to respond If no status groups are specified, the listener responds to status changes in all groups.

Overview

The following information is available for `StatusEvent` objects:

Method	Description
<code>getStatusGroup()</code>	Returns the status group to which the set or cleared status belongs.
<code>getDate()</code>	Returns the time at which the status was set or cleared.
<code>hasChangedActiveStatusGroups()</code>	Specifies whether setting or clearing the status has altered the number of active status groups.
<code>getStatusDescription()</code>	Description that was specified when creating the status. Can return <code>null</code> if the set or cleared status has no description.

Example

Implementation of a status listener in a cyclic background application:

```
1 public class BackgroundTask
2   extends RoboticsAPICyclicBackgroundTask
3   implements IStatusListener {
4
5   @Inject
6   private IStatusController statusController;
7   @Inject
8   private ITaskLogger logger;
9
10  @Override
11  public void initialize() {
```

```

12  //initialize your task here
13  initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
14      CycleBehavior.BestEffort);
15
16  //add status listener
17  statusController.addStatusListener(this,
18      DefaultStatusGroups.SAFETY_STOP,
19      DefaultStatusGroups.ERROR_GENERAL);
20 }
21
22 @Override
23 public void runCyclic() {
24
25 }
26
27 @Override
28 public void onStatusSet(StatusEvent statusEvent) {
29     logger.info("Status of group " +
30         statusEvent.getStatusGroup() + " set: " +
31         statusEvent.getStatusDescription());
32 }
33
34 @Override
35 public void onStatusCleared(StatusEvent statusEvent) {
36     logger.info("Status of group " +
37         statusEvent.getStatusGroup() + " cleared: " +
38         statusEvent.getStatusDescription());
39 }
40
41 @Override
42 public void dispose() {
43     //remove status listener
44     statusController.removeStatusListener(this);
45 }
46 }
```

Line	Description
3	The BackgroundTask class implements the IStatusListener interface with implements IStatusListener.
5, 6	A status controller of type IStatusController is integrated into the task by means of dependency injection.
7, 8	A logger object of type ITaskLogger is integrated into the task by means of dependency injection. The logger object can be used to display status information on the smartHMI.
16 ... 19	In the initialize() method, the status listener is registered on the status controller. The keyword this is used to add the BackgroundTask class to itself as a status listener. The status groups SAFETY_STOP and ERROR_GENERAL are transferred during registration. In this way, the listener is only informed of status changes in these status groups.
27 ... 32	The onStatusSet method is called if a status is set with one of the subscribed status groups. The status group and the description of the set status are then logged.

Line	Description
34 ... 39	The onStatusCleared method is called if a status is cleared with one of the subscribed status groups. The status group and the description of the cleared status are then logged.
41 ... 45	In the dispose() method, the status listener is unregistered.

19 Programming with a compliant robot

19.1 Sensors and control

Without additional equipment, a standard industrial robot can only be operated under position control. The aim of position control is to keep the difference between the specified and actual robot position as small as possible at all times.

Apart from position sensors for determining the current joint position, the LBR also has joint torque sensors in every axis, which allow the current joint torques to be measured. These data enable the use of an impedance controller in addition to position control, thus making it possible to implement compliant behavior of the robot. The underlying model is a virtual spring damper system with configurable values for stiffness and damping. Furthermore, additional forces and force oscillations can be overlaid.

The special sensor technology and the available controller mechanisms make the LBR highly sensitive and compliant. This enables it to react very quickly to process forces and makes it particularly suitable for a wide range of joining tasks and for interaction with humans.

19.2 Overview of servo controllers

The robot can be operated with different controllers. For each control type, a separate class is provided by the RoboticsAPI in the package com.kuka.roboticsAPI.motionModel.controlModeModel. The shared superclass is AbstractMotionControlMode.

Controller	Description
Position controller	Data type: PositionControlMode The aim of position control is to execute the specified path with the maximum possible positional accuracy and without path deviation. As standard, external influences such as obstacles or process forces are not taken into account.
Cartesian impedance controller	Data type: CartesianImpedanceControlMode The Cartesian impedance controller is modeled on a virtual spring damper system with configurable values for stiffness and damping. This spring is extended between the setpoint and actual positions of the TCP. This allows the robot to react in a compliant manner to external influences.
Cartesian impedance controller with overlaid force oscillation	Data type: CartesianSinelImpedanceControlMode Special form of the Cartesian impedance controller. In addition to the compliant behavior, constant force setpoints and sinusoidal force oscillations can be overlaid. This controller can be used, for example, to implement force-dependent search runs and vibration motions for joining processes.
Axis-specific impedance controller	Data type: JointImpedanceControlMode The axis-specific impedance controller is modeled on a virtual spring damper. The values for stiffness and damping can be configured for each axis.

19.3 Using controllers in robot applications

Description

In robot applications, the controller to be used is set separately for every motion command. As standard, the following steps are required for this:

Procedure

1. Create the controller object of the desired controller data type.
2. Parameterize the controller object to define the control response.
3. Set the controller as the motion parameter for a motion command.

19.3.1 Creating a controller object

Description

To be able to use a controller, a variable of the desired controller data type must first be created and initialized. As standard, the controller object is generated using the standard constructor.

Syntax

```
ControlMode controlMode;  
controlMode = new ControlMode();
```

Explanation of the syntax

Element	Description
ControlMode	Data type of the controller (subclass of AbstractMotionControlMode)
controlMode	Name of controller object

Example

Creating a Cartesian impedance controller:

```
CartesianImpedanceControlMode cartImpCtrlMode;  
cartImpCtrlMode = new CartesianImpedanceControlMode();
```

19.3.2 Defining controller parameters

The parameters that can be set depend on the type of the controller used. The individual controller classes in the KUKA RoboticsAPI provide specific “set” and “get” methods for each parameter.

(>>> 19.5.2 "Parameterization of the Cartesian impedance controller"
Page 572)

(>>> 19.6.3 "Parameterization of the impedance controller with overlaid
force oscillation" Page 581)

(>>> 19.8 "Axis-specific impedance controller" Page 592)

19.3.3 Transferring the controller object as a motion parameter

Description

The controller object is transferred to a motion as a parameter using the command `setMode(...)`. If no controller object is transferred as a parameter to a motion, the motion is automatically executed with position control.



Motions which use the Cartesian impedance controller must not contain any poses in the proximity of singularity positions.

Syntax

```
object.move (motion.setMode (controlMode) );
```

Explanation of the syntax

Element	Description
<i>motion</i>	Type: Motion Motion to be executed
<i>controlMode</i>	Type: Subclass of AbstractMotionControlMode Name of controller object

19.4 Position controller

With position control, the motors are controlled in such a way that the current position of the robot always matches the setpoint position specified by the controller with just a minimal difference. The position controller is particularly suitable in cases where precise positioning is required.

The position controller is represented by the class `PositionControlMode`. The data type has no configurable parameters for adapting the robot.

If the controller mode of a motion is not explicitly specified, then the position controller is used.

19.5 Cartesian impedance controller

The Cartesian impedance controller is represented by the class `CartesianImpedanceControlMode`.

As standard, the impedance controller refers to the coordinate system with which the motion command is executed.

Examples:

- `robot.move(...);`
The impedance controller refers to the flange coordinate system of the robot.
- `gripper.move(...);`
The impedance controller refers to the standard frame defined for gripper motions.
- `gripper.getFrame("/TipCenter").move(...);`
The impedance controller refers to the tool coordinate system that extends from the "TipCenter" frame on the gripper.

Behavior of the robot

Under impedance control, the robot's behavior is compliant. It is sensitive and can react to external influences such as obstacles or process forces. The application of external forces can cause the robot to leave the planned path.

The underlying model is based on virtual springs and dampers, which are stretched out due to the difference between the currently measured and the specified position of the TCP. The characteristics of the springs are described by stiffness values, and those of the dampers are described by damping values. These parameters can be set individually for every translational and rotational dimension.



If the robot is moved under impedance control, the programmed robot configuration, e.g. the status value, cannot be guaranteed.

19.5.1 Calculation of the forces on the basis of Hooke's law

If the measured and specified robot positions correspond, the virtual springs are slack. As the robot's behavior is compliant, an external force or a motion command results in a deviation between the setpoint and actual positions of the robot. This results in a deflection of the virtual springs, leading to a force in accordance with Hooke's law.

The resultant force F can be calculated on the basis of Hooke's law using the set spring stiffness C and the deflection Δx :

$$F = C \cdot \Delta x$$

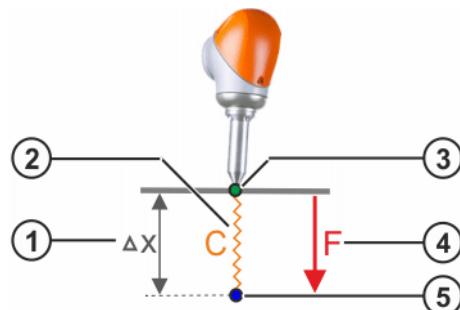


Fig. 19-1: Virtual spring with spring stiffness C

- | | |
|-------------------------|-----------------------|
| 1 Deflection Δx | 4 Resulting force F |
| 2 Virtual spring | 5 Setpoint position |
| 3 Actual position | |

If the robot is at a resistance, it exerts the calculated force. If it is positioned in free space, it moves toward the setpoint position. Due to internal friction forces in the joints, path deviations occur on the way to the setpoint position, whose magnitude depends on the set spring stiffness. Higher stiffness values lead to smaller deviations.

If the robot is already at the setpoint position and an external force is applied to the system, the robot yields to this force until the forces resulting from compliance control cancel out the external forces.

Examples

The force exerted at the contact point depends on the difference between the setpoint position and the actual position and the set stiffness.

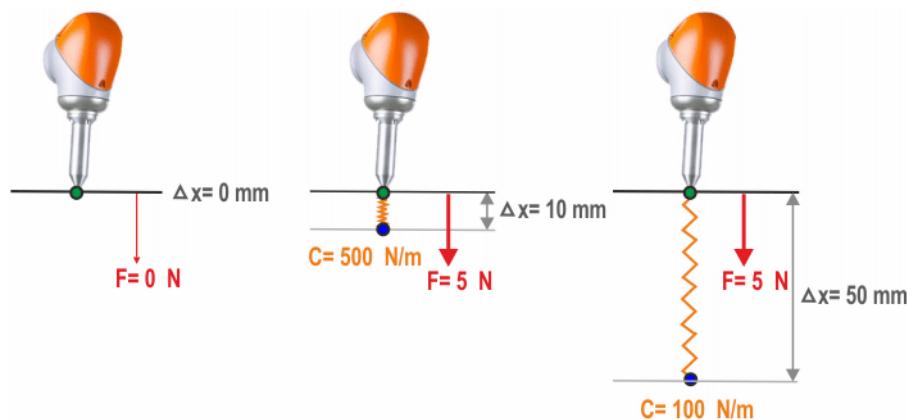


Fig. 19-2: Force exerted on contact

As shown in the figure (>>> [Fig. 19-2](#)), a large position difference and low stiffness can result in the same force as a smaller position difference and greater stiffness. If the force is increased by a motion in a contact situation, the time required to reach this force differs if the Cartesian velocity is identical.

If higher stiffness values are used, a desired force can be reached earlier, as only a small position difference is required. Since the setpoint position is reached quickly, a jerk can be produced in this way.

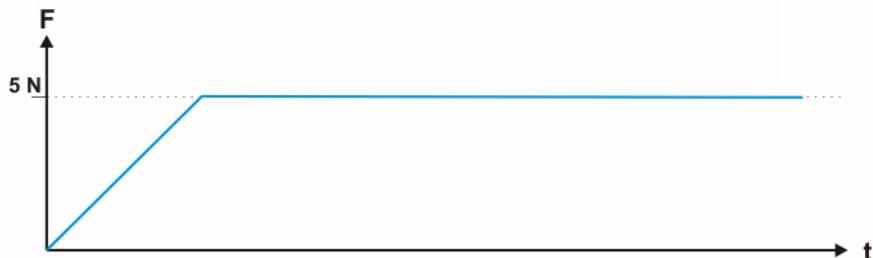


Fig. 19-3: Force over time (high stiffness, small position difference)

In the case of a large position difference and low stiffness, the force is built up more slowly. This can be used, for example, if the robot moves to the contact point and the impact loads are to be reduced.

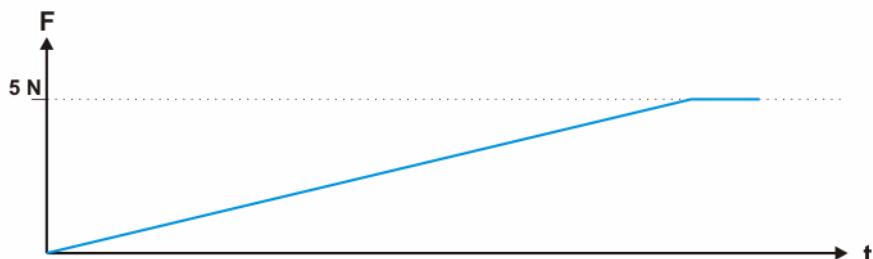


Fig. 19-4: Force over time (low stiffness, large position difference)

Setpoint/actual deviations in more than one direction lead to deflection of all the affected virtual springs. The magnitude and direction of the overall force results from vector addition of the individual forces for each direction.

The deflection in the X direction by Δx and in the Y direction by Δy result in force F_x in the X direction and F_y in the Y direction. The vector addition results in the overall force F_{res} .

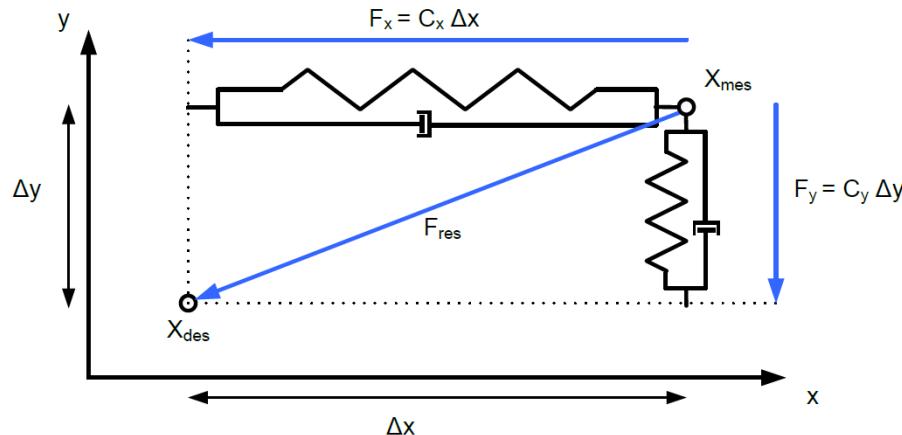


Fig. 19-5: Overall force in the case of deflection in 2 directions

19.5.2 Parameterization of the Cartesian impedance controller

Under impedance control, the robot behaves like a spring. The characteristics of this spring are defined by different parameters. This results in the behavior of the robot.

With a Cartesian impedance controller, forces can be overlaid for all Cartesian degrees of freedom. Forces acting about an axis generate a torque. For this reason, the overlaid torque and not the overlaid force is specified for the rotational degrees of freedom. For the sake of simplification, the terms "force" and "force oscillation" are taken to include the terms "torque" and "torque oscillation" for the rotational degrees of freedom in the following text.



CAUTION

Risk of injury due to unpredictable robot motions in impedance control

In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot. Injuries or damage to property may result.

- Sufficiently restrict the maximum permissible Cartesian deviation from the path and the maximum permissible Cartesian force on the TCP.



CAUTION

The medical product manufacturer must evaluate use of an additional applied part with an energy supply system under impedance control in his risk management process and take the appropriate measures.

The following controller properties can be defined individually for each Cartesian degree of freedom:

- Stiffness
- Damping
- Force to be applied in addition to the spring

The following controller properties can be defined irrespective of the degree of freedom:

- Stiffness of the redundancy degree of freedom
- Damping of the redundancy degree of freedom

- Limitation of the maximum force on the TCP
- Maximum Cartesian velocity
- Maximum Cartesian path deviation

NOTICE

In the case of Cartesian impedance control, conservative parameterization must be carried out. This applies particularly for:

- Maximum Cartesian stiffness values
- Maximum deflections
- Maximum distance between the TCP of the tool and the robot flange

The stiffness characteristics of the null space motion must not be modified. Incorrect parameterization can lead to unexpected robot behavior under impedance control.

The following values must be complied with during parameterization:

Parameter	Description
Damping ratio	Damping for all directions in space: <ul style="list-style-type: none"> • $d = 1$
Stiffness	Limit values for the Cartesian stiffness: <ul style="list-style-type: none"> • Maximum values:<ul style="list-style-type: none"> – Translation (unit: N/m): 1500 – Rotation (unit: Nm/rad): 90 • Minimum values:<ul style="list-style-type: none"> – Translation (unit: N/m): 5 – Rotation (unit: Nm/rad): 10
Spacing	Maximum distance between the TCP of the tool and the robot flange (unit: cm) <ul style="list-style-type: none"> • ≤ 15
Deflection	Maximum deflection in each direction in space (unit: cm) <ul style="list-style-type: none"> • < 10
Force applied	The maximum force applied in each direction in space (unit: N) <ul style="list-style-type: none"> • < 30

19.5.2.1 Representation of Cartesian degrees of freedom

In the RoboticsAPI, the degrees of freedom of the Cartesian impedance controller are represented by the Enum CartDOF (package: com.kuka.roboticsAPI.geometricModel). The values of this Enum can be used to describe either each degree of freedom individually or the combination of a number of degrees of freedom.

Enum value	Description
CartDOF.X	Translational degree of freedom in the X direction
CartDOF.Y	Translational degree of freedom in the Y direction
CartDOF.Z	Translational degree of freedom in the Z direction
CartDOF.TRANSI	Combination of the translational degrees of freedom in the X, Y and Z directions
CartDOF.A	Rotational degree of freedom about the Z axis

Enum value	Description
CartDOF.B	Rotational degree of freedom about the Y axis
CartDOF.C	Rotational degree of freedom about the X axis
CartDOF.ROT	Combination of rotational degrees of freedom about the X, Y and Z axes
CartDOF.ALL	Combination of all Cartesian degrees of freedom

19.5.2.2 Defining controller parameters for individual degrees of freedom

Description

Some parameters of the Cartesian impedance controller can be defined individually for each Cartesian degree of freedom.

During programming, the Cartesian degrees of freedom for which the controller parameter is to apply are specified first. The parametrize(...) method of the controller data types is used for this purpose. To define the degrees of freedom, one or more parameters of the type CartDOF are transferred to this method.

After this, the "set" method of the desired controller parameter is called via the dot operator. This controller parameter is set to the value specified as the input parameter of the set method for all degrees of freedom specified in parametrize(...).

Syntax

```
controlMode.parametrize(CartDOF.degreeOfFreedom_1
<, CartDOF.degreeOfFreedom_2,...) .setParameter(value);
```

Explanation of the syntax

Element	Description
controlMode	Type: CartesianImpedanceControlMode Name of controller object
degreeOfFreedom_1, degreeOfFreedom_2, ...	Type: CartDOF List of degrees of freedom to be described
setParameter(value)	Method for setting a controller parameter A separate method is available for each settable parameter (value = value of the parameter).

Example

A LIN motion is to be executed to a defined point under impedance control. The Cartesian impedance controller is configured in such a way that the currently used TCP – here the robot flange frame – is compliant in the Z direction.

```
CartesianImpedanceControlMode cartImpCtrlMode = new
CartesianImpedanceControlMode();

cartImpCtrlMode.parametrize(CartDOF.X,
CartDOF.Y).setStiffness(3000.0);
cartImpCtrlMode.parametrize(CartDOF.Z).setStiffness(1.0);
cartImpCtrlMode.parametrize(CartDOF.ROT).setStiffness(300.0);
```

```

cartImpCtrlMode.parametrize(CartDOF.ALL).setDamping(0.7);

robot.move(lingetApplicationData().getFrame("/P1")).setCartVelocity(800).setMode(cartImpCtrlMode);

```

19.5.2.3 Controller parameters specific to the degrees of freedom

Overview

The following methods are available for the parameters of the Cartesian impedance controller that are specific to the degrees of freedom:

Method	Description
setStiffness(...)	<p>Spring stiffness (type: double)</p> <p>The spring stiffness determines the extent to which the robot yields to an external force and deviates from its planned path.</p> <p>Translational degrees of freedom (unit: N/m):</p> <ul style="list-style-type: none"> 0.0 ... 5000.0 Default: 2000.0 <p>Rotational degrees of freedom (unit: Nm/rad):</p> <ul style="list-style-type: none"> 0.0 ... 300.0 Default: 200.0 <p>Note: If no spring stiffness is specified for a degree of freedom, the default value is used for this degree of freedom.</p>
setDamping(...)	<p>Spring damping (type: double)</p> <p>The spring damping determines the extent to which the virtual springs oscillate after deflection.</p> <p>For all degrees of freedom (without unit: Lehr's damping ratio):</p> <ul style="list-style-type: none"> 0.1 ... 1.0 Default: 0.7 <p>Note: If no spring damping is specified for a degree of freedom, the default value is used for this degree of freedom.</p>

Method	Description
setAdditionalControlForce(...)	<p>Force applied in addition to the spring (type: double)</p> <p>The additional force results in a Cartesian force at the TCP. This force acts in addition to the forces resulting from the spring stiffness.</p> <p>Translational degrees of freedom (unit: N):</p> <ul style="list-style-type: none"> • Negative and positive values possible Default: 0.0 <p>Rotational degrees of freedom (unit: Nm):</p> <ul style="list-style-type: none"> • Negative and positive values possible Default: 0.0 <p>Note: As standard, the maximum Cartesian force that can be applied is limited. If required, this limit value can be increased with setMaxControlForce(...).</p> <p>Note: If no additional force is specified for a degree of freedom, the default value is used for this degree of freedom.</p> <p>Note: The force is overlaid without a delay. If the force to be overlaid is too great, this can result in overloading of the robot and cancelation of the program. The class <code>CartesianSinelImpedanceControlMode</code> has the option of overlaying forces after a delay.</p>

19.5.2.4 Controller parameters independent of the degrees of freedom

Some settings apply irrespective of the Cartesian degrees of freedom. The set methods used to define these controller parameters belong to the class `CartesianImpedanceControlMode` and are called directly on the controller object.

Overview

The following methods are available for the parameters of the Cartesian impedance controller that are independent of the degrees of freedom:

Method	Description
setNullSpaceStiffness(...)	<p>Spring stiffness of the redundancy degree of freedom (type: double, unit: Nm/rad)</p> <p>The spring stiffness determines the extent to which the robot yields to an external force and deviates from its planned path.</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>Note: If no spring stiffness is specified for the redundancy degree of freedom, a default value is used for this degree of freedom.</p>
setNullSpaceDamping(...)	<p>Spring damping of the redundancy degree of freedom (type: double)</p> <p>The spring damping determines the extent to which the virtual springs oscillate after deflection.</p> <ul style="list-style-type: none"> • 0.3 ... 1.0 <p>Note: If no spring damping is specified for the redundancy degree of freedom, a default value is used for this degree of freedom.</p>

Method	Description
setMaxControlForce(...)	<p>Limitation of the maximum force on the TCP</p> <p>The maximum force applied to the TCP by the virtual springs is limited. The maximum force required to deflect the virtual spring is thus also defined. Whether or not the motion is to be aborted if the maximum force at the TCP is exceeded is also defined.</p> <p>Syntax:</p> <ul style="list-style-type: none"> • <code>setMaxControlForce(maxForceX, maxForceY, maxForceZ, maxTorqueA, maxTorqueB, maxTorqueC, addStopCondition)</code> <p>Explanation of the syntax:</p> <ul style="list-style-type: none"> • <i>maxForceX/Y/Z</i>: Maximum force at the TCP in the corresponding Cartesian direction (type: double, unit: N) <ul style="list-style-type: none"> – ≥ 0.0 Default: Value from the machine data <p>The user-specific maximum value can be requested by the controller object using the <code>getMaxControlForce()</code> method. If no user-specific value has been set, the method returns NULL.</p> • <i>maxTorqueA/B/C</i>: Maximum torque at the TCP in the corresponding rotational direction (type: double, unit: Nm) <ul style="list-style-type: none"> – ≥ 0.0 Default: Value from the machine data <p>The user-specific maximum value can be requested by the controller object using the <code>getMaxControlForce()</code> method. If no user-specific value has been set, the method returns NULL.</p> • <i>addStopCondition</i>: Cancelation of the motion if the maximum force at the TCP is exceeded (type: boolean) <ul style="list-style-type: none"> – true: Motion is aborted. – false: Motion is not aborted. <p>Note: If the force limitation is only to be applied for individual degrees of freedom, correspondingly high values must be assigned to those degrees of freedom that are not to be limited.</p> <p>Note: If no force limitation is defined, the default value from the machine data is used.</p>

Method	Description
setMaxCartesianVelocity(...)	<p>Maximum Cartesian velocity The motion is aborted if the defined velocity limit is exceeded.</p> <p>Syntax:</p> <ul style="list-style-type: none"> • <code>setMaxCartesianVelocity(maxVelocityX, maxVelocityY, maxVelocityZ, maxVelocityA, maxVelocityB, maxVelocityC)</code> <p>Explanation of the syntax:</p> <ul style="list-style-type: none"> • <i>maxVelocityXYZ</i>: Maximum permissible translational velocity at the TCP in the corresponding Cartesian direction (type: double, unit: mm/s) <ul style="list-style-type: none"> – ≥ 0.0 • <i>maxVelocityAIBIC</i>: Maximum permissible rotational velocity at the TCP in the corresponding rotational direction (type: double, unit: rad/s) <ul style="list-style-type: none"> – ≥ 0.0 <p>Note: If the velocity limitation is only to be applied for individual degrees of freedom, correspondingly high values must be assigned to those degrees of freedom that are not to be limited.</p>
setMaxPathDeviation(...)	<p>Maximum Cartesian path deviation Defines the maximum permissible Cartesian path deviation from the currently planned setpoint position for a compliant motion. The motion is aborted if the defined maximum path deviation is exceeded.</p> <p>Syntax:</p> <ul style="list-style-type: none"> • <code>setMaxPathDeviation(maxDeviationX, maxDeviationY, maxDeviationZ, maxDeviationA, maxDeviationB, maxDeviationC)</code> <p>Explanation of the syntax:</p> <ul style="list-style-type: none"> • <i>maxDeviationXYZ</i>: Maximum permissible path deviation at the TCP in the corresponding Cartesian direction (type: double, unit: mm) <ul style="list-style-type: none"> – ≥ 0.0 • <i>maxDeviationAIBIC</i>: Maximum permissible rotational deviation at the TCP in the corresponding rotational direction (type: double, unit: rad/s) <ul style="list-style-type: none"> – ≥ 0.0 <p>Note: If the path deviation is only to be applied for individual degrees of freedom, correspondingly high values must be assigned to those degrees of freedom that are not to be limited.</p>

Example 1

A robot under impedance control is to be compliant in its redundant degree of freedom in order to be able to respond to obstacles during the motion. For this, stiffness and damping of the redundant degree of freedom are parameterized for the impedance controller.

```
CartesianImpedanceControlMode mode =
new CartesianImpedanceControlMode();
mode.setNullSpaceStiffness(10.0);
mode.setNullSpaceDamping(0.7);
```

Example 2

A robot is to move along a table plate in compliant mode. A Cartesian impedance controller is parameterized for this. A high stiffness value is set for the Z direction of the tool coordinate system in the TCP. An additional force of 20 N is also to be applied. The motion is aborted if a force limit of 50 N in the Z direction is exceeded. A low stiffness value is set in the XY plane. The Cartesian deviation in the X and Y directions must not exceed 10 mm, however. Suitable higher values are specified for all other parameters.

```
CartesianImpedanceControlMode mode =
    new CartesianImpedanceControlMode();

mode.parametrize(CartDOF.Z).setStiffness(3000.0);
mode.parametrize(CartDOF.Z).setAdditionalControlForce(20.0);
mode.setMaxControlForce(100.0, 100.0, 50.0, 20.0, 20.0,
20.0, true);

mode.parametrize(CartDOF.X, CartDOF.Y).setStiffness(10.0);
mode.setMaxPathDeviation(10.0, 10.0, 50.0, 2.0, 2.0, 2.0);
```

19.6

Cartesian impedance controller with overlaid force oscillation

The Cartesian impedance controller with overlaid force oscillation is a special form of the Cartesian impedance controller. The force can be overlaid separately for each Cartesian degree of freedom.

Force oscillations about an axis generate torque oscillations. Overlaying torque oscillations can result in the generation of rotational oscillations.

Overlaying constant or sinusoidal forces causes the robot to move. Suitable combinations of oscillations in the individual degrees of freedom can be used to generate different motion patterns.

Using overlaid oscillations, it is possible to implement compliant pendulum motions for search runs and vibrations in the tool for joining processes.

The Cartesian impedance controller with overlaid force oscillation is represented by the class `CartesianSineImpedanceControlMode`.

Behavior of the robot

In this form of impedance control, the overlaid force causes the robot to leave the planned path in a targeted way. The new path is thus determined by a wide range of different parameters.

In addition to stiffness and damping, further parameters can be defined, e.g. frequency and amplitude. The programmed velocity of the robot also plays a significant role for the actual path.



Overlaying additional forces has a strong influence on the robot motion and the forces exerted by the robot. For example, low stiffness and high overlaid forces can cause the robot to accelerate suddenly. Parameterization must therefore be carried out with caution if working with force activations. For example, begin by overlaying low forces and approach the appropriate force values step by step. In addition, the motion resulting from the overlaid force must always be tested in T1 mode first.

19.6.1 Overlaying a simple force oscillation

By overlaying a simple force oscillation, the working point is diverted from the planned path (= path without overlaid oscillations) and is instead moved from the start point to the end point of the motion in a sinusoidal path.

Example

The robot executes a relative motion in the Y direction of the tool coordinate system in the TCP. A sinusoidal force oscillation in the X direction is overlaid. The result is a wave-like path in the XY plane of the coordinate system.

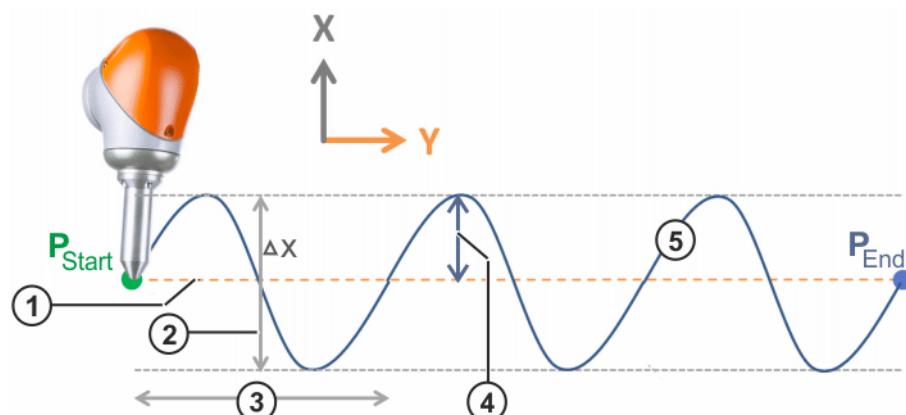


Fig. 19-6: Overlaying a simple force oscillation

- | | |
|-------------------------|-------------|
| 1 Original path | 4 Amplitude |
| 2 Deflection Δx | 5 New path |
| 3 Wavelength | |

The maximum deflection Δx is the deviation from the original path in the positive and negative X directions. The maximum deflection is determined by the stiffness and amplitude which are defined for the impedance controller in the Cartesian X direction, e.g.:

- Cartesian stiffness: $C = 500 \text{ N/m}$
- Amplitude: $F = 5 \text{ N}$

The maximum deflection results from Hooke's law:

$$\Delta x = F / C = 5 \text{ N} / (500 \text{ N/m}) = 1 / (100 \text{ 1/m}) = 1 \text{ cm}$$

The wavelength can be used to determine how many oscillations the robot is to execute between the start point and end point of the motion. The wavelength is determined by the frequency which is defined for the impedance controller with overlaid force oscillation, as well as by the programmed robot velocity.

Wavelength λ is calculated as follows:

$$\lambda = c / f = \text{robot velocity} / \text{frequency}$$

19.6.2 Overlaying superposed force oscillations (Lissajous curves)

Lissajous curves result when a sinusoidal force oscillation is overlaid in 2 different Cartesian directions. The superposition of the two oscillations makes it possible to create very different forms for the path. The exact path depends on a number of parameters.

Application

Two sinusoidal force oscillations of different frequencies can be superposed to generate vibrations at the TCP. For example, such vibrations can remove tension and jamming which occur during an assembly process.

Example

A sinusoidal force oscillation is overlaid in both the X and Y directions of the tool coordinate system in the TCP. The maximum deflections Δx and Δy are determined by the stiffness and amplitude, which are defined for the impedance controller in the Cartesian X and Y directions.

In addition to the known parameters of the impedance controller, the phase offset between the two oscillations plays a significant role in the path.

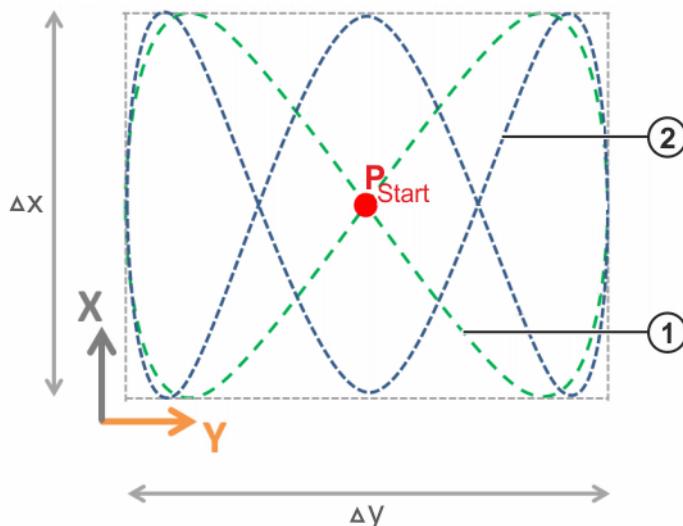


Fig. 19-7: Path of a Lissajous curve

- 1 Path without phase offset (frequency ratio X:Y = 2:1)
- 2 Path with phase offset (frequency ratio X:Y = 3:1)

The form of the path is mainly determined by the ratio of the two frequencies and the phase offset between the two oscillations. The resulting curve is always axisymmetric and point-symmetric. The set power amplitude and stiffness for an oscillation direction results in its position amplitude. The ratio between the two position amplitudes determines the ratio between the width to the height of the curve.

19.6.3 Parameterization of the impedance controller with overlaid force oscillation

The Cartesian impedance controller with overlaid force oscillation is a special form of the standard impedance controller.

With a Cartesian impedance controller with overlaid force oscillation, forces can be overlaid for all Cartesian degrees of freedom. Forces acting about an axis generate a torque. For this reason, the overlaid torque and not the overlaid force is specified for the rotational degrees of freedom. For the sake of simplification, the terms "force" and "force oscillation" are taken to include the terms "torque" and "torque oscillation" for the rotational degrees of freedom in the following text.

**CAUTION****Risk of injury due to unpredictable robot motions in impedance control**

In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot. Injuries or damage to property may result.

- Sufficiently restrict the maximum permissible Cartesian deviation from the path and the maximum permissible Cartesian force on the TCP.

**CAUTION**

The medical product manufacturer must evaluate use of an additional applied part with an energy supply system under impedance control in his risk management process and take the appropriate measures.

The Cartesian impedance controller with overlaid force oscillation is parameterized in the same way as the standard impedance controller. The controller parameters specific to the degrees of freedom and the controller parameters independent of the degrees of freedom as described for the standard impedance controller can be used in the same way for the impedance controller with overlaid force oscillation.

(>>> *19.5.2 "Parameterization of the Cartesian impedance controller"*
Page 572)



Exception: The `setAdditionalControlForce(...)` method of the class `CartesianImpedanceControlMode` for overlaying a force to be applied in addition to the spring is available for the class `CartesianSineImpedanceControlMode`, but should not be used.

The `setBias(...)` method is available for overlaying constant forces in the class `CartesianSineImpedanceControlMode`.

The following additional controller properties can be defined individually for each Cartesian degree of freedom:

- Amplitude of the force oscillation
- Frequency of the force oscillation
- Phase offset of the force oscillation
- Superposed constant force
- Force limitation of the force oscillation
- Limitation of the deflection due to the force oscillation

The following additional controller properties can be defined irrespective of the degree of freedom:

- Rise time of the force oscillation
- Hold time of the force oscillation
- Fall time of the force oscillation
- Overall duration of the force oscillation

19.6.3.1 Controller parameters specific to the degrees of freedom

Overview

The following methods are available for the parameters of the Cartesian impedance controller with overlaid force oscillation that are specific to the degrees of freedom:

Method	Description
setAmplitude(...)	<p>Amplitude of the force oscillation (type: double) Amplitude and stiffness determine the position amplitude. Translational degrees of freedom (unit: N):</p> <ul style="list-style-type: none"> • ≥ 0.0 Default: 0.0 <p>Rotational degrees of freedom (unit: Nm):</p> <ul style="list-style-type: none"> • ≥ 0.0 Default: 0.0 <p>Note: If no amplitude is specified for a degree of freedom, the default value is used for this degree of freedom.</p>
setFrequency(...)	<p>Frequency of the force oscillation (type: double; unit: Hz) Frequency and Cartesian velocity determine the wavelength of the force oscillation.</p> <ul style="list-style-type: none"> • 0.0 ... 15.0 Default: 0.0 <p>Note: If no frequency is specified for a degree of freedom, the default value is used for this degree of freedom.</p>
setPhaseDeg(...)	<p>Phase offset of the force oscillation at the start of the force overlay (type: double; unit: °)</p> <ul style="list-style-type: none"> • ≥ 0.0 Default: 0.0 <p>Note: If no phase offset is specified for a degree of freedom, the default value is used for this degree of freedom.</p>

Method	Description
setBias(...)	<p>Constant force overlaid (type: double)</p> <p>Using setBias(...), a constant force can be overlaid in addition to the overlaid force oscillation. This force adds to the force resulting from the spring stiffness and defined force oscillation.</p> <p>If a constant force is overlaid without an additional force oscillation, this results in a force characteristic which rises as a function of the rise time defined with setRiseTime(...) and then remains constant. setRiseTime(...) belongs to the controller parameters that are independent of the degrees of freedom.</p> <p>(>>> <i>19.6.3.2 "Controller parameters independent of the degrees of freedom" Page 585</i>)</p> <p>If a constant force is overlaid in addition to a force oscillation, the force oscillation is offset in the defined direction.</p> <p>Translational degrees of freedom (unit: N):</p> <ul style="list-style-type: none"> Negative and positive values possible <p>Default: 0.0</p> <p>Rotational degrees of freedom (unit: Nm):</p> <ul style="list-style-type: none"> Negative and positive values possible <p>Default: 0.0</p> <p>Note: As standard, the maximum Cartesian force that can be applied is limited. If required, this limit value can be increased with setMaxControlForce(...).</p> <p>Note: If no additional constant force is overlaid for a degree of freedom, the default value is used for this degree of freedom.</p>
setForceLimit(...)	<p>Force limitation of the force oscillation (type: double)</p> <p>Defines the limit value that the overall force, i.e. the sum of the amplitude of the force oscillation and additionally overlaid constant force, must not exceed. If the overall force exceeds the limit value, the overlaid force is reduced to the limit value.</p> <p>Translational degrees of freedom (unit: N):</p> <ul style="list-style-type: none"> ≥ 0.0 <p>Default: Not limited</p> <p>Rotational degrees of freedom (unit: Nm):</p> <ul style="list-style-type: none"> ≥ 0.0 <p>Default: Not limited</p> <p>Note: If no force limit is specified for a degree of freedom, the default value is used for this degree of freedom.</p>

Method	Description
setPositionLimit(...)	<p>Maximum deflection due to the force oscillation (type: double)</p> <p>If the maximum permissible deflection is exceeded, the force is deactivated. The force is reactivated as soon as the robot is back in the permissible range.</p> <p>Translational degrees of freedom (unit: mm):</p> <ul style="list-style-type: none"> • ≥ 0.0 Default: Not limited <p>Rotational degrees of freedom (unit: rad):</p> <ul style="list-style-type: none"> • ≥ 0.0 Default: Not limited <p>Note: If no maximum deflection is specified for a degree of freedom, the default value is used for this degree of freedom.</p>

Example

During a joining process, an oscillation about the Z axis of the tool coordinate system in the TCP is to be generated. The Cartesian impedance controller with overlaid force oscillation is used for this. With a stiffness of 10 Nm/rad and an amplitude of 15 Nm, the position amplitude is approx. 1.5 rad. The frequency is set to 5 Hz. In order to exert an additional pressing force in the direction of motion, a constant force of 5 N is generated in the Z direction and superposed on the overlaid force oscillation about the Z axis.

```
CartesianSineImpedanceControlMode sineMode =
    new CartesianSineImpedanceControlMode();

sineMode.parametrize(CartDOF.Z).setStiffness(4000.0);
sineMode.parametrize(CartDOF.Z).setBias(5.0);

sineMode.parametrize(CartDOF.A).setStiffness(10.0);
sineMode.parametrize(CartDOF.A).setAmplitude(15.0);
sineMode.parametrize(CartDOF.A).setFrequency(5.0);

tool.getFrame("/TCP").move(linRel(0.0, 0.0, 10.0)
    .setCartVelocity(10.0)
    .setMode(sineMode));
```

19.6.3.2 Controller parameters independent of the degrees of freedom

Some settings apply irrespective of the Cartesian degrees of freedom. The set methods used to define these controller parameters belong to the class `CartesianSineImpedanceControlMode` and are called directly on the controller object.

Overview

The following methods are available for the parameters of the Cartesian impedance controller with overlaid force oscillation that are independent of the degrees of freedom:

Method	Description
setTotalTime(...)	<p>Overall duration of the force oscillation (type: double; unit: s) (>>> "Overall duration of the force oscillation" Page 586)</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>Default: Unlimited</p>
setRiseTime(...)	<p>Rise time of the force oscillation (type: double; unit: s)</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>Default: 0.0</p> <p>Note: If no rise time is specified for a degree of freedom, the default value is used. This means that the amplitude rises abruptly to the defined value without a transition. If the force to be overlaid is too great, this can result in overloading of the robot and cancelation of the program.</p>
setHoldTime(...)	<p>Hold time of the force oscillation (type: double; unit: s)</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>Default: Unlimited</p> <p>Note: If no hold time is specified for a degree of freedom, the default value is used. This means that the overlaid force oscillation ends with the corresponding motion.</p>
setFallTime(...)	<p>Fall time of the force oscillation (type: double; unit: s)</p> <ul style="list-style-type: none"> • ≥ 0.0 <p>Default: 0.0</p> <p>Note: If no fall time is specified for a degree of freedom, the default value is used. This means that the amplitude falls abruptly to zero without a transition. If the drop in force is too great, this can result in overloading of the robot and cancelation of the program.</p>
setStayActiveUntil PatternFinished(...)	<p>Response if the motion duration is exceeded (type: boolean)</p> <p>If the force oscillation lasts longer than the motion, it is possible to define whether the oscillation is terminated or continued after the end of the motion.</p> <ul style="list-style-type: none"> • true: Oscillation is continued after the end of the motion. • false: Oscillation is terminated at the end of the motion. <p>Default: false</p> <p>Note: If the response when the motion duration is exceeded is not specified, the default value is used.</p>

Overall duration of the force oscillation

The overall duration is the sum of the rise time, hold time and fall time of the force oscillation:

- Rise time
Time in which the amplitude of the force oscillation is built up.
- Hold time
Time in which the force oscillation is executed with the defined amplitude.
- Fall time
Time in which the amplitude of the force oscillation is reduced back to zero.

Rise time, hold time and fall time of the force oscillation can be defined individually, or indirectly by defining the overall duration of the force oscillation.

If the overall duration is defined using `setTotalTime(...)`, the rise time and fall time are defined automatically.

Calculation:

- Rise time = fall time = $(1/\text{frequency}) \cdot 0.5$
- Of the frequencies defined for the force oscillation (relative to all degrees of freedom), the frequency that results in the largest possible rise and fall times is used for the calculation.
- If exclusively constant forces are overlaid, the frequency of all degrees of freedom is 0.0 Hz. Rise and fall time are set to 0.0 s.
- If the calculated sum of rise time and fall time exceeds the defined overall duration, the rise time and fall time are each set to 25% of the overall duration and the hold time to 50%.

If the overall duration of the force oscillation is shorter than the duration of the corresponding motion, the force oscillation ends before the end of the motion. The response if the motion duration is exceeded is defined using `setStayActiveUntilPatternFinished(...)`.

19.7 Static methods for impedance controller with superposed force oscillation

Overview

The Cartesian impedance controller with overlaid force oscillation can also be configured via static methods of the class `CartesianSineImpedanceControlMode`. This simplifies the programming, in particular of Lissajous curves, as the user only has to specify a few parameters. The remaining parameters which are important for the implementation are calculated and set automatically. Default values are used for all other parameters. Additional settings are made as described using the `parametrize(...)` function and the set methods of `CartesianSineImpedanceControlMode`.

- `createDesiredForce(...)`: Static method for constant force
- `createSinePattern(...)`: Static method for simple force oscillations
- `createLissajousPattern(...)`: Static method for Lissajous curves
- `createSpiralPattern(...)`: Static method for spirals

Specification of Cartesian planes

In contrast to simple oscillations, no individual degree of freedom is transferred to Lissajous curves and spirals, but rather the plane in which the path is to run. The plane is specified via the `Enum CartPlane` (the package `com.kuka.roboticsAPI.geometricModel`).

Enum value	Description
<code>CartPlane.XY</code>	Path in the XY plane
<code>CartPlane.XZ</code>	Path in the XZ plane
<code>CartPlane.YZ</code>	Path in the YZ plane

19.7.1 Overlaying a constant force

Description

The `createDesiredForce(...)` method overlays a constant force, that does not change over time, in one Cartesian direction.

Syntax

```
controlMode = CartesianSineImpedanceControlMode
.createDesiredForce(CartDOF.degreeOfFreedom, force, stiffness);
```

Explanation of the syntax

Element	Description
<code>controlMode</code>	Type: <code>CartesianSineImpedanceControlMode</code> Name of controller object
<code>degreeOfFreedom</code>	Type: <code>CartDOF</code> Degree of freedom for which the constant force is to be overlaid.
<code>force</code>	Type: <code>double</code> Value of the overlaid constant force. Corresponds to the call of <code>setBias(...)</code> for the specified degree of freedom. Translational degrees of freedom (unit: N): <ul style="list-style-type: none">• ≥ 0.0 Rotational degrees of freedom (unit: Nm): <ul style="list-style-type: none">• ≥ 0.0
<code>stiffness</code>	Type: <code>double</code> Stiffness value for the specified degree of freedom Translational degrees of freedom (unit: N/m): <ul style="list-style-type: none">• 0.0 ... 5000.0 Rotational degrees of freedom (unit: Nm/rad): <ul style="list-style-type: none">• 0.0 ... 300.0

19.7.2 Overlaying a simple force oscillation

Description

The `createSinePattern(...)` method overlays a simple force oscillation in one Cartesian direction.

Syntax

```
controlMode = CartesianSineImpedanceControlMode.createSinePattern(CartDOF.degreeOfFreedom, frequency, amplitude, stiffness);
```

Explanation of the syntax

Element	Description
<i>controlMode</i>	Type: CartesianSineImpedanceControlMode Name of controller object
<i>degreeOf-Freedom</i>	Type: CartDOF Degree of freedom for which the force oscillation is to be overlaid.
<i>frequency</i>	Type: double Frequency of the oscillation (unit: Hz) • 0.0 ... 15.0
<i>amplitude</i>	Type: double Amplitude of the oscillation which is overlaid in the direction of the specified degree of freedom Translational degrees of freedom (unit: N): • ≥ 0.0 Rotational degrees of freedom (unit: Nm): • ≥ 0.0
<i>stiffness</i>	Type: double Stiffness value for the specified degree of freedom Translational degrees of freedom (unit: N/m): • 0.0 ... 5000.0 Rotational degrees of freedom (unit: Nm/rad): • 0.0 ... 300.0

Example

From the current position, a relative motion of 15 cm is to be executed in the Y direction. The motion is to run in a wave path with a deflection of approx. 10 cm (derived from the amplitude and stiffness) and a frequency of 2 Hz in the X direction.

```
CartesianSineImpedanceControlMode sineMode;
sineMode =
CartesianSineImpedanceControlMode.createSinePattern(CartDOF.X,
, 2.0, 50.0, 500.0);

robot.move(linRel(0.0, 150.0,
0.0).setCartVelocity(100).setMode(sineMode));
```

19.7.3 Overlaying a Lissajous oscillation

Description

The createLissajousPattern(...) method is used to generate a 2-dimensional oscillation in one plane. The plane is transferred as a value of type CartPlane. The other transferred parameters refer to the first degree of freedom of the specified plane (example: for CartPlane.XY, the specified values are relative to CartDOF.X).

The parameters of the second degree of freedom of the plane are calculated to produce a Lissajous curve with the following characteristics:

- Amplitude ratio, 1st degree of freedom : 2nd degree of freedom: 1 : 1
- Frequency ratio, 1st degree of freedom : 2nd degree of freedom: 1 : 0.4
- Phase offset between 1st and 2nd degree of freedom: $\frac{1}{2} \cdot \pi$

Syntax

```
controlMode = CartesianSineImpedanceControlMode.createLissajousPattern(CartPlane.plane, frequency, amplitude, stiffness);
```

Explanation of the syntax

Element	Description
<i>controlMode</i>	Type: <code>CartesianSineImpedanceControlMode</code> Name of controller object
<i>plane</i>	Type: <code>Enum</code> of type <code>CartPlane</code> Plane in which the Lissajous oscillation is to be overlaid
<i>frequency</i>	Type: <code>double</code> Frequency of the oscillation for the first degree of freedom of the specified plane (unit: Hz) <ul style="list-style-type: none"> • 0.0 ... 15.0 The frequency for the second degree of freedom is calculated as follows: <ul style="list-style-type: none"> • $frequency \cdot 0.4$
<i>amplitude</i>	Type: <code>double</code> Amplitude of the oscillation for both degrees of freedom of the specified plane (unit: N) <ul style="list-style-type: none"> • ≥ 0.0
<i>stiffness</i>	Type: <code>double</code> Stiffness values for both degrees of freedom of the specified plane (unit: N/m) <ul style="list-style-type: none"> • 0.0 ... 5000.0

Example

An oscillation in the form of a Lissajous curve with a frequency ratio X : Y of 1 : 0.4 and a phase offset in Y of $\pi/2$ is to be generated on the robot flange. Path without phase offset (= blue line (>>> *Fig. 19-7*)).

```
CartesianSineImpedanceControlMode lissajousMode;  
  
lissajousMode =  
CartesianSineImpedanceControlMode.createLissajousPattern(Cart  
Plane.XY, 10.0, 50.0, 500.0);  
  
robot.move(linRel(0.0, 150.0,  
0.0).setCartVelocity(100).setMode(lissajousMode));
```

19.7.4 Overlaying a spiral-shaped force oscillation

Description

The `createSpiralPattern(...)` method is used to generate a spiral-shaped force oscillation in one plane.

The force characteristic is created by overlaying 2 sinusoidal force oscillations. The oscillations are shifted in phase by $\pi/2$ (90°). The amplitudes of the oscillations rise constantly up to the defined value and then return to zero. This results in a spiral pattern which extends up to the defined amplitude value and then contracts again.

In the resulting robot motion, the TCP moves along this spiral. The Cartesian extent of the spiral depends on the values defined for stiffness and amplitude as well as any obstacles present.

The plane in which the spiral-shaped oscillation is to be overlaid is transferred as a value of type `CartPlane`. The values defined for the parameters stiffness, frequency and amplitude are identical for both degrees of freedom of the plane.

In addition, a value is transferred for the total time of the force oscillation. The time is divided evenly between the upward and downward motion of the oscillation:

Rise time = Total time / 2

Hold time = 0

Fall time = Total time / 2

Syntax

```
controlMode = CartesianSineImpedanceControlMode.createSpiralPattern(CartPlane.plane, frequency, amplitude, stiffness, totalTime);
```

Explanation of the syntax

Element	Description
<code>controlMode</code>	Type: <code>CartesianSineImpedanceControlMode</code> Name of controller object
<code>plane</code>	Type: <code>Enum</code> of type <code>CartPlane</code> Plane in which the spiral-shaped oscillation is to be overlaid
<code>frequency</code>	Type: <code>double</code> Frequency of the oscillation for both degrees of freedom of the specified plane (unit: N) • 0.0 ... 15.0
<code>amplitude</code>	Type: <code>double</code> Amplitude of the oscillation for both degrees of freedom of the specified plane (unit: N) • ≥ 0.0
<code>stiffness</code>	Type: <code>double</code> Stiffness values for both degrees of freedom of the specified plane (unit: N/m) • 0.0 ... 5000.0

Element	Description
<i>totalTime</i>	Type: double Total time for the spiral-shaped oscillation. The time is divided evenly between the upward and downward motion of the oscillation (unit: s). • ≥ 0.0

Example

At the current position of the robot flange, a spiral-shaped force oscillation is to be overlaid in the XY plane of the flange coordinate system. The force is to rise helically up to a maximum value of 100 N. Once per second, the force characteristic is to turn around the start point of the spiral (frequency of the force oscillation: 1.0 Hz). The force spiral must rise and fall within 10 seconds.

```
CartesianSineImpedanceControlMode spiralMode;
spiralMode =
CartesianSineImpedanceControlMode.createSpiralPattern(CartPla
ne.XY, 1.0, 100, 500, 10);
robot.move(positionHold(spiralMode, 10, TimeUnit.SECONDS));
```

The number of turns is a function of the total time for a turn (t_{period}). The time for a turn corresponds to the duration of an oscillation period, e.g.:

- Frequency of the force oscillation: $f = 1.0 \text{ Hz}$
- Total time: $t = 10 \text{ s}$

The number of turns is calculated as follows:

$$\text{NumberTurns} = \text{Total time} / t_{\text{Period}} = 10 \text{ s} / 1 \text{ s} = 10$$

$$t_{\text{Period}} = 1 / f = 1 / 1.0 \text{ Hz} = 1 \text{ s}$$

The maximum deflection results from Hooke's law:

$$\Delta x = F / C = 100 \text{ N} / (500 \text{ N/m}) = 0.2 \text{ m} = 20 \text{ cm}$$

19.8 Axis-specific impedance controller

The axis-specific impedance controller is represented by the class `JointImpedanceControlMode`. In this control mode, the robot's behavior is compliant.

The underlying model uses virtual springs and dampers. Unlike with the Cartesian impedance controller, however, these springs and dampers are stretched out due to the difference between the currently measured and the specified axis positions. For this reason, singularity positions of the robot have no influence on the impedance behavior.

19.8.1 Parameterization of the axis-specific impedance controller



CAUTION

Risk of injury due to unpredictable robot motions in impedance control

In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot. Injuries or damage to property may result.

- Sufficiently restrict the maximum permissible Cartesian deviation from the path and the maximum permissible Cartesian force on the TCP.



If the application is paused with the spring tensioned under impedance control, the motion command is interrupted. When the application is resumed, the spring is tensioned again. This can result in jerky motion of the robot.

The following controller properties can be defined individually for each axis:

- Stiffness
- Damping

19.8.2 Methods of the axis-specific impedance controller

Overview

The following set methods are available for the axis-specific impedance controller:

Method	Description
setStiffness(...)	<p>Spring stiffness (Type: double[]; unit: Nm/rad)</p> <p>The axis-specific spring stiffness determines the degree of compliance of an axis when force is applied.</p> <ul style="list-style-type: none"> • ≥ 0.0 Default: 1000 <p>Note: The spring stiffness must be specified for every axis.</p>
setDamping(...)	<p>Spring damping (type: double[]; without unit: Lehr's damping ratio)</p> <p>The axis-specific spring damping determines the extent to which the virtual springs oscillate after deflection.</p> <ul style="list-style-type: none"> • 0.0 ... 1.0 Default: 0.7 <p>Note: The spring damping must be specified for every axis.</p>
setStiffness ForAllJoints(...)	<p>Spring stiffness (type: double, unit: Nm/rad)</p> <p>A value determines the degree of compliance of all axes when force is applied.</p> <ul style="list-style-type: none"> • ≥ 0.0

Method	Description
setDamping ForAllJoints(...)	<p>Spring damping (type: double; without unit: Lehr's damping ratio) A value determines the extent to which the virtual springs in all axes oscillate after deflection.</p> <ul style="list-style-type: none"> • 0.0 ... 1.0

Constructor syntax

```
JointImpedanceControlMode jointImp =
new JointImpedanceControlMode(stiffnessJ1 ... stiffnessJ7);
```

Explanation of the syntax

Element	Description
<i>jointImp</i>	Type: JointImpedanceControlMode Name of controller object
<i>stiffnessJ1</i> ... <i>stiffnessJ7</i>	Type: double; unit: Nm/rad Axis-specific spring stiffnesses The number of values is dependent on the axis selection (here: 7 axes).

Example 1

7 axes are to be controlled using the axis-specific impedance controller. Initial values for the axis-specific spring stiffnesses are defined in the constructor of the controller. The stiffness for axis A4 is to be modified subsequently. The spring damping is to be identical for all axes.

```
JointImpedanceControlMode jointImp =
new JointImpedanceControlMode(2000.0, 2000.0, 2000.0,
    2000.0, 100.0, 100.0, 100.0);
//...
jointImp.setStiffness(2000.0, 2000.0, 2000.0, 1500.0,
    100.0, 100.0, 100.0);
jointImp.setDampingForAllJoints(0.5);
```

Example 2

7 axes are to be controlled using the axis-specific impedance controller. Initial values for the axis-specific spring stiffnesses are defined in the constructor of the controller. The spring stiffness and spring damping are subsequently to be identical for all axes.

```
JointImpedanceControlMode jointImp =
new JointImpedanceControlMode(2000.0, 2000.0, 2000.0,
    2000.0, 100.0, 100.0, 100.0);
//...
jointImp.setStiffnessForAllJoints(100);
jointImp.setDampingForAllJoints(0.5);
```

19.9 Holding the position under servo control

Description

Using the motion command `positionHold(...)`, the robot can hold its Cartesian setpoint position over a set period of time and remain under servo control.

If the robot is operated in compliance control, it can remove itself from its setpoint position. Whether, how far and in which direction the robot moves from the current Cartesian setpoint position (= position at the start of the command `positionHold(...)`) depends on the set controller parameters and the resulting forces. In addition, the compliant robot under servo control can be forced off its setpoint position by external forces.

Syntax

```
object.move(positionHold(controlMode, time, unit));
```

Explanation of the syntax

Element	Description
<code>controlMode</code>	Type: subclass of <code>AbstractMotionControlMode</code> Name of controller object
<code>time</code>	Type: <code>long</code> Indicates how long the specified <code>controlMode</code> is to be held. The value must be ≥ 0 . A value of < 0 indicates infinite. The time unit is defined with <code>unit</code> .
<code>unit</code>	Type: enum of type <code>TimeUnit</code> Unit of the specified time The Enum <code>TimeUnit</code> is an integral part of the standard Java library.

Example

The robot is to be held in its current position for 10 seconds. During this time, the robot is switched to “soft” mode in the Cartesian X direction.

```
CartesianImpedanceControlMode controlMode = new
CartesianImpedanceControlMode();

controlMode.parametrize(CartDOF.X).setStiffness(1000.0);
controlMode.parametrize(CartDOF.ALL).setDamping(0.7);

robot.move(positionHold(controlMode, 10, TimeUnit.SECONDS));
```


20 Diagnosis

20.1 Field bus diagnosis



WorkVisual can be used for precise error analysis. Additional information about field bus diagnosis with WorkVisual is contained in the WorkVisual documentation.



If the robot controller is used as a PROFINET master or device, hardware problems can result in an inability to access bus devices. In this case, use of a diagnostic tool, such as WorkVisual, Step 7 or Wireshark, is recommended.

20.1.1 Displaying general field bus errors

Description

The general error state of the connected field buses can be displayed on the smartHMI.

Procedure

1. Select the **KUKA_Sunrise_Cabinet_1** tile at the Station level.
The status indicator of the **Field buses** tile indicates the collective state of all field buses connected to the controller.
2. Select the **Field buses** tile.
The detail view opens with error information about the currently connected field buses.

20.1.2 Displaying the error state of I/Os and I/O groups

Description

The status indicator in the **I/O groups** area of the navigation bar of the smartHMI displays the state of the configured I/O groups.

- The lower indicator shows the collective state of all configured I/O groups.
- The upper indicator shows the state of the selected I/O group.

Procedure

- In the navigation bar, select the desired I/O group from **I/O groups**.
The detail view of the I/O group opens. Any faulty inputs/outputs are indicated.



If PROFINET is used and errors occur at individual terminals, not only the affected inputs/outputs, but all configured PROFINET I/Os are marked as faulty.

20.2 Displaying a log

Description

A log of the events and changes in state of the system can be displayed on the smartHMI.

Procedure

1. Open the Station level or the Robot level.
2. Select the **Log** tile. The **Log** view opens.

If the view is opened via the Robot level, only those log entries are displayed as standard which affect the robot selected in the navigation bar.

20.2.1 “Log” view

Overview

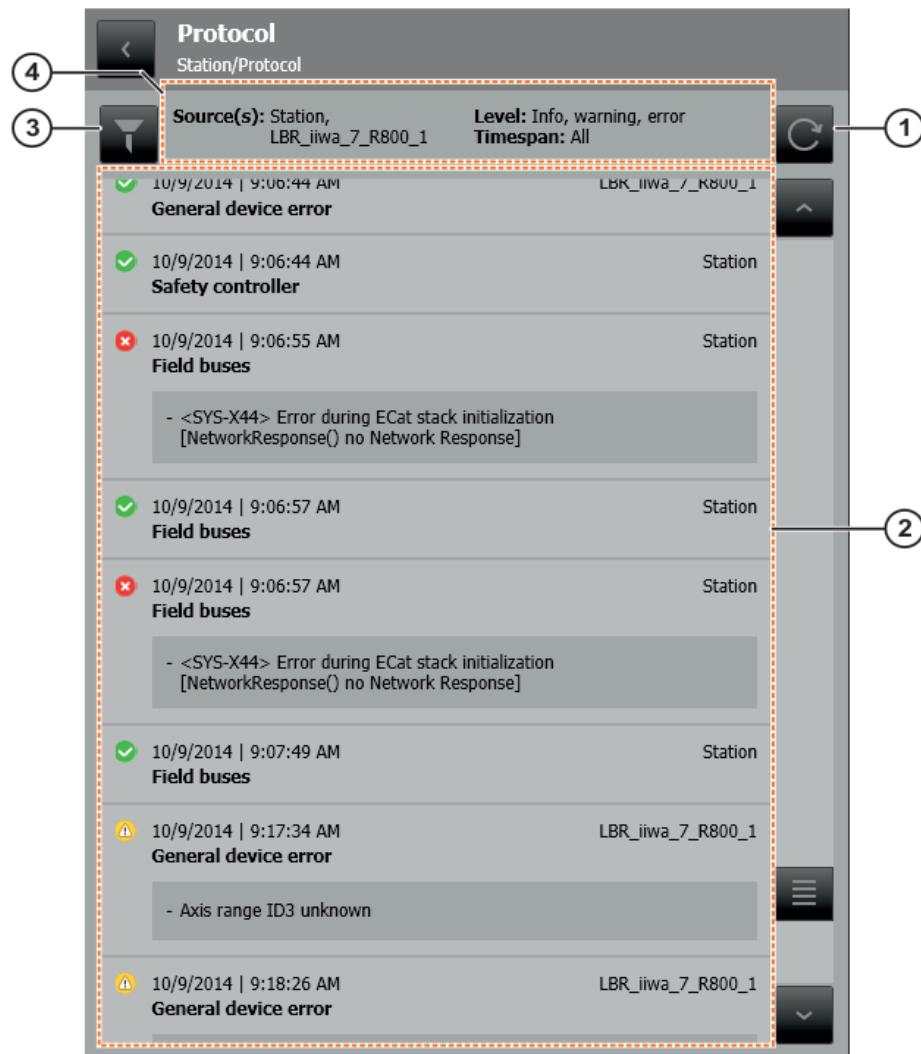


Fig. 20-1: View of log

Item	Description
1	Refresh button Refreshes the displayed log entries. As standard, the most recent entry is shown at the top of the list after refreshing. If a time filter is active, the oldest entry is shown at the top of the list.
2	List of log entries (>>> <i>"Log event" Page 599</i>)

Item	Description
3	Filter settings button Opens the Filter settings window in which the log entries can be filtered according to various criteria.
4	Filter settings display The currently active filters are displayed here.

Log event

The log entries contain various information pertaining to each log event.

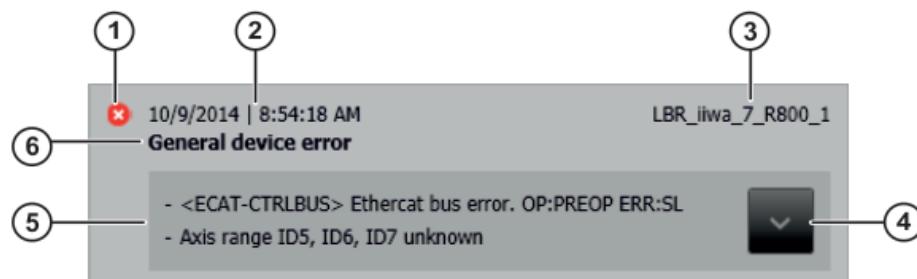


Fig. 20-2: Information about the log event

Item	Description
1	Log level of the event (>>> "Log level" Page 599)
2	Date and time of the log event (system time of the robot controller)
3	Source of the log event (robot or station)
4	Button to maximize/minimize the detail view The button is only available if more than 2 symptoms are present.
5	Symptoms of the log event (detail view) As standard, up to 2 symptoms are displayed per event.
6	Category or brief description of the log event

Log level

The following icons display the log level of an event:

Icon	Description
	Error Critical event which results in a system error state
	Warning Critical event which can result in an error
	Information Non-critical event or information pertaining to the change in state

20.2.2 Filtering log entries

Precondition

- The **Log** view is open.

Procedure

- Touch the **Filter settings** button. The **Filter settings** window opens.
- Select the desired filters with the appropriate buttons.
- Touch the **Filter settings** button or an area outside the window.
The **Filter settings** window is closed and the selected filters are activated.



The filters are reset when the **Log** view is closed. When the view is re-opened, the default settings are reactivated.

Description

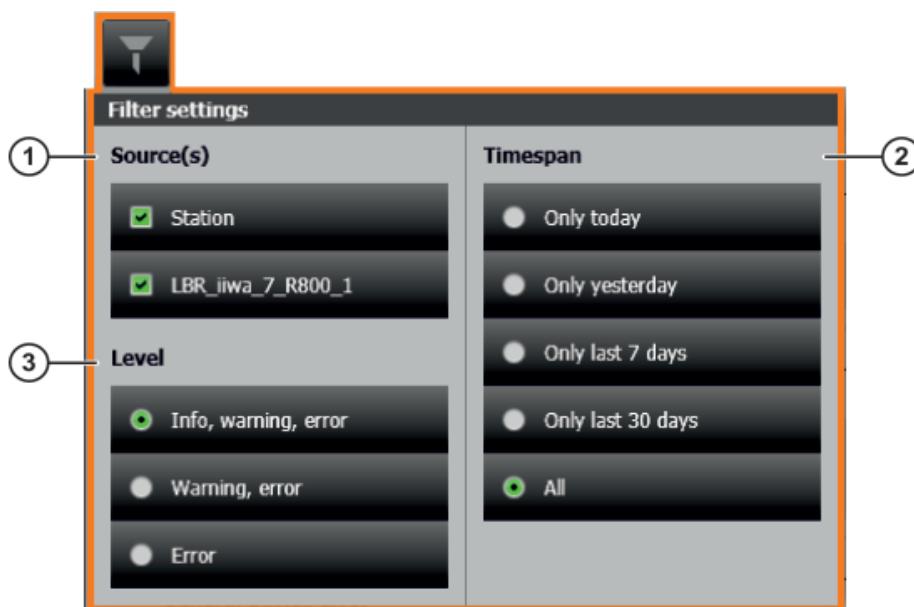


Fig. 20-3: “Filter settings” window

Item	Description
1	<p>Filter Source(s)</p> <p>The log entries can be filtered according to the sources that caused the log event.</p> <ul style="list-style-type: none">Station: All log entries are displayed which affect the station and the inputs/outputs of field buses.Robot: Only those log entries are displayed which affect the robot selected in the navigation bar. <p>Default for log at Station level: All sources are selected.</p> <p>Default for log at Robot level: The source is the robot selected in the navigation bar.</p>
2	<p>Filter Timespan</p> <p>A time filter can be activated to display only the log entries of a specific timespan.</p> <p>Default: All (no time filter active)</p>

Item	Description
3	Filter Level The log entries can be filtered according to their log level. Default: Info, warning, error (no filter active for log level)

20.3 Displaying error messages

If errors occur while a robot application is being executed, the corresponding error messages are displayed on the smartHMI (application view). Errors during execution of background tasks are not displayed there.

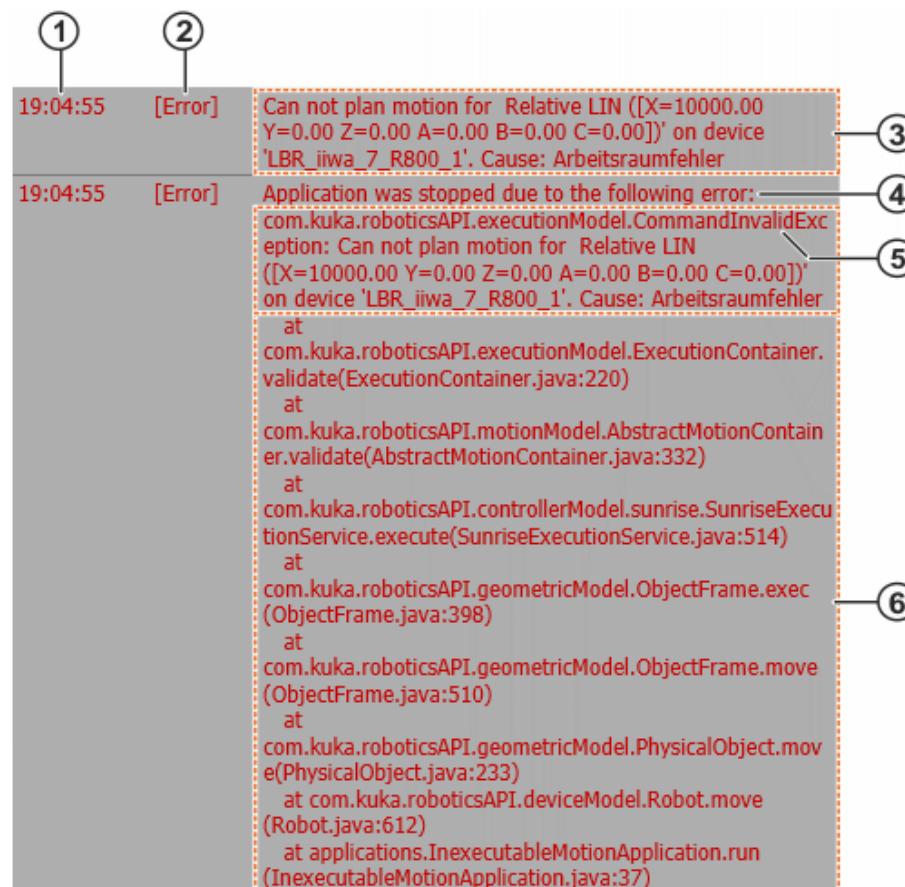
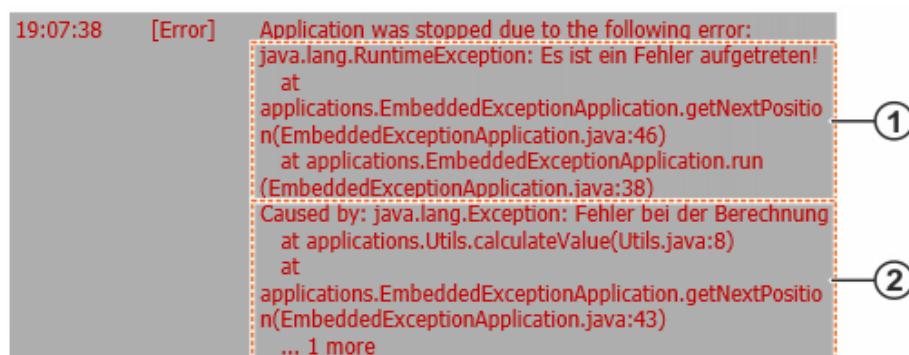


Fig. 20-4: Structure of error message (example)

Item	Description
1	Time stamp Time at which the error occurred
2	Level Log level of the message Errors have the log level Error .
3	Error message
4	Information when application is terminated, e.g. following a real-time error

Item	Description
5	Error type Errors are defined as Java classes. The name of the class and the corresponding package are displayed. The error message follows (see item 3).
6	Stack trace The method calls which led to the error are displayed in ascending order. The methods are specified with their full identifiers. In addition, the number of the program line in which the error occurred is displayed. The stack trace can be used to determine the program position at which the method which ultimately caused the error was called. Example, read from the bottom to the top: <ul style="list-style-type: none"> • Origin of the error: run() method of the application InexecutableMotion.java, line 37 • In line 37 of the application, the move(...) method of the robot class was called. In the source code of the Robot.java class, the error occurred in line 612 when the move(...) method of the PhysicalObject class was called. • ... • The actual error occurred in line 220 in the source code of the ExecutionContainer.java class when the validate(...) method was called.

Often, an error is the result of a chain of preceding errors. In this case, the entire error chain is displayed in descending order.



The screenshot shows a terminal window with the following output:

```
19:07:38 [Error] Application was stopped due to the following error:  
java.lang.RuntimeException: Es ist ein Fehler aufgetreten!  
at applications.EmbeddedExceptionApplication.getNextPosition(EmbeddedExceptionApplication.java:46)  
at applications.EmbeddedExceptionApplication.run(EmbeddedExceptionApplication.java:38)  
Caused by: java.lang.Exception: Fehler bei der Berechnung  
at applications.Utils.calculateValue(Utils.java:8)  
at applications.EmbeddedExceptionApplication.getNextPosition(EmbeddedExceptionApplication.java:43)  
... 1 more
```

Two annotations are present: a red dashed box labeled ① encloses the main error message and the first few lines of the stack trace; a red dashed box labeled ② encloses the 'Caused by' section and its associated stack trace frame.

Fig. 20-5: Display of error chain (example)

Item	Description
1	Consequential error The last element in the error chain is displayed here. In the example, this is an error of type RuntimeException which occurred during execution of the run() method in line 38 of the application EmbeddedExceptionApplication.java.
2	Causative error The display of the causative error is always initiated as follows: <ul style="list-style-type: none"> • Caused by: <i>Error type</i> In the example, the causative error is of type Exception and occurred when the calculateValue(...) method of the Utils class was called. The entire error chain is thus displayed up to the actual cause of error.

20.4 Displaying messages of the virus scanner

Precondition

- Virus scanner is installed.

Procedure

- Select > **KUKA_Sunrise_Cabinet_1** > **Virus scanner** at the Station level. The **Virus scanner** view opens.



Messages from the virus scanner can also be displayed using the **Log** tile.

Description

The **Virus scanner** view contains the following data:

- **Virus scanner state:** active / inactive
- **Version of virus definition file:** Version of the virus definition file
- Messages about detected viruses

The message generated when a virus is found contains the following data:

- Name of the virus
- Name of the file in which the virus is located, including path specification
- Date and time of detection

If viruses are found, the status display of the **Virus scanner** tile switches to "Warning". The status of the files affected by the viruses is automatically set to "Quarantine".



If the robot can no longer be moved due to a virus infection, the following options are available:

- Reinstall the System Software on the robot controller.
- If the robot can still not be moved, create the diagnosis package **KRCDiag** and contact KUKA Service.

20.5 Collecting diagnostic information for error analysis at KUKA

For error analysis, KUKA Customer Support requires diagnostic data from the robot controller.

For this purpose, a ZIP file called **KRCDiag** is created, which can be archived on the robot controller under D:\DiagnosisPackages or on a USB stick connected to the robot controller. The diagnosis package **KRCDiag** contains the data which KUKA Customer Support requires to analyze an error. These include information about the system resources, machine data and much more.

Sunrise.Workbench can also be used to access the diagnostic information. For this purpose, either an existing diagnosis package is loaded from the robot controller or a new package is created.



Projects and applications are not included in the diagnosis package. It is advisable to transfer these data separately, as they can contain important information for troubleshooting.



Recommendation: If possible, only collect diagnostic information when the robot is stationary.



If the collection of diagnostic information fails while an application is running, stop and cancel the application and restart the diagnostic process.



All relevant LOG files for the **KRCDiag** diagnosis package can be loaded from the controller. Direct access to individual files on the robot controller is not allowed.

20.5.1 Creating a diagnosis package with the smartHMI

Description

With this procedure, the diagnosis package **KRCDiag** can be created and archived on the robot controller under D:\DiagnosisPackages or on a USB stick.

Procedure

1. For archiving to a USB stick: Plug the USB stick into the robot controller and wait until the LED on the USB stick remains permanently lit.
2. In the main menu, select **Diagnosis > Create diagnosis package** and select the desired file location.
 - **Hard drive**
 - **USB stick**

The diagnostic information is compiled. 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.

20.5.2 Creating a diagnosis package with the smartPAD

Description

This procedure uses keys on the smartPAD instead of menu items. It can thus also be used if the smartHMI is not available.

The **KRCDiag** diagnosis package is created and archived on the robot controller under D:\DiagnosisPackages.



The key sequence described in the procedure must be executed within 2 seconds.

Procedure

1. Press the “Main menu” key and hold it down.
2. Press the keypad key twice.
3. Release the “Main menu” key.

The diagnostic information is compiled. 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.

20.5.3 Creating a diagnosis package with Sunrise.Workbench

Precondition

- Network connection to the robot controller

Procedure

1. Right-click on the project in the **Package Explorer** and select **Sunrise > Create diagnosis package** from the context menu. The wizard for creating the diagnosis package opens.
2. Select **Browse...** and navigate to the directory in which the diagnosis package **KRCdiag** is to be created. If necessary, create a folder for the diagnosis package by clicking on **Create new folder**. Confirm with **OK**.
3. Click on **Next >**. The diagnosis package is created in the specified folder.
4. To navigate to the folder in which the diagnosis package was created, e.g. to send it directly by e-mail, click on **Open target folder in Windows Explorer**.
5. Click on **Finish**. The wizard is closed.



Projects and applications are not included in the diagnosis package. It is advisable to transfer these data separately, as they can contain important information for troubleshooting.

20.5.4 Loading existing diagnosis packages from the robot controller

Precondition

- Network connection to the robot controller

Procedure

1. Right-click on the project in the **Package Explorer** and select **Sunrise > Create diagnosis package** in the context menu. The wizard for creating the diagnosis package opens.
2. Select **Browse...** and navigate to the directory in which the diagnosis package **KRCdiag** is to be copied. If necessary, create a folder for the diagnosis package by clicking on **Create new folder**. Confirm with **OK**.
3. Activate the radio button **Load existing diagnosis packages from controller** and select the desired diagnosis packages.
4. Click on **Next >**. The diagnosis packages are copied into the specified folder.

If the folder already contains a diagnosis package of the same name, a user dialog is displayed. The copying operation can be canceled.

5. To navigate to the folder into which the diagnosis packages were copied, e.g. to send them directly by e-mail, click on **Open target folder in Windows Explorer**.
6. Click on **Finish**. The wizard is closed.

21 Remote debugging

Remote debugging is used for the discovery and diagnosis of errors in programs.

Remote debugging is carried out using Sunrise.Workbench for applications and background tasks running on the controller.

Since remote debugging is largely identical for applications and background tasks, the term "task" is used generically below.



Advanced Java and Eclipse skills are a prerequisite for use of remote debugging.

21.1 Debugging session sequence

Step	Description
1	<p>Starting a debugging session</p> <p>When starting a debugging session, a remote connection is established between Sunrise.Workbench and the robot controller. The project in the workspace of Sunrise.Workbench and the active project on the robot controller are automatically checked for consistency and synchronization is requested if required.</p> <p>(>>> 21.1.2 "Starting the debugging session" Page 609)</p>
2	<p>Performing remote debugging of the task</p> <p>The programmer uses break points to define the positions in the program code at which execution of the task is to be interrupted during remote debugging.</p> <p>If remote debugging is to be carried out for an application that has not yet been started, the application must be started manually via the smartPAD once the remote connection has been established.</p> <p>Once task execution has been stopped at a break point, further program execution can be controlled by Sunrise.Workbench by executing the source code of the task step by step. On completion of a step, task execution is automatically stopped.</p> <p>(>>> 21.3.2 "Break points" Page 614)</p>
3	<p>Using debugging functions</p> <p>While task execution is interrupted, debugger functions, such as the observation and modification of variable values, can be used. Adaptation of the source code is also possible.</p> <p>(>>> 21.3.6 "Variables view" Page 629)</p> <p>(>>> 21.3.7 "Monitoring processes" Page 633)</p> <p>(>>> 21.3.8 "Modifying source code" Page 636)</p>

Step	Description
4	<p>Ending a debugging session</p> <p>When ending a debugging session, the remote connection to the controller is disconnected. Execution of the running task can now no longer be influenced by Sunrise.Workbench. If modifications have been made to the code, project synchronization is offered.</p> <p>If, when the remote debugging connection is terminated, the task is currently paused at a break point, the program is automatically resumed after this termination.</p> <p>(>>> 21.1.3 "Ending the debugging session" Page 610)</p>

21.1.1 Remote debugging of tasks

Description

Remote debugging is used to detect and diagnose errors in programs and tasks.

Tools that support this process are called debuggers. The remote debugger integrated into Sunrise.Workbench is based on the standard Java and Eclipse debugger.

Eclipse enables the debugging of applications that are executed in a different Java runtime environment or on different physical machines. In the case of remote debugging of tasks, the Sunrise.Workbench debugger is used; the task itself is executed on the controller.

During remote debugging, a connection is established between Sunrise.Workbench and the robot controller. A debugging session is started in this way. During remote debugging, the execution of tasks running on the controller can be monitored via Sunrise.Workbench and it is possible to influence program execution. Errors can be diagnosed and the source code can be optimized.

The **Debugging** perspective contains the most important views for remote debugging.



While a debugging session is running, the smartPAD is used for starting applications and issuing the motion enable signal.



All safety functions configured for the project are also active during remote debugging.

Remote debugging of tasks running on the controller is described below. General knowledge of remote debugging is assumed. The most important fundamentals are summarized in the chapters. (>>> [21.3 "Fundamentals of remote debugging" Page 613](#))

Overview

The functionalities offered by the debugger include the following:

- Performance of online diagnosis
- Stopping program execution at defined positions using break points
- Line-by-line or section-by-section execution of the source code of running tasks
- Tracking of the process by means of observation of variables and monitoring expressions
- Modification of variable values

- Setting of break points if definable exceptions occur during program execution

21.1.2 Starting the debugging session

Description

To start a debugging session, a remote connection must be established between Sunrise.Workbench and the robot controller.

Once the first active break point has been reached, execution of the corresponding task is paused and the functions of the debugger can be used.

(>>> [21.3.2 "Break points" Page 614](#))

Precondition

- Network connection between robot controller and development computer
- The project that is active on the robot controller is located in the workspace of Sunrise.Workbench.

Procedure

1. Select the desired project in the **Project Explorer**.
2. Click on the **Debug project** button.
The system scans the robot controller for existing project data. If the scan fails, the cause of the error is displayed in a message.
3. If the scan is successful, the **Project synchronization** window opens. Select the desired synchronization direction.



There must be no project on the controller at the time of synchronization, or the existing project must differ from the local project. If the projects are identical, the synchronization dialog is not displayed and the remote debugging session is started immediately.

4. Click on **Execute**.
5. The system signals that the remote connection to the robot controller has been established successfully. The remote connection is established with **OK** and the debugging session is started.
6. When the first active break point is reached, the corresponding task is paused.

If the **Debugging** perspective is not active, the dialog **Confirm change of perspective** is displayed in Sunrise.Workbench. It is recommended that the dialog is ended with **Yes** to switch to the **Debugging** perspective of Sunrise.Workbench.



In the case of a change of perspective, the decision in the dialog box can be saved with **Remember my decision**.

(>>> [21.3.1 "Overview of user interface – "Debugging" perspective" Page 613](#))

Once the task has been paused, its source code is opened in the editor area. The current position of the command pointer is indicated by the fact that the next command line to be executed is selected.

(>>> [21.3.3 "Command pointer" Page 620](#))

21.1.3 Ending the debugging session

Description

In order to end the debugging session correctly, the remote connection between Sunrise.Workbench and the robot controller must be disconnected. If modifications have been made to the code during remote debugging, synchronization of the project is offered.

Precondition

- Network connection between robot controller and development computer
- The project that is active on the controller is located in the workspace of Sunrise.Workbench.

Procedure

1. Click on **Stop debug mode**.

During an active debugging session, this replaces the **Debug project** button.

2. If modifications were made to the source code of the task during the debugging session, the **Retain project changes** dialog opens:

- **Synchronize project** is used to synchronize the project and transfer the changes to the controller.
- With **Cancel**, the changes are only saved permanently for the project in the workspace of Sunrise.Workbench. On the controller, the changes are deleted after switching off and back on.

NOTICE

It is advisable to use the **Synchronize project** option, as the system response may otherwise be different after a reboot. If the reboot is not carried out immediately, these changes in behavior may be unexpected and could result in damage to the machine.

NOTICE

If execution of a task is paused when the connection is disconnected, task execution is resumed automatically immediately after disconnection. This also applies to the running application. In Automatic mode, and in the Test modes if the enabling switch and Start button are pressed, the robot may move. It is thus advisable to disconnect the remote connection only if the application has been terminated, or to pause motion execution first by pressing the Start button on the smartPAD.

21.2 Debugging tasks



Remote debugging influences the time response of tasks. The time response may therefore deviate from the real time response during normal execution of the task.

NOTICE

Irrespective of the mode, the system response may change. Debugging of a task in Automatic mode must be carried out with particular care. Interventions that result in a change of state must be tested in Manual Reduced Velocity mode (T1).

Interventions and commands that cause a change of state include:

- Modification of program execution
- Execution of additional commands
- Motion commands
- Modification of variables
- Modification of the source code during debugging
- Setting inputs/outputs
- Changes of values, e.g. by calling set methods

This can lead to deviations from the program execution of the task. The deviations may have an effect beyond the duration of the debugging session. Unexpected robot motions are also possible.

Overview

Debugging can be performed for all tasks running on the controller. In order to debug an application, it may be necessary to start the application via the smartPAD.



The smartPAD is also required during the debugging of tasks, e.g.:

- Issuing motion enable signal
- Starting applications
- Pausing motion commands
- Executing motion commands in the test modes
- Stopping an application prematurely

As soon as the first active break point is reached after the remote connection has been established, execution of the corresponding task can be controlled via Sunrise.Workbench. Various functions are available for this. The selected function determines the command line up to which the task is continued.

If task execution is paused during debugging, additional functions are available and changes can be made to the source code:

- Available functions:
(>>> [21.3.5 "Overview of the toolbar in the "Debugging" view" Page 623](#))
- Additional functions of the debugger:
(>>> [21.3.6 "Variables view" Page 629](#))
(>>> [21.3.7 "Monitoring processes" Page 633](#))
- Information about modification of the source code during debugging:
(>>> [21.3.8 "Modifying source code" Page 636](#))

21.2.1 Remote debugging of a robot application**Description**

A possible procedure for debugging a robot application is described below.



A running application can be ended via the smartPAD during debugging. In this case, all break points are temporarily deactivated to ensure that the application is terminated correctly.



If the application for which debugging is being carried out does not contain any active break points, it is executed completely without stopping and then terminated.

Precondition

- Debugging session started
- Application started in accordance with the selected operating mode

Procedure

1. On reaching an active break point, the application is stopped by the debugger. Program execution can now be influenced by Sunrise.Workbench.
2. At the break point, pressing the corresponding button in the toolbar of the **Debugging** view or using the corresponding keyboard shortcut defines the step at which the application is to be resumed.
3. The application is resumed until the command line defined by selecting the function is reached. If a code section to be executed contains motion commands, this has a special effect on the sequence.
4. In order to continue the application on reaching a synchronous motion command or to execute an asynchronous motion command, the following actions must additionally be carried out on the smartPAD in accordance with the operating mode:
 - T1, T2:
 - Press and hold down the enabling switch.
 - Press and hold down the Start key.
 - AUT:

In AUT mode, no additional operator action is required. The motion is executed immediately.
5. Once the program section has been executed, the application is stopped.
Exception: With **Resume**, the application is continued until the next break point or the end of the application is reached.
6. Debugging functions, such as the observation of variables or the changing of values, can be used between the individual steps.

21.2.2 Remote debugging of a background task

Description

Debugging can also be carried out for background tasks. If a background task contains active break points, execution of the background task is stopped at these points during a debugging session.

Debugging of background tasks is essentially carried out in the same way as debugging of applications. Manual background tasks must be started manually by the user. Furthermore, background tasks should not contain motion commands. Debugging of background tasks is thus not affected by the selected operating mode.

Precondition

- Existing remote connection
- All background tasks are running automatically on the controller.
Non-automatic background tasks must be started manually by the user.

Procedure

1. On reaching an active break point, the background task is stopped by the debugger. Program execution can now be influenced by Sunrise.Workbench.
 2. At the break point, pressing the corresponding button in the toolbar of the **Debugging** view or using the corresponding keyboard shortcut defines the step at which the application is to be resumed.
 3. The background task is resumed until the command line defined by selecting the function is reached.
 4. Once the program section has been executed, the background task is stopped.
- Exception: With **Resume**, the background task is continued until the next break point or the end of a non-cyclic background task is reached.
5. Debugging functions, such as the observation of variables or the changing of values, can be used between the individual steps.

21.3 Fundamentals of remote debugging

21.3.1 Overview of user interface – “Debugging” perspective

The **Debugging** perspective contains a suitable arrangement of views that are useful for remote debugging.

A configuration of the **Debugging** perspective is displayed below. The **Debug** perspective can be expanded to include additional views.

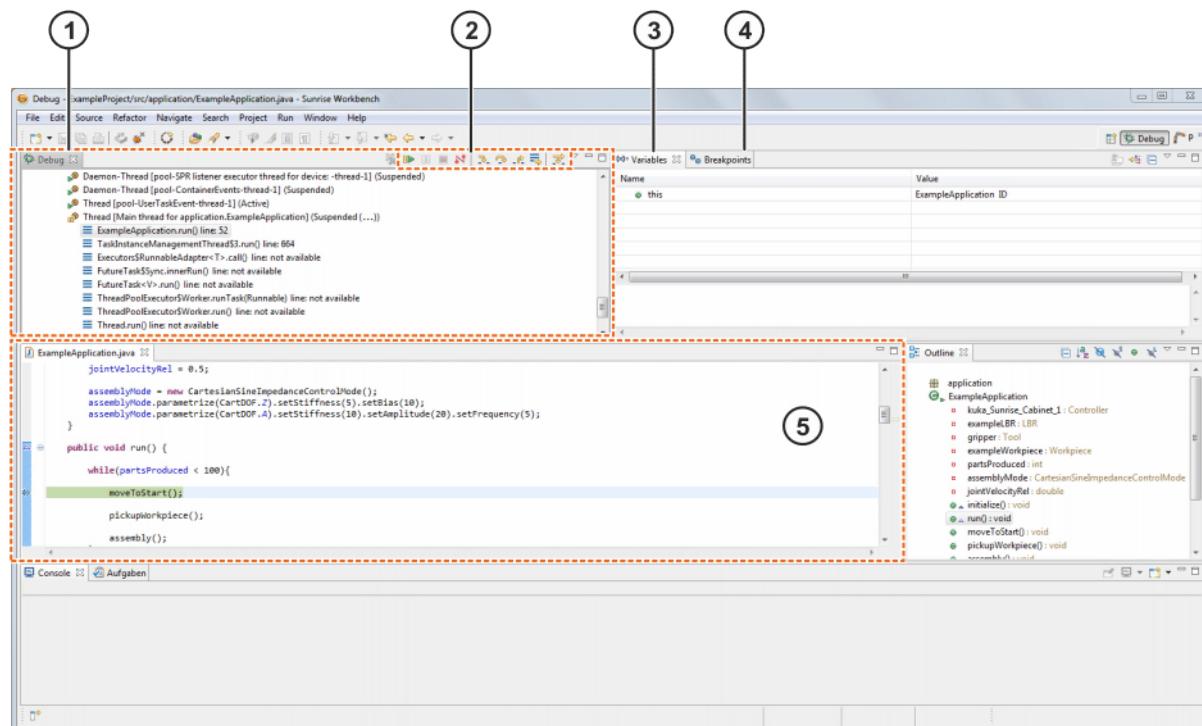


Fig. 21-1: Overview of user interface – “Debugging” perspective

Item	Description
1	Debug view Displays threads of the virtual machine connected for debugging.
2	Debugging toolbar Program execution during remote debugging is controlled by means of the buttons. The toolbar can be displayed using the drop-down arrows or the “Show Debug Toolbar” button.
3	Variables view The variables are displayed and managed here. The variables refer to the currently selected context (line/method) in the “Debug” view (item 1). Modification of values is possible.
4	Breakpoints view Break points are displayed and managed here.
5	Editor area During remote debugging, the source code currently being executed can be displayed here. If task execution is paused, the current command line is highlighted. Modifications to the source code are generally possible and are applied in the running task. An exception is changes to the schema that cannot be applied in the running task.

21.3.2 Break points

Overview

The use of break points is a major component of remote debugging. The programmer uses break points in the source code to define specific points in the program at which the program is to be stopped during remote debugging.

Break points are created and managed in Sunrise.Workbench. Break points only pause the task during a debugging session. It is not taken into consideration during normal program execution.

The creation, deletion, activation and deactivation of break points and the modification of their properties are possible before and during remote debugging.

Depending on the position in the code at which the break point is used, a distinction is made between different types of break point.

- Line break point

The line break point is the most commonly used break point. The line break point is placed next to a command line. Program execution is stopped when the break point is reached. The command line next to it is not executed until remote debugging is resumed.

- Monitoring point

A monitoring point is placed next to the declaration of a field. Program execution is stopped before read and/or write access to the field.

- Method break point

A method break point is placed next to the header of a method. Program execution before the method is entered and/or left.

- Exception break point

An exception break point stops program execution when an error occurs. Exception break points are displayed and created in the **Break points** view.

In order to define more precisely the response on reaching the break point, certain properties can be parameterized for each break point. Different settings are possible, depending on the type of break point.

21.3.2.1 Creating and deleting break points

Description

Break points are created in the editor area of Sunrise.Workbench.

```

1 ExampleApplication.java
2
3 package application;
4
5 import javax.inject.Inject;
6
7
8 * Implementation of a robot application.
9 public class ExampleApplication extends RoboticsAPIApplication {
10     @Inject
11     private LBR robot;
12
13     @Override
14     public void initialize() {
15         // initialize your application here
16     }
17
18     @Override
19     public void run() {
20         // your application execution starts here
21         robot.move(ptpHome());
22         robot.move(ptp(getFrame("/Start")));
23         mainTask();
24         robot.move(ptp(getFrame("/End")));
25     }
26
27     private void mainTask() {
28         robot.move(ptp(getFrame("/P1")));
29         // ...
30         robot.move(ptp(getFrame("/P10")));
31     }
32
33 }

```

Fig. 21-2: Creating break points

Item	Description
1	<p>Editor bar</p> <p>Break points are displayed next to the corresponding command line in the bar with a gray background at the left-hand edge of the editor. Break points can be added to the editor bar, deleted, activated or deactivated.</p>
2	<p>Monitoring point (in this case for the array "robot")</p> <ul style="list-style-type: none"> • Break point inserted next to the declaration of an array • Indicated by means of a pair of glasses and/or a pencil
3	<p>Line break point (in this case for the command <code>robot.move(ptpHome());</code>)</p> <ul style="list-style-type: none"> • Break point inserted next to a command line • Indicated by a blue circle

Item	Description
4	<p>Method break point (here: mainTask() method)</p> <ul style="list-style-type: none"> • Break point inserted next to the header of a method • Indicated by a blue circle with arrow

Procedure

1. Open the class in the editor area of Sunrise.Workbench.
2. Search for the command line next to which a break point is to be set.
3. Double-click next to the desired command line in the editor bar. A new break point is inserted and indicated by the corresponding icon on the bar.
4. To remove a break point for a command line, double-click on the corresponding icon.



Break points can also be added and removed by right-clicking on the corresponding point of the editor bar. Select the entry **Break point on/off** in the context menu that appears.

21.3.2.2 Deactivating and activating break points

Description

If a break point is not to be deleted completely, but merely ignored temporarily during remote debugging, deactivation of the break point is possible. It remains available with all its properties and can be reactivated again if required.

Procedure

1. Open the class in the editor area.
2. Search for the command line containing the break point that is to be deactivated.
3. Right-click on the icon of the break point and select **Deactivate break point** from the context menu. The break point is deactivated. The icon for the break point is grayed out.
4. To activate a deactivated break point, right-click on the icon of the break point and select **Activate break point** from the context menu. The break point is active again.

21.3.2.3 Editing the properties of the break points

Description

The properties of a break point define the conditions for stopping a task when the break point is reached. The settings are dependent on the type of break point.

Procedure

1. Open the class in the editor area.
2. Search for the command line containing the break point whose properties are to be edited.
3. Right-click on the icon of the break point and select **Breakpoint properties** from the context menu. The **Properties** dialog opens.
4. Select **Breakpoint properties**. Edit the properties of the break point.

5. Confirm with **OK**. The dialog is closed.

21.3.2.4 “Break points” view

The view contains a list of the break points of all classes in the workspace of Sunrise.Workbench. The view offers the following functions:

- Display of all break points
- Activation, deactivation and deletion of break points
- Modification of break point properties
- Addition of exception break points

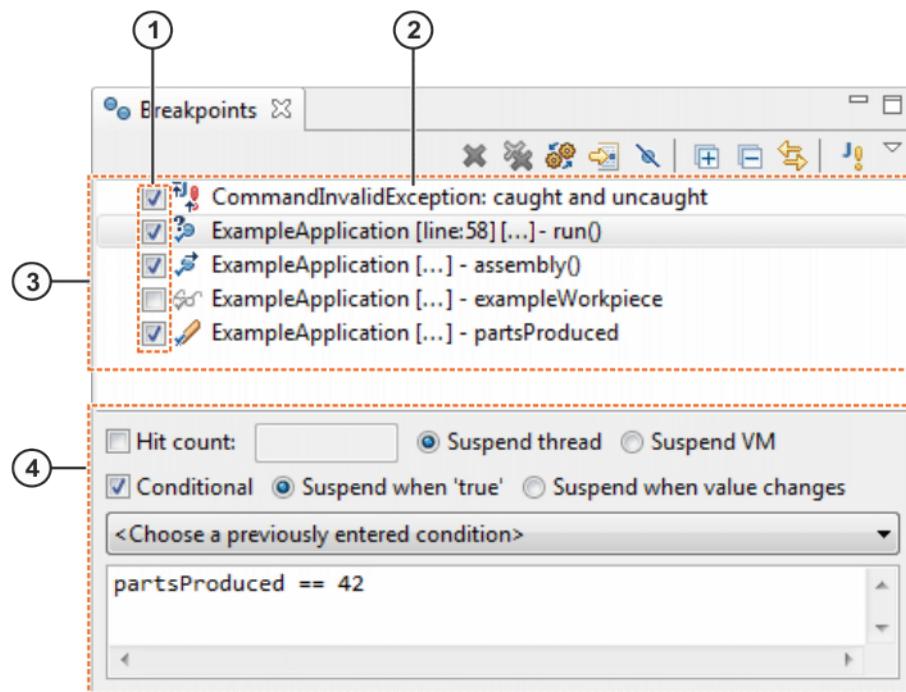


Fig. 21-3: “Break points” view

Overview

Item	Description
1	Activation of the break point Check box active: break point is activated. Check box not active: break point is not activated.
2	Designation of the break point The designation is composed of special properties of the break point in order to enable unambiguous identification.
3	Break point list List of the break points of all classes in the workspace
4	Break point properties The properties of the break point selected in the list can be displayed and edited in this area.

The functionalities offered by the buttons in the toolbar include the following:

Button	Description
	Remove selected break points Deletes the break points selected in the break point list.
	Remove all break points Deletes all break points in the list.
	Go to file for break point The class containing the break point selected in the list is opened in the editor area in the foreground and the corresponding command line is selected.
	Skip all break points If this button is active, all break points are deactivated and do not cause the execution of the corresponding task to be stopped.
	Add break point for Java exception condition Opens the dialog for adding an exception break point.

21.3.2.5 Conditional break point

Description

For break points, a condition can be formulated as an additional property. Such a conditional break point only causes the corresponding task to stop if a condition defined by the user is met when the break point is reached.

Conditions can be defined for the following break points:

- Line break point
- Method break point

In the case of a conditional break point, a question mark is added next to the icon for the break point in the editor bar.

NOTICE

The condition is evaluated every time the break point is reached. This influences the time response of the task.

It is recommended not to use state-changing commands in the condition. Interventions and commands that cause a change of state include:

- Motion commands
- Modification of variables
- Modification of the source code
- Setting inputs/outputs
- Changes of values, e.g. by calling set methods

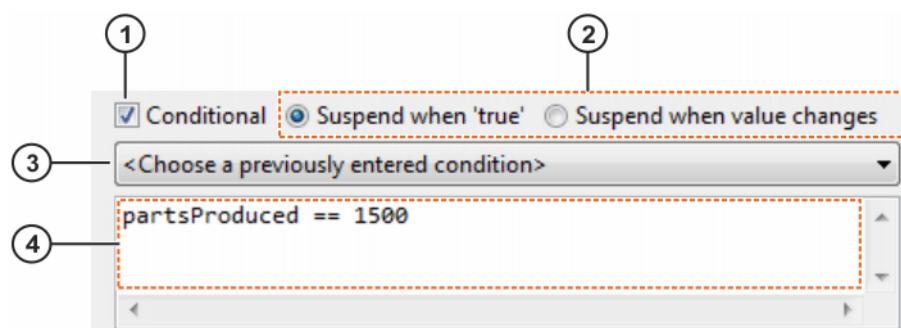


Fig. 21-4: Setting the properties

Overview

Item	Description
1	<p>Check box for activation of the condition</p> <ul style="list-style-type: none"> Check box not active: The condition is not active. The corresponding task is stopped every time the break point is reached. Check box active: The condition is active. Depending on the result of the evaluated condition, the corresponding task is stopped when the break point is reached.
2	<p>Selection of the event</p> <p>Defines the event that causes the corresponding task to be stopped.</p> <ul style="list-style-type: none"> Suspend when 'true' The task is stopped exactly on reaching the break point if the defined condition is met (return value TRUE). Suspend when value changes The task is stopped exactly on reaching the break point if the state of the condition has changed since the last time the break point was reached (change of state from condition met to condition not met or vice versa).
3	<p>List of recently entered conditions</p> <p>If a condition is selected from the list, it is entered in the editor box and the previous contents of the box are deleted.</p>
4	<p>Editor box</p> <p>The condition is entered in the editor box.</p> <p>Certain rules must be observed. Simple Boolean expressions can be entered, e.g.:</p> <ul style="list-style-type: none"> input == FALSE counter <= 510 <p>Complex Java instructions can also be formulated. The commands are then executed every time the break point is reached. A Boolean value must be returned at the end of the sequence of instructions in order to enable evaluation of the condition. Correct syntax must be observed.</p> <p>Variables and commands that are also available at the position of the break point in the source code of the task or class can be used when formulating the conditions.</p> <p>Evaluation of the conditions results in a significant change in the time response. It is advisable to limit the number and duration of the commands used to an absolute minimum, as the conditions are evaluated every time the break point is reached.</p>

Example

An application is to be interrupted before the start of a joining operation if the applied force is insufficient. In this example, the applied force is considered to be insufficient if the force acting on the Z axis of the flange is less than 5 N.

The following break point properties are set for this:

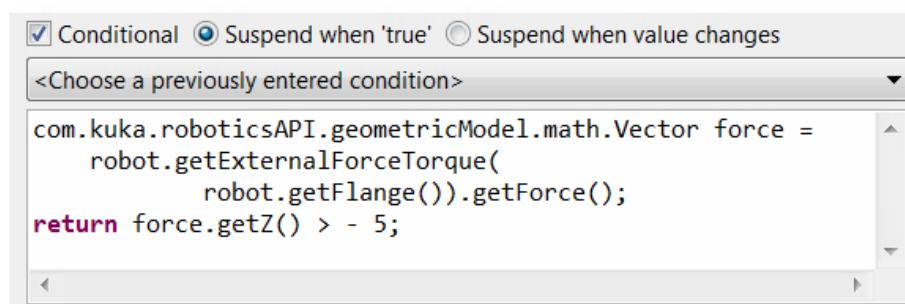


Fig. 21-5: Conditional break point

The application is only stopped if the result of evaluation of the condition is TRUE. The condition consists of a sequence of instructions that are executed every time the break point is reached. First of all, the calculated force at the robot flange is requested. The system then checks whether the Z component of the force vector falls below -5 Nm. The result of the evaluation is returned.

21.3.2.6 Suspend thread property

Description

The selection of **Suspend thread** in the properties of a break point must not be changed.

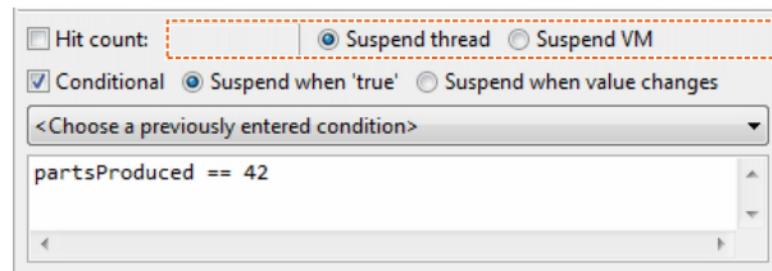
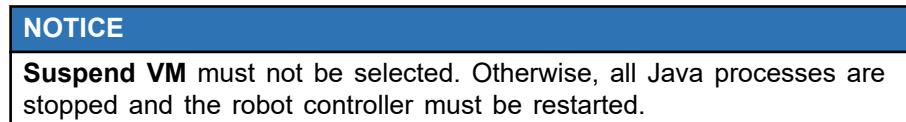


Fig. 21-6: Defining processes

21.3.3 Command pointer

During debugging, program execution is controlled manually.

The command pointer indicates the current position in the source code and the next command to be executed. During program execution, the command pointer jumps from one command line to the next.

During remote debugging, the command pointer is moved through the source code of the task, and the classes used by it, by Sunrise.Workbench. The command lines that the command pointer moves past are executed.

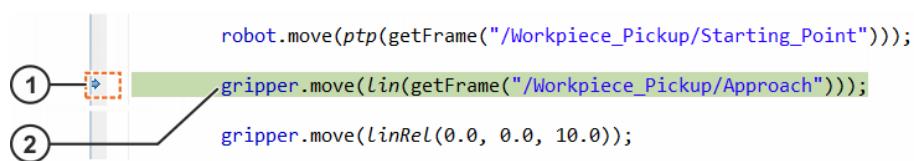


Fig. 21-7: Command pointer

Item	Description
1	Position of the command pointer (blue arrow) The command pointer indicates the next command to be executed. The current position of the command pointer in the source code is indicated by a blue arrow.
2	Next command line to be executed In the paused thread, the currently selected row in the stack trace.

21.3.4 “Debugging” view

Description

The **Debug** view contains the toolbar and a list of all Java threads running on the controller. The task for which debugging is carried out is one of the threads running on the controller.

In the **Debug** view, the corresponding stack trace is displayed beneath a thread. The stack trace contains the current method calls of a thread and is used for tracking program execution.

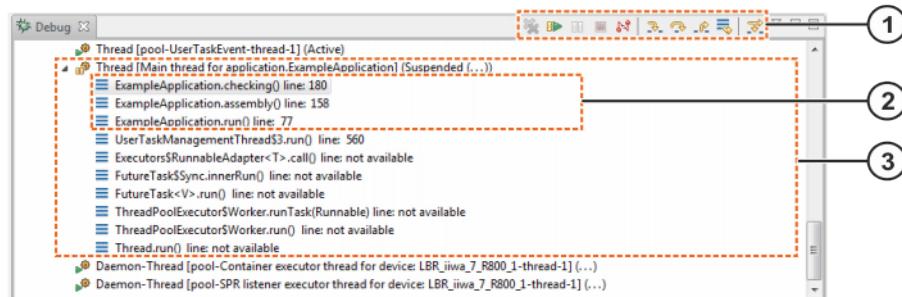


Fig. 21-8: “Debugging” view

Item	Description
1	Toolbar Program execution during remote debugging is controlled by means of the buttons.
2	Task thread Task-relevant part of stack trace of the application thread. The designation contains the name of the executed tasks (here application.ExampleApplication). The corresponding stack trace is located beneath the thread.

Item	Description
3	<p>Stack trace</p> <p>The stack trace of the task thread contains the methods that are relevant for execution of the task. The called methods are specified with their identifiers. In this way, the user can identify the relevant methods.</p> <p>The methods are specified in the order in which they are called.</p>

Example

In a robot application, the `assembly()` method is called in the `run()` method in order to assemble a component. The `assembly()` method then calls the `checking()` method to check whether the assembly process has been successfully completed:

```
public void run() {
    // ...
    assembly();
    // ...
}

public void assembly() {
    // ...
    boolean result = checking();
    // ...
}

public boolean checking() {
    // ...
    return result;
}
```

Execution is interrupted at a break point in the `checking()` method. The command pointer is located before the next command line to be executed. The `checking()` method is selected in the stack trace of the task thread:

The screenshot shows the KUKA Sunrise Workbench interface with a stack trace for a task thread. The stack trace is as follows:

- Thread [Main thread for application.ExampleApplication]
- ExampleApplication.checking() line: 170
- ExampleApplication.assembly() line: 148
- ExampleApplication.run() line: 67

A callout highlights the `checking()` method, which is enclosed in a dashed orange box. Below the stack trace, the code for the `checking()` method is displayed, with the command pointer (indicated by a green bar) positioned at the beginning of the second line of code:

```
public boolean checking(){
    // Tries to move gripper. If assembly was performed, motion is stopped by ...
    ForceCondition checkAssembly = ForceCondition.createNormalForceCondition(grippe
    IMotionContainer controlMotion = gripper.move(linRel(0.0, 0.0, -15.0).break
```

Fig. 21-9: Position of command pointer in the `checking()` method

If the `run()` method is selected in the stack trace of the task thread, the current position of the command pointer in the `run()` method is displayed:

```

Thread [Main thread for application.ExampleApplication]
  ExampleApplication.checking() line: 170
  ExampleApplication.assembly() line: 148
  ExampleApplication.run() line: 67

public void run() {
    // While loop: Process is r
    while(partsProduced < 100){

        // Moves robot to a def
        moveToStart();

        // Process of picking u
        pickupWorkpiece();

        // Assembly process
        assembly();
    }
}

```

Fig. 21-10: Position of command pointer in the run() method

The filled white arrow icon indicates the call of assembly(), and thus the progress of the task in the run() method.

21.3.5 Overview of the toolbar in the “Debugging” view

Program execution is controlled by means of the buttons in the toolbar. Keyboard commands can alternatively be used for most functions.

Button	Name/description
	Resume Key: F8 Execution of a task is continued until the next break point or the end of the task is reached. (">>>> 21.3.5.1 "Continuing execution (Resume)" Page 624)
	Step in Key: F5 If the current command line contains an instruction, it is executed. If the current command line is a method call, the command pointer jumps to the start of the called method. (">>>> 21.3.5.2 "Jump into the method (Step in)" Page 624)
	Step over Key: F6 The current command line is executed completely. If the line contains a method call, the method is executed completely (exception: the method has breakpoints of its own). (">>>> 21.3.5.3 "Executing a method completely (Step over)" Page 625)

Button	Name/description
	<p>Step back Key: F7 The method currently being executed is executed through to the end as long as there are no breakpoints present before the end of the method. Task execution then stops in the calling method. (>>> <i>21.3.5.4 "Terminating the executed method (Step back)" Page 626</i>)</p>
	<p>Back to frame No key assigned This function can be used to jump to a point in the source code that has already been executed. (>>> <i>21.3.5.5 "Executing code sections again (Back to frame)" Page 627</i>)</p>
	<p>Pause No key assigned Pauses execution. (>>> <i>21.3.5.7 "Debugging: pausing threads (Pause)" Page 629</i>)</p>
---	<p>Execution to line (only available as a keyboard shortcut) Key combination: Ctrl+R Task execution is resumed until the command pointer reaches a command line defined by the user. (>>> <i>21.3.5.6 "Defining the code section to be executed (Execution to line)" Page 628</i>)</p>

21.3.5.1 Continuing execution (Resume)

The **Resume** button is used to continue execution of a task until the next break point or the end of the task is reached.

21.3.5.2 Jump into the method (Step in)

Description

If the current command line contains a method call, the command pointer jumps to the start of the called method when **Step in** is used.



The source code of the called method is only displayed if the source code of this method is available. If the source code is not available, the warning **Source not found** is displayed.

- Execution can be resumed.
- The user has no way of viewing the command currently being executed.
- In this case, **Step back** takes the user back to source code that can be displayed.
(>>> [21.3.5.4 "Terminating the executed method \(Step back\)"](#)
[Page 626](#))
- Use of **Step over** is recommended for jumping into a motion command (robot.move(...)).
(>>> [21.3.5.3 "Executing a method completely \(Step over\)"](#)
[Page 625](#))

If the current command line contains not a method call, but an individual instruction, the command line is executed and the command pointer jumps to the next command line.

Example

The application was interrupted before the call of the pickupWorkpiece() method. **Step in** causes the command pointer to jump to the start of the method:

```

while(partsProduced < 100) {
    // Moves robot to a defined start position
    moveToStart();

    // Process of picking up workpiece
    pickupWorkpiece();

    // Assembly process
    assembly();
}

private void pickupWorkpiece() {
    robot.move(ptp(getFrame("/Workpiece_Pickup/Starting_Point")));
    gripper.move(lin(getFrame("/Workpiece_Pickup/Approach")));
    gripper.move(linRel(0.0, 0.0, 10.0));
}

```

Fig. 21-11: Jump to method

21.3.5.3 Executing a method completely (Step over)

Description

Step over executes the current command line and the command pointer jumps to the next program line.

If the command line contains a method call, the method is executed completely as long as it does not contain a break point.

Formatting

Since the source code is executed step by step during remote debugging, the formatting of the source text influences the number of steps required for complete execution of the commands when using **Step over**.

Formatting example: Object of type `CartesianImpedanceControlMode`
With the following formatting, 3 steps are required when using **Step over**:

```
CartesianImpedanceControlMode mode =
new CartesianImpedanceControlMode();
mode.parametrize(CartDOF.Z).setStiffness(500);
```

If the code is divided by further line breaks, the code section is completely executed after a total of 4 steps with the following formatting when using **Step over**:

```
CartesianImpedanceControlMode mode =
new CartesianImpedanceControlMode();
mode.parametrize(CartDOF.Z)
.setStiffness(500);
```

Example

The application was interrupted before execution of the pickupWorkpiece() method. **Step over** completely executes the method and the command pointer jumps to the following command line.

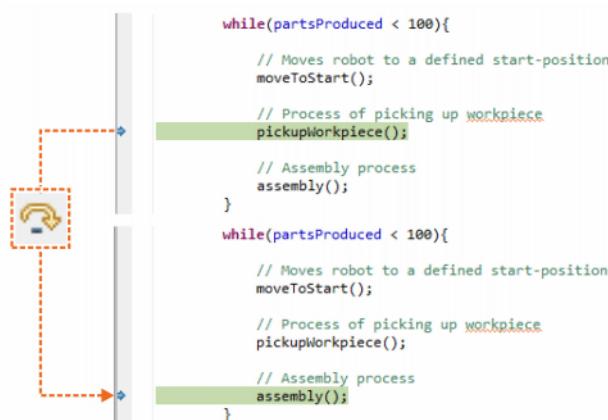


Fig. 21-12: “Step over” method

21.3.5.4 Terminating the executed method (**Step back**)

Description

Step back causes the method in which the command pointer is currently located to be executed completely (as long as there are no further break points present). The command pointer returns to the calling method and jumps to the following command line. Program execution is paused.



If, when **Step back** is used, program execution is interrupted before the end of the method has been reached and resumed with **Resume**, the pause requested by **Step back** is invalidated. In this case, execution of the task is not interrupted at the end of the method.

Example

The command pointer is located inside the pickupWorkpiece() method that was called by the run() method. With **Step back**, the pickupWorkpiece() method is executed completely and execution of the application is stopped before the next command line in the run() method:

```

public void pickupWorkpiece() {
    robot.move(ptp(getFrame("/Workpiece_Pickup/Starting_Point")));
    gripper.move(lin(getFrame("/Workpiece_Pickup/Approach")));
    gripper.move(linRel(0.0, 0.0, 10.0));
    // Moves gripper to workpiece. Motion is stopped if force > 10 N
    double threshold = 10;
    ForceCondition workpieceDetection = ForceCondition.createNormalForceCondition(
        gripper.getFrame("/Grip_Workpiece"), CoordinateAxis.Z, threshold);
    gripper.move(linRel(0.0, 0.0, 100.0).breakWhen(workpieceDetection).setCartVelocity(50));
    // ...
}

@Override
public void run() {
    // While loop: Process is repeated until 100 parts are produced
    while(partsProduced < 100) {
        // Moves robot to a defined start position
        moveToStart();

        // Process of picking up workpiece
        pickupWorkpiece();
        // Assembly process
        assembly();
    }
}

```

Fig. 21-13: “Step back” to calling method

21.3.5.5 Executing code sections again (Back to frame)

NOTICE

Back to frame changes the normal program execution. If state-changing interventions are carried out in the current method or its submethods, this can result in unexpected system behavior and unexpected robot motions.

Interventions and commands that cause a change of state include:

- Motion commands
- Modification of variables
- Modification of the source code
- Setting inputs/outputs
- Changes of values, e.g. by calling set methods

Description

Back to frame can be used to run program sections that have already been executed again. As standard, the command pointer jumps to the start of the method that is currently being executed. Program execution is then paused.

In the **Debugging** view, it is possible to return to each call level of the task using the stack trace. To do so, the desired method is selected in the stack trace. **Back to frame** causes the command pointer to jump to the start of this method.

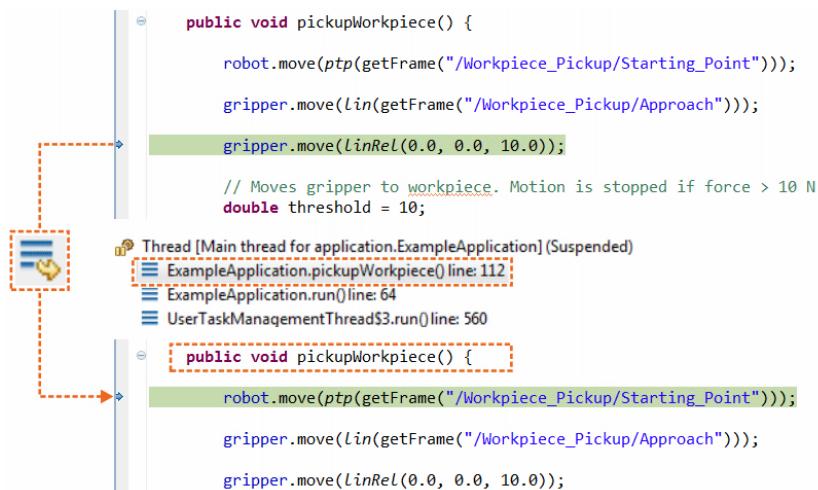
Once the command pointer has been placed at a previously executed position in the code by means of **Back to frame**, the following code can be executed (again).



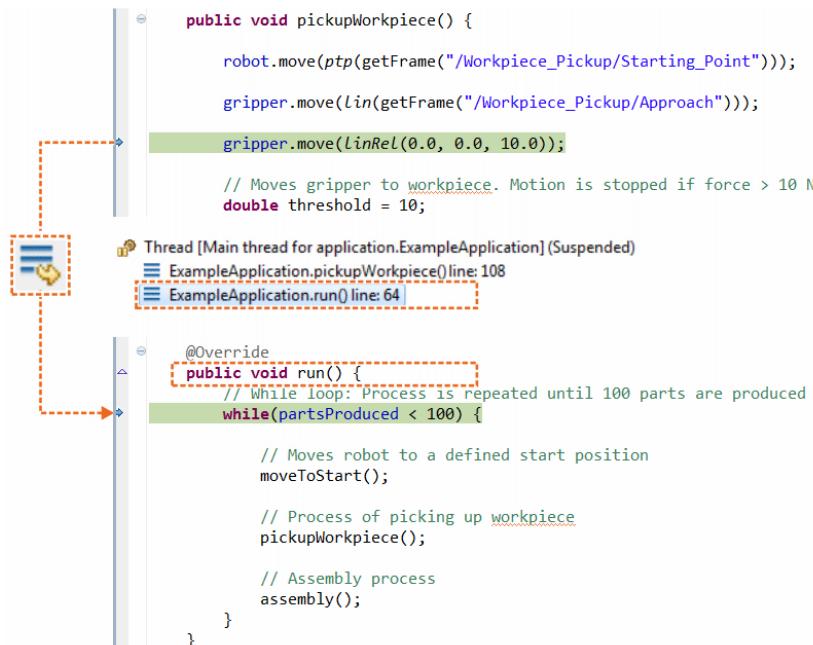
When **Back to frame** is used, modifications to arrays or external data that were not carried out in the affected code section are not undone.

Example

The command pointer is currently located in the `pickupWorkpiece()` method. As standard, **Back to frame** causes it to jump back to the start of the method. Execution of the task is resumed from there.

**Fig. 21-14: Back to frame (jump to start of current method)**

If the run() method is first selected in the stack trace of the task, **Back to frame** causes the command pointer to jump to the start of the run() method:

**Fig. 21-15: Back to frame (jump to start of run() method)**

21.3.5.6 Defining the code section to be executed (Execution to line)

Description

With **Execution to line**, the program is resumed until the command pointer reaches a command line defined by the user. **Execution to line** is not available in the **Debugging** view.

Procedure

1. Left-click into the line to which the task is to be executed. The line is highlighted with a blue background.
2. Task execution is resumed as far as the selected line or a preceding break point by means of the keyboard shortcut Ctrl+R.

Alternatively, the function can be selected from the context menu **Execution to line** after right-clicking into the desired command line.



The request for pausing task execution at the selected command line is only valid once. If execution is stopped before the command line is reached, and then resumed with **Resume**, execution is not stopped when the command line is reached.

21.3.5.7 Debugging: pausing threads (Pause)

Task execution can be paused manually by pressing the **Pause** button.



If the **Pause** function is used, the user must ensure that the corresponding thread task is selected in the **Debugging** view. The functioning of the controller may otherwise be adversely affected to such an extent that a reboot of the controller is required. Motion commands that have already been sent to the controller are not paused by the **Pause** function, but processed in the controller and executed.

When pausing, as when reaching a break point, the current command line is displayed in the editor area. If the corresponding source code is not available when using **Pause**, the warning **Source not found** is displayed in the editor area.



The Start/Pause key on the smartPAD is only used to pause motion commands. Pausing via the smartPAD only affects execution of the application if a synchronous motion command is due to be executed, as the command pointer only jumps to the next motion line after the motion command has been completed.

21.3.6 Variables view

The **Variables** view is integrated into the **Debugging** perspective.

It contains a table with all variables that are available at the currently indicated position of the command pointer.

The variables are not available during execution of a task. The updated values are only displayed while execution is paused.



If variable values change during remote debugging, program execution will be modified.

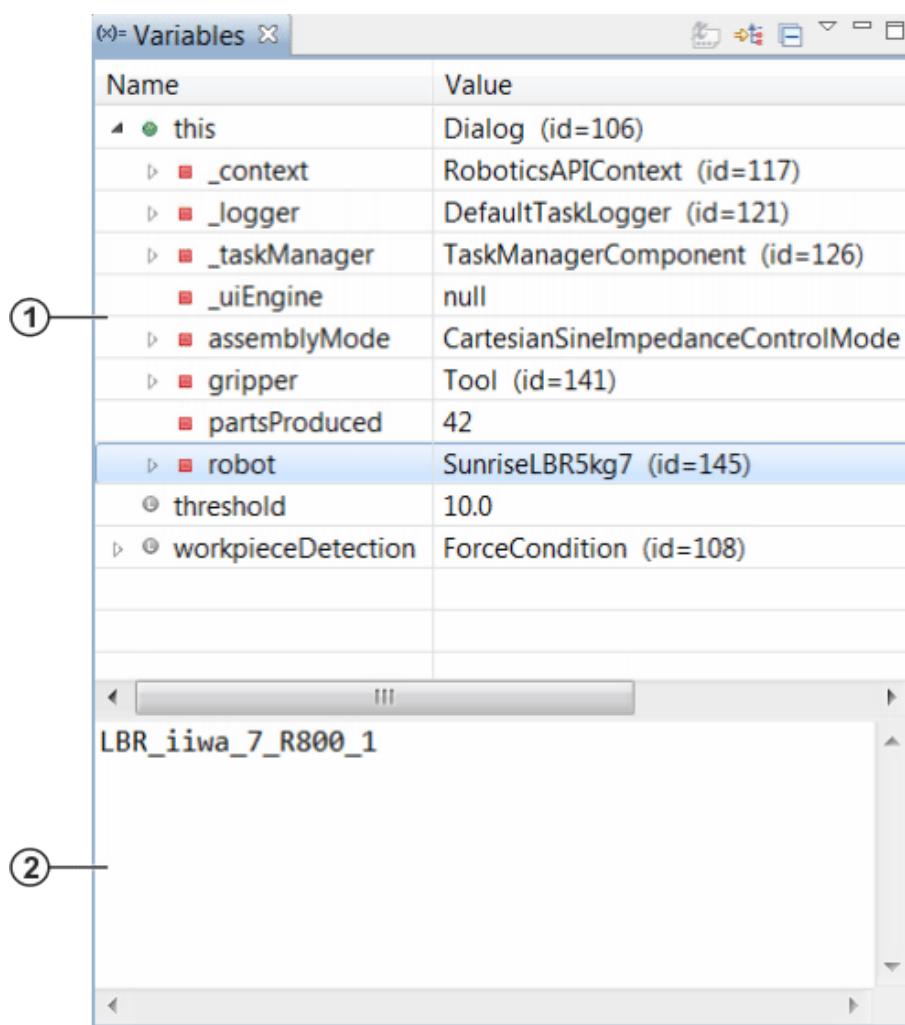


Fig. 21-16: Variables view

Item	Description
1	<p>Table of available variables</p> <p>The table contains the currently available arrays and local variables and their values. Only those variables that are available at the position of the command pointer in the selected method in the stack trace of the Debugging view are displayed.</p> <p>The Name column contains the variable name. Variables with a complex type are displayed hierarchically. Variables with complex data types can be expanded and their arrays displayed using the icon to the left of the name.</p> <p>The current value of the variable is displayed in the Value column. In the case of variables with complex data types, the result of the call of the <code>toString()</code> method is displayed as standard. The values of primitive data types and string values can be modified directly in the table.</p>
2	<p>Detailed information</p> <p>This area contains detailed information about the variable selected in the table. The variable value is displayed for primitive data types and strings. In the case of complex data types, the result of the call of the <code>toString()</code> method is displayed as standard.</p>

21.3.6.1 Displaying and modifying variables

Description

Irrespective of their visibility, variables and their values can be displayed and modified in the **Variables** view.

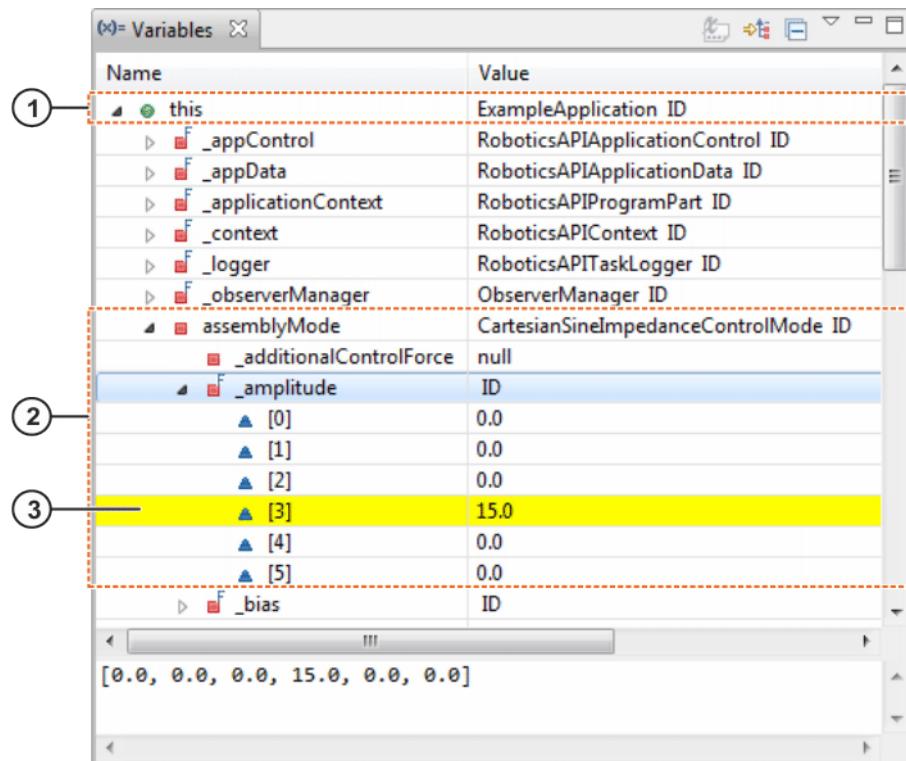


Fig. 21-17: Displaying variables

Item	Description
1	Instance The variable this refers to the instance of the class whose method has been selected in the stack trace and in whose source code the command pointer is currently displayed. During remote debugging of a task, the robot that is being used can be accessed, e.g. via the instance of the class. Here, this is the application for which remote debugging is being carried out.
2	Representation of complex data types Variables with a complex data type (here the class <code>CartesianSineImpedanceControlMode</code>) are displayed in a hierarchical structure. Expanding the structure displays the arrays of the referenced object. Arrays of primitive data types and strings are at the bottom level.
3	Changes of values The values of primitive data types and string values can be modified directly in the table. Once a value has been modified, the variable is highlighted in yellow in the table.

Precondition

- Task execution is paused.

Procedure

As standard, only those variables that are available at the position of the command pointer in the selected method in the stack trace of the **Debugging** view are displayed:

1. In order to display variables that are available in a different method, select the method in the stack trace of the **Debugging** view.
2. In order to modify variables of primitive data types, left-click on the value of the variable displayed in the **Value** column.
3. Enter the new value and confirm with the Enter key.



If incorrect values are entered, a message is displayed and the old value is retained. However, only incorrect entries that are recognized as such by the autocorrect function of the Java editor are prevented.

New values can be assigned to variables with complex data types in the dialog **Change object value**:

1. Right-click on the desired variable in the table and select **Change value...** from the context menu. The **Change object value** dialog opens.
2. Enter the corresponding instructions in the editor box.



When making entries, it must be ensured that syntactically correct Java source code is used and that a value with a suitable data type is returned at the end of the sequence of instructions.

21.3.6.2 Advanced context help for variables

If task execution is paused during remote debugging, the Java editor has advanced context help for variables. The advanced context help is then available for all variables that are available at the position of the command pointer in the selected method in the stack trace.

To display the context help, the mouse pointer is moved over the desired variable in the source code. A window opens, displaying information about the variables (data type, name, current value).

Complex data types are displayed in a hierarchical structure, like in the **Variables** view. Expanding the structure displays the arrays of the referenced object. Elementary data types and strings are located at the bottom hierarchy level.

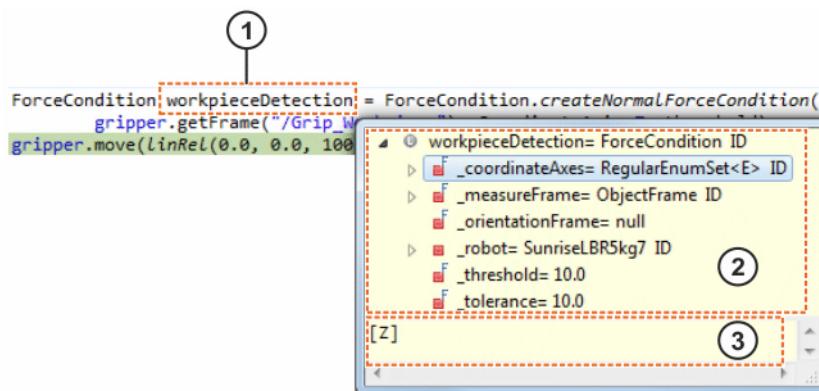


Fig. 21-18: Advanced context help

Item	Description
1	Variable (source code) Variable in the source code for which the advanced context help is displayed.
2	Variable (context help) Advanced context help for the variable. The designation and value are displayed. In the case of complex data types, the data type is also specified. Variables with a complex type are displayed hierarchically in a tree structure.
3	Details Details of the selected component are displayed here. In the case of variables with primitive data types and strings, the corresponding value is displayed; in the case of variables with a complex data type, the result of the call of <code>toString()</code> is displayed as standard.

21.3.7 Monitoring processes

Description

During remote debugging, data can also be monitored that are not available as variables. These include, for example, the current position of the robot.

Monitoring expressions can be formulated in Sunrise.Workbench. The monitoring expressions are managed in the **Expressions** view and evaluated each time task execution is stopped during a debugging session. Both individual expressions and more complex instruction sequences can be entered. Correct syntax must be observed.

Configured monitoring expressions are not deleted after the end of the debugging session and are thus also taken into consideration in subsequent debugging sessions.

NOTICE

It is recommended that monitoring expressions are only used for requesting states and that no state-changing commands are used in the expressions.

Interventions and commands that cause a change of state include:

- Motion commands
- Modification of variables
- Modification of the source code
- Setting inputs/outputs
- Changes of values, e.g. by calling “set” methods

Procedure

Display the **Expressions** view:

- Menu sequence **Window > Show View**
- The **Expressions** view can be selected via the **Other...** menu item. The view can be found in the “Debugging” group.

Overview

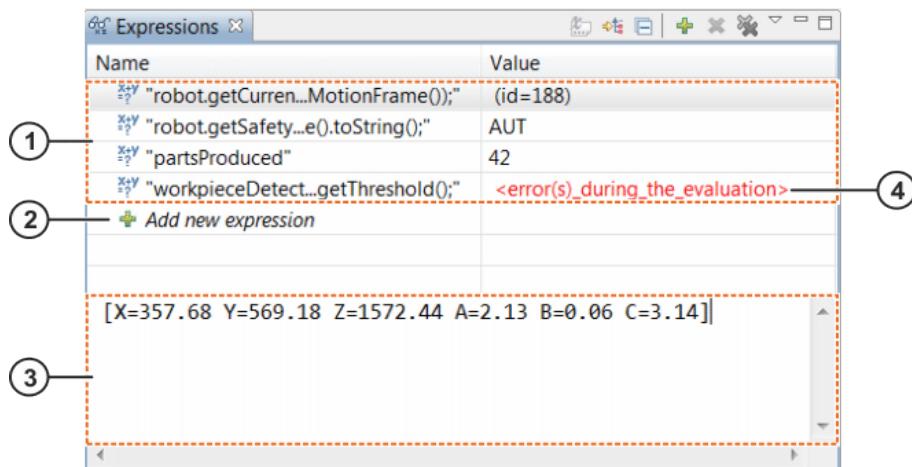


Fig. 21-19: “Expressions” view

Item	Description
1	Table of created monitoring expressions The Name column contains the source code of the monitoring expression. If available, the return value of the expression is specified under Value .
2	Line for new expression New expressions can be entered in the first unoccupied line of the table.
3	Details Detailed information about the selected expression is displayed in this area. As standard, complex data types are the result of the call of <code>toString()</code> on the return value of the monitoring expression. For variables of primitive data types and strings, the corresponding value is displayed.
4	Evaluation error If an expression cannot be evaluated, an error message is displayed in the Value column.

21.3.7.1 Adding new monitoring expressions

Precondition

- **Expressions** view opened.

Procedure

1. Left-click into the first blank line (indicated by a green + symbol) in the **Name** column.
2. Enter the monitoring expression in the **Name** column and confirm with the Enter key. The monitoring expression is added.
If a debugging session is active and task execution has been stopped, the expression is evaluated immediately.

NOTICE

Monitoring expressions are not automatically deleted after the end of the debugging session and are thus also active in subsequent debugging sessions. The use of monitoring expressions may modify program execution. It is recommended not to use state-changing commands in monitoring expressions. Unexpected behavior may otherwise result.

21.3.7.2 Deleting monitoring expressions

Monitoring expressions can be deleted.

Precondition

- **Expressions** view opened.

Procedure

- Right-click in the line with the monitoring expression that is to be deleted.
Select the entry **Delete** from the context menu.

21.3.7.3 Evaluating monitoring expressions

Description

During remote debugging of a task, monitoring expressions are automatically re-evaluated and updated in the following situations:

- On stopping execution at a break point
- On stopping execution by means of the debugging function **Pause**
- At the end of execution of one of the following debugging functions:
 - **Step in**
 - **Step over**
 - **Step back**
 - **Back to frame**
 - **Execute to line**

If task execution is interrupted during debugging, evaluation of a monitoring expression can be explicitly requested.

Procedure

- Right-click in the line with the expression that is to be monitored.
Select the entry **Re-evaluate monitoring expression** in the context menu.

Evaluation of a monitoring expression is only successful if the command at the current position of the command pointer in the method selected in the stack trace of the task thread can be executed.

Example

During remote debugging of a task, the current Cartesian position of the tool TCP is to be displayed after every execution step. A monitoring expression is formulated for this.

The identifier of the robot array of the application is `robot` (data type: LBR). The gripper is represented by the `gripper` array (data type: com.kuka.roboticsAPI.geometricModel.Tool). The following command call is thus required for requesting the current position of the gripper TCP:

```
robot.getCurrentCartesianPosition(gripper.getDefaultMotionFrame());
```

This command is entered in the **Name** column in the **Expressions** view:

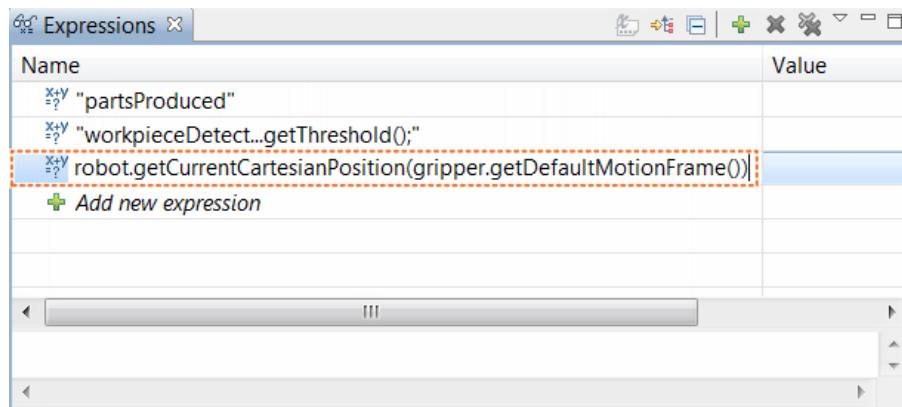


Fig. 21-20: Entering a monitoring expression

The monitoring expression is re-evaluated every time task execution is paused. The returned value is of type com.kuka.geometricModel.Frame. Complex data types are displayed hierarchically. Expanding the tree structure displays the arrays of the returned object.

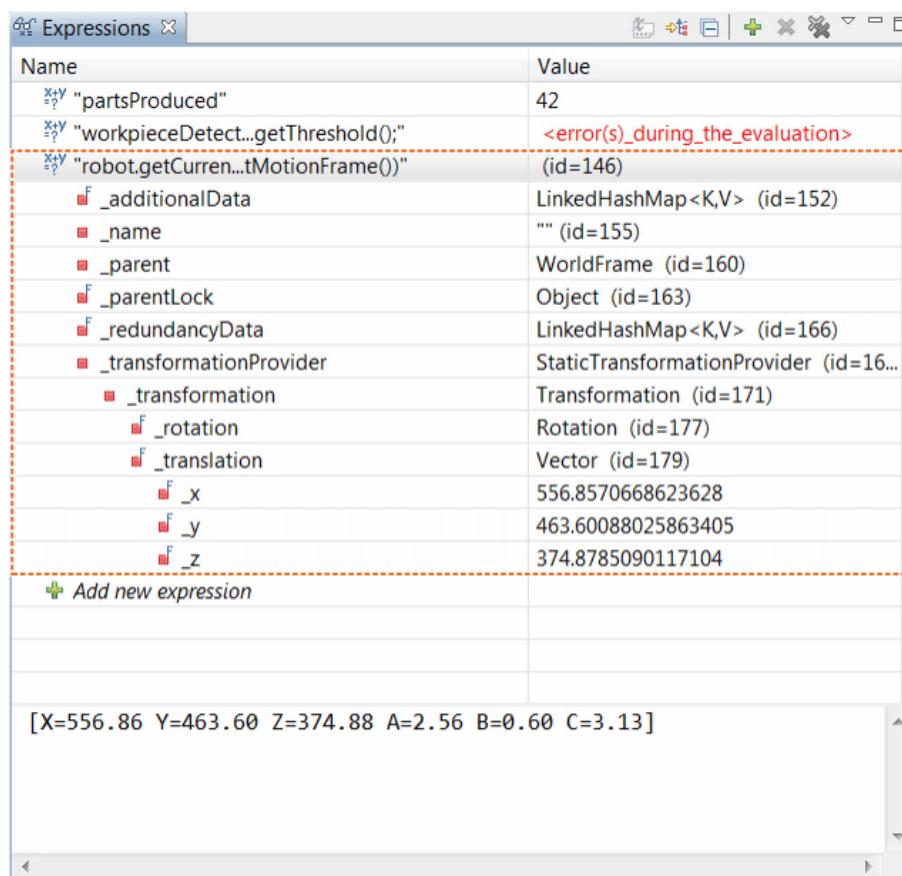


Fig. 21-21: Evaluating a monitoring expression

21.3.8 Modifying source code

During an active debugging session, it is possible to perform certain modifications in the source code of tasks and other classes.

The following must always be observed in the case of modifications to the source code during an active debugging session:

- Only modify source code while execution of the task is paused. Do not modify source code during execution of the task.
- Save modifications to the source code before starting or resuming execution of the task.

21.3.8.1 Impermissible modification of the source code

The following modifications to the source code may lead to complications and should thus not be made during an active debugging session:

- Addition of new methods or fields
- Modification of the designation of a method or field
- Modification of the data type of a field
- Modification of the return type of a method
- Modification of the number of transfer parameters of a method
- Modification of the data type of transfer parameters of a method
- Generation of syntax errors in the code



After saving an impermissible modification of the source code during remote debugging, a dialog opens with a corresponding warning message.

- **Next** button: The debugging session can be resumed.
- **Disconnect** button: The debugging session can be aborted and the remote connection disconnected. It is advisable to abort the debugging session and reestablish the remote connection.

21.3.8.2 Permissible modification of the source code

The following must be taken into consideration if, during debugging of a task, modifications are made in the source code of this task or in the source code of the classes used in it:

- If modifications are made to the source code in a method that is currently located in the stack trace of the task thread, the command pointer jumps to the start of this method after saving the change.

NOTICE

The jump by the command pointer to the start of a method on saving modifies the normal program sequence. This can result in unexpected system behavior and unexpected robot motions.

22 Appendix

22.1 Risk management

Following the integration of the LBR Med, the medical product manufacturer must evaluate its risk management for the medical product or the overall system and take the appropriate measures. The following list provides a rough guideline. The risk management process must also take the attached documentation (Important information on LBR Med) into consideration:

- Suitability of the drape used:

For use in a sterile environment, a drape can be used to avoid the risk of contamination and infection. The suitability of the drape used must be evaluated in the risk management process for the medical product. The drape used must meet the following requirements as it concerns intended use:

- Unrestricted robot mobility
- Manual guidance mode must not be adversely affected: A stretched drape must not cause any additional external loads.
- The drape must not be damaged or relocated during robot motions, e.g. by lifting.
- The medical product manufacturer must provide the model and type designations for the approved drape to the end user (customer).
- The drape used must not lead to overheating. The ambient conditions and the surface temperatures must be taken into account.

- Avoidance of the risk of contamination and infection due to liquids:

On account of its design, the robot exhibits gaps and open cavities in which liquids could collect. The medical product manufacturer must evaluate the appropriate measures to avoid the risk of contamination and infection in its risk management process and implement these measures. Possible measures:

- Preventing liquids from collecting (e.g. condensation)
- Using a drape

- Parts that come in contact with the patient:

- The robot and the media flange are not applied parts:

According to IEC 60601-1, cl. 4.6, the medical product manufacturer must evaluate in its risk management process whether the parts that could come into contact with the patient (but are not applied parts) must meet the requirements for applied parts. The medical product manufacturer must determine whether these components must comply with the requirements for type B, BF or CF applied parts.

The robot and the media flange meet the requirements for type B applied parts.

The medical product manufacturer must finally evaluate the following in its risk management plan:

- Which parts of the robot and the media flange could come in contact with a patient.
- Location, probability and duration of contact with the robot and the media flange by a patient.

- Responsiveness and the necessity of protecting a patient in the case of contact with the robot and the media flange by a patient.
- The robot controller must not be touched by the patient:
The medical product manufacturer must ensure that the robot controller is not accessible to unauthorized personnel or patients, e.g. by using an additional cover. The robot controller must only be accessible to maintenance personnel through the use of tools.
The robot controller is not intended for direct operation in a sterile environment. If the intended use requires this, the medical product manufacturer must evaluate this in its risk management and take corresponding measures (e.g. additional housing).
- Leakage currents in the overall system:
The medical product manufacturer must ensure that the limit values for leakage currents are not exceeded. The leakage currents must be measured for the overall system in accordance with the classification of the applied part.
- Hazard due to vibrations:
Vibrations may occur on the robot during operation. The transmission of vibrations can lead to “discomfort” for patients or end users. The medical product manufacturer must minimize the occurrence of vibrations on the overall system. The medical product manufacturer must evaluate this through its risk management and take the appropriate measures.
- Unexpected stopping:
The medical product manufacturer must ensure that unexpected stopping of the robot does not lead to an unacceptable risk.
The medical product manufacturer must define and implement appropriate measures which eliminate the corresponding risk or reduce it to an acceptable level.
- Stopping of the robot in the event of a fault
In its risk management, the medical product manufacturer must evaluate what behavior is required of the robot when stopping due to a fault. If the risk management permits it, a safety stop 1 for stopping the robot in an emergency is to be preferred to stopping using the holding brakes (safety stop 0) in order to prevent unnecessary wear of the holding brakes.
- Single fault safety:
To ensure single fault safety for the robot, the medical product manufacturer must correctly configure the safety configuration in Sunrise Workbench based on its own, application-specific risk management process.
The functions described in the “Operating and Programming Instructions for System Integrators” are available to the medical product manufacturer for this.
- Setting up alarm systems:
To meet the requirements from the standard IEC 60601-1-8, the medical product manufacturer can set up alarm systems which are based on the data of the LBR Med. The medical product manufacturer must define suitable alarm conditions and the selection of the values to be monitored.
- Integration of the PEMS from the medical product manufacturer into the user’s IT network:
The medical product manufacturer must analyze the associated hazard that could arise from integration of the LBR Med into IT networks. The medical product manufacturer must evaluate this via a threat analysis

in its risk management process as well as take and document protective measures. The information must be made available to the user by the system integrator according to IEC 60601-1, cl. 14.13.

The following hazards can arise when integrating the LBR Med into an IT network:

- Incorrect flow of data between the system of the integrator and the robot controller.

Possible remedies:

- Secure sequence control prevents misinterpretations of incorrect commands via the data flow
- Secure communications protocol (CRC, time stamp, packet counter)
- Standard protocol for starting, pausing and restarting the application
- Excessive loading of the IT network via network nodes.

Possible remedies:

- Asynchronous information between the system of the integrator and the robot controller
- Watchdogs for time-critical sequences
- Cyclical sequences with requirements for hard real time only in the real-time part of the controller
- Direct or indirect access to the LBR Med from the IT network.

Possible remedies:

- Complete insulation of the robot controller on the network side of the IT network (incoming and outgoing connections)
- System hardening of the integrator against network access
- Protection of the robot controller against direct physical access

- Brakes as a safety measure:

The brakes of the robot are considered as a safety measure and are only applied during operation in the event of a fault. The integrity of the brakes is ensured through a cyclical brake test.



Further information about the brake test is contained in the “Operating and Programming Instructions for System Integrators”.

The reduction of the holding torque is detected in good time before reaching the minimum permissible holding torque for the brakes and signaled by the system using the **Warning** message. The medical product manufacturer must take the cyclical execution of the brake test into account in the development and planning of the application.

- Trapping hazard:

In its risk management, the medical product manufacturer must evaluate the hazard to the operator (e.g. medical personnel) due to crushing at potential trapping points. The medical product manufacturer must take appropriate measures to ensure that the robot motion is stopped if a trapping hazard point is reached.

The safety functions must be configured accordingly for this.

- Properties of the sensor system:

For applications with high requirements in terms of accuracy, it must be ensured that all measured values are assigned defined tolerances (e.g. for space monitoring). For further information concerning tolerance values, please contact KUKA Customer Support.

- Use of an applied part with an additional energy supply system:
In the case of installation of an applied part with an additional energy supply system, the medical product manufacturer must ensure that the maximum rated payload is not exceeded on account of the additional weight.
The medical product manufacturer must evaluate the use of an applied part with an additional energy supply system in respect of the following in its risk management process:
 - The impact on safety-oriented monitoring functions (e.g. collision detection).
 - The impact on manual guidance.
 - The impact on motions which use impedance functions.
- Hazard due to strong magnets:
Using the robot in the vicinity of magnetic fields can result in malfunctions, functional failure or damage to the robot. The medical product manufacturer must evaluate this through its risk management and take the appropriate measures.
- Hazard due to unverified absolutely accurate robot model:
The absolute accuracy of the robot cannot be assured without an external check. Remastering is not sufficient to operate a robot with absolute accuracy. If the robot is operated without an external check of absolute accuracy, death, severe injuries or damage to property may result.
The following situations can lead to a change in the robot structure and a loss of absolute accuracy:
 - Forces applied during transportation
 - Improper installation of the robot or of the tool on the flange
 - Excessively high process forces during operation
 - Collision during operation
 - Collision after operation, in the switched-off state
 - Aging and wearPossible measures:
 - When integrating the robot into a medical product, the system integrator must ensure that the robot structure is checked for a change after the application of force and that the absolutely accurate model is re-verified.
 - In his risk management, the system integrator must evaluate the hazards that can arise during operation and during aging and wear, take measures and document them. Information about the external verification of absolute accuracy must be provided to the user by the system integrator.

23 KUKA Service

23.1 Requesting support

Introduction

This documentation provides information on operation and operator control, and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

Information

The following information is required for processing a support request:

- Description of the problem, including information about the duration and frequency of the fault
- The greatest possible amount of information about the hardware and software components of the overall system

The following list gives an indication of the information which is relevant in many cases:

- Model and serial number of the kinematic system, e.g. the manipulator
 - Model and serial number of the controller
 - Model and serial number of the energy supply system
 - Designation and version of the system software
 - Designations and versions of other software components or modifications
 - System Software diagnosis package
- Additionally for KUKA Sunrise: Existing projects including applications
- For versions of KUKA System Software older than V8: Archive of the software (Diagnosis package is not yet available here.)
- Application used
 - External axes used

23.2 KUKA Customer Support

The contact details of the local subsidiaries can be found at:

www.kuka.com/customer-service-contacts

Index

“Brake” safety reaction.....	550
“Ready for motion”, checking.....	431
3-point method.....	145
6D mouse.....	80
A	
ABC 2-point method.....	142
ABC world method.....	144
Accessories.....	25, 29
Activating, safety configuration.....	273
Activation delay, for safety function.....	300
Actual position, axis-specific.....	123
Actual position, Cartesian.....	124
addCartesianForce(...).	481
addCartesianTorque(...).	482
addCommandedCartesianPositionXYZ(...).	482
addCommandedJointPosition(...).	482
addControllerListener(...).	431, 436
addCurrentCartesianPositionXYZ(...).	483
addCurrentJointPosition(...).	482
addDoubleUserKey(...).	489
addExternalJointTorque(...).	481
addInternalJointTorque(...).	481
addMonitoringListener().	155
addUserKey(...).	489
Administrator.....	184, 185
Allow muting via input.....	341
AMF.....	20
API.....	20
App_Enable.....	222, 231
App_Start.....	222, 513
Appendix.....	639
Application data (view).	58
Application mode.....	102
Application override.....	101, 119, 120, 438
Application tool.....	105
Application, pausing.....	501
Approximate positioning.....	354
Approximate positioning point.....	354
areAllAxesGMSReferenced().	435
areAllAxesPositionReferenced().	436
Asynchronous motion execution.....	386
attachTo(...).	405
AUT (operating mode).	31
AutExt_Active.....	223
AutExt_AppReadyToStart.....	223, 513
Auto-complete.....	369
Automatic (operating mode)	
AUT.....	31
Automatic mode.....	51
Automatic mode (AMF).	248, 250, 276
Auxiliary point.....	346, 388
awaitFileAvailable(...).	486
Axis-specific impedance controller.....	567, 592
Axis-specific monitoring spaces, defining.....	295
Axis-specific robot position, requesting.....	425
Axis limit.....	295
Axis range.....	31, 295

Axis range monitoring (AMF).....	248, 251, 295
Axis torque condition.....	443
Axis torque monitoring.....	300
Axis torque monitoring (AMF)....	249, 252, 301
Axis torques, requesting.....	419
Axis velocity monitoring (AMF)... 249, 251, 282	

B

Background application, new.....	65
Background application, starting.....	121, 122
Background application, stopping.....	121
Background task, cyclic.....	535
Background tasks.....	533
Backup Manager.....	128, 203
Backup Manager, configuration.....	198
Base-related TCP force component (AMF).....	249, 252, 306, 337
Base calibration.....	145
Base coordinate system.....	99, 145
Base for jogging.....	163
Blocking wait.....	478
BooleanICondition.....	442
Brake defect.....	45
Brake ramp monitoring, Brake.....	551
Brake test.....	151
BrakeTestDiskSpaceState(Enum).....	156
Braking distance.....	31
Break conditions for motions.....	464
Break conditions, evaluating.....	466
Break point, conditional.....	618
Break point, view.....	617
Break points.....	614
breakWhen().....	465
breakWhen(...).	466
Bus I/Os, mapping.....	217

C

Calibration.....	138
Calibration, base.....	145
Calibration, tool.....	138
cancel().....	502
Cartesian impedance controller..	567, 569, 579
Cartesian position, requesting.....	425
Cartesian protected space monitoring (AMF).....	249, 251, 292
Cartesian protected spaces, defining.....	292
Cartesian setpoint/actual value difference, requesting.....	427
Cartesian velocity monitoring (AMF)..	249, 251, 284
Cartesian workspace monitoring (AMF)... 249, 251,	291
Cartesian workspaces.....	290
CartesianTorqueCondition.....	441
CB Test Certificate.....	30
CB Test Report.....	30
CE mark.....	30
checkFreeDiskSpace().....	155

Checksum, safety configuration.....	273	Debug project (button).....	60
CIRC.....	388	Debugging session, ending.....	610
CIRC, motion type.....	346	Debugging session, starting.....	609
Circular motion.....	388	Debugging, view.....	621
Cleaning work.....	52	Declaration.....	377
clipApplicationOverride(...).....	439	Declaration of conformity.....	30
clipManualOverride(...).....	439	Declaration of incorporation.....	30
Collision detection.....	301	Decommissioning.....	52
Collision detection (AMF).....	249, 252, 302	Default application.....	67, 221, 225, 234
Command pointer.....	620	Default frame for motions.....	175
Complex conditions.....	442	Default user group.....	95, 184, 185
Complex data types.....	378	DefaultApp_Error.....	223
Condition for Boolean signals.....	463	Dependency injection.....	379, 380, 382
Condition for the range of values of a signal	463	detach().....	408
Condition, Cartesian torque.....	452	Development process.....	239
Conditional branch.....	506	Diagnosis.....	597
Connecting cables.....	25, 29	Diagnosis package.....	643
Connection manager.....	80	Diagnosis package, creating.....	604, 605
Constant force, overlaying.....	588	Diagnosis package, loading from the robot controller.....	605
Contamination.....	47	Diagnostic information, collecting.....	604
Continuous Path.....	345	Disclaimer.....	29
Controller object, creating.....	568	Display child frames (button).....	111
Controller parameters		displayModalDialog(...).....	500
Defining.....	568	Disposal.....	52
Coordinate system, for jog keys.....	85	Distance component condition.....	462
Coordinate systems.....	99	Distance condition.....	461
Counting loop.....	503	Distributor.....	23
CP motion.....	345	DO WHILE loop.....	505
CP spline block.....	345	Documentation, LBR Med.....	19
CP spline block, creating.....	392		
Create child frame (button).....	111, 112		
Create frame (button).....	111, 112		
createAndEnableConditionObserver(...).....	476	E	
createConditionObserver(...).....	476	EC declaration of conformity.....	30
createDesiredForce(...).....	587	Effective program override.....	119, 120, 438
createLissajousPattern(...).....	587	EMC Directive.....	30
createNormalForceCondition(...).....	445, 447	EMERGENCY STOP.....	76, 80
createShearForceCondition(...).....	445, 448	EMERGENCY STOP device.....	37, 38, 40
createSinePattern(...).....	587	EMERGENCY STOP smartPAD (AMF)....	249, 275
createSpatialForceCondition(...).....	445, 446	EMERGENCY STOP, external.....	38, 40
createSpatialTorqueCondition(...).....	454	enable(), DataRecorder.....	484
createSpiralPattern(...).....	587	Enabling device.....	37, 38
createTiltingTorqueCondition(...).....	454, 456	Enabling device, external.....	38, 40
createTurningTorqueCondition(...).....	454, 455	Enabling switch.....	78
createUserKeyBar(...).....	488	Enabling switches.....	38
CRR.....	31, 96	End user.....	23
CSV.....	20	equals(...).....	466
Customer.....	23	Error treatment.....	514
CyclicMonitoringState(Enum).....	156	ESD.....	31
		ESM.....	20, 262
		ESM mechanism.....	267
		ESM state, deleting.....	269
		ESM state, new.....	267
D		EtherCAT.....	20
Danger zone.....	31	Ethernet interface.....	20, 22
Data types.....	376	EVC.....	21
Data, backing up manually.....	132	Event-driven Safety Monitoring.....	247
Data, recording and evaluating.....	479	Exception.....	21
Data, restoring manually.....	132	executeBrakeTest().....	154
DataRecorder.....	479, 480		
Debug (perspective).....	59		

D

Danger zone.....	31
Data types.....	376
Data, backing up manually.....	132
Data, recording and evaluating.....	479
Data, restoring manually.....	132
DataRecorder.....	479, 480
Debug (perspective).....	59

Expert.....	95, 185	getEmergencyStopEx().....	434
Extended AMF.....	252	getEmergencyStopInt().....	434
External control.....	221	getEnablingDeviceState().....	434
External E-STOP.....	277	getExecutionMode().....	437
External position referencing.....	312, 316	getExternalForceTorque(...).....	421, 422
F			
Fast entry, Java.....	369	getExternalTorque().....	419
Faults.....	46	getFailedConditionsForNormalUse().....	520
Field bus diagnosis.....	597	getFiredBreakConditionInfo().....	466
Field bus, Ethernet-based.....	196	getFiredCondition().....	466, 467, 472
Field buses, overview.....	209	getFiredStopCondition().....	530
File, closing.....	59	getFlange().....	405
Filter settings.....	599	getForce().....	422
Flange coordinate system.....	99	getForceInaccuracy().....	423
Fonts.....	376	getFrame(...).....	406, 408
FOR loop.....	503	getGammaRad().....	428
Force component condition.....	450	getHomePosition().....	429
Force condition.....	444	getManualOverride().....	438
ForceComponentCondition.....	441	getMeasuredTorque().....	419
ForceCondition.....	441	getMissedEvents().....	472
Frame.....	21	getMotion().....	513
Frame management.....	161	getMotionContainer().....	472
Frame, creating.....	111	getObserverManager().....	475, 478
Frame, creating for application.....	162	getOperationMode().....	434
Frame, deleting.....	164	getOperationModeVelocityLimit().....	553
Frame, designating as base.....	163	getOperatorSafetyState().....	434
Frame, moving.....	164	getPositionInfo().....	466
Frame, properties, application data.....	165	getPositionInformation().....	427, 472
Frame, properties, object templates.....	173	getPositionInformation(...).....	424
Frame, teaching with hand guiding device..	114	getRecovery().....	512
FrameDistanceComponentCondition.....	441	getRecoveryStrategy().....	512
FrameDistanceCondition.....	441	getRotationOffset().....	427
Frames, manually addressing.....	115	getSafetyState().....	433
Frames, reteaching.....	112	getSafetyStopSignal().....	434
Frames, view.....	109	getSafetyVelocityLimit().....	554
FSoE.....	21	getSingleTorqueValue(...).....	419
Function test.....	48	getStartPosition().....	513
G			
General safety measures.....	45	getStateOfFreeDiskSpace().....	158
Get_State.....	222	getStatusDescription().....	563
getAggregatedVelocityLimit().....	554	getStatusGroup().....	563
getAlphaRad().....	428	getStoppedMotion().....	466, 468
getApplicationData().getFrame(...).....	167	getTimeTillBraketestOverdue().....	155
getApplicationOverride().....	438	getTorque().....	422
getApplicationUI().....	488	getTorqueInaccuracy().....	423
getBetaRad().....	428	getTorqueValues().....	419
getCommandedCartesianPosition().....	472	getTranslationOffset().....	427
getCommandedCartesianPosition(...).....	424	getTriggerTime().....	472
getCommandedJointPosition().....	424, 472	getVelocityLimitSources().....	554
getCurrentCartesianPosition().....	424, 472	GMS.....	21
getCurrentCyclicMonitoringState().....	158	Graphics card.....	55
getCurrentJointPosition().....	424, 472	Groups of persons.....	33
getCurrentState().....	154		
getDate().....	563		
getDeviceVelocityLimit().....	553		
getEffectiveOverride().....	439		
H			
halt().....	501		
Hand guiding device enabling activated (AMF).....	249, 250, 279		
Hand guiding device enabling deactivated (AMF).....	249, 250, 278		
Handguiding Support.....	193, 196		
handGuiding().....	353, 398		

Handling of failed motion commands.....	514	isPostponementAcceptable().....	158
hasActiveMotionCommand().....	432	isPostponementPossible().....	154
hasChangedActiveStatusGroups().....	563	isReadyForNormalUse().....	520
High-velocity mode (AMF).....	248, 250, 276	isReadyToMove().....	431
HOME position.....	428	isRecording().....	486
HOME position, changing.....	428	isRecoveryRequired().....	512
HOME position, requesting.....	429	isRecoveryRequired(...).....	512
Hooke's law.....	570	IStatusController, interface.....	558, 560
HRC.....	22	IStatusListener, interface.....	558
		IStatusMonitor, interface.....	558, 561
		isTorqueValid(...).....	423
		ISunriseControllerStateListener.....	436
		IT security.....	46
I/O configuration, exporting.....	219	ITaskFunctionMonitor.....	544
I/O configuration, new.....	210	ITorqueSensitiveRobot (interface).....	419
I/O configuration, opening.....	210	ITriggerAction, interface.....	470
I/O group, creating.....	214	IUserKeyBar (interface).....	489
I/O group, deleting.....	215		
I/O group, editing.....	214		
I/O group, exporting as a template.....	216		
I/O group, importing from a template.....	216		
I/O Mapping (window).....	217, 218		
IAnyEdgeListener.....	474		
IApplicationOverrideControl (interface).....	438		
ICallbackAction, interface.....	470		
ICondition (interface).....	440		
IControllerStateListener.....	431		
Identification plate.....	78, 79, 82		
IF ELSE branch.....	506		
IFallingEdgeListener.....	474		
IForceSensitiveRobot (interface).....	420		
Information about robot and robot controller.....	128		
Initialization.....	378		
initialize().....	368, 535, 538		
initializeCyclic(...).....	535		
Input signal (AMF).....	248, 250, 277		
Inputs/outputs, display.....	125		
Installation.....	193		
Installation direction.....	193		
Installation, KUKA Sunrise.Workbench.....	55		
Intended use.....	28		
Introduction.....	19		
invalidateMastering(...).....	525		
IORangeCondition.....	442		
IP address ranges, blocked.....	61		
IP address, robot controller.....	128		
IRecovery, interface.....	512		
IRisingEdgeListener.....	474		
ISafetyState (interface).....	433		
isAxisGMSReferenced(...).....	435		
isAxisMastered(...).....	525		
isAxisPositionReferenced(...).....	435		
isBrakeTestExecutable().....	155, 158		
isDiskFull().....	155, 158		
isEnabled().....	486		
isFileAvailable().....	486		
isForceValid(...).....	423		
isInHome().....	429		
isJoggingActive().....	529		
isMastered().....	430		
		K	
		Keyboard key.....	77, 80
		Keypad.....	88
		KLI.....	21, 193, 196
		Knowledge and skills, required.....	19
		KRCDiag.....	604
		KUKA Customer Support.....	128, 643
		KUKA Line Interface.....	196
		KUKA PSM.....	261, 275
		KUKA RoboticsAPI.....	21
		KUKA Service.....	643
		KUKA smartHMI.....	21, 85

KUKA smartPAD.....	21, 32	Monitoring, physical safeguards.....	39
KUKA smartPAD-2.....	21, 32	Motion command, canceling.....	502
KUKA Sunrise Cabinet.....	21	Motion enable (AMF).....	248, 250, 276
KUKA Sunrise Cabinet Med.....	25	Motion execution, pausing.....	502
KUKA Sunrise.EnhancedVelocityControl.....	549	Motion programming, basic principles.....	345
KUKA Sunrise.OS.....	21	Motion types.....	345
KUKA Sunrise.StatusController.....	557	MotionBatch.....	390
L			
Labeling.....	43	MotionPathCondition.....	441
Language.....	86, 94	Mounting orientation.....	62, 99
Language package, installing.....	206	move(...).....	386, 408, 514
Language selection (button).....	86	moveAsync(...).....	386, 408, 516
LBR.....	22	Multiple branches.....	508
LBR Med.....	22, 29		
Licenses.....	23		
LIN.....	388		
LIN REL.....	389		
LIN, motion type.....	346		
Linear motion.....	388, 389		
Lissajous oscillation, overlaying.....	589		
Load data.....	170		
Log entries, filtering.....	600		
Log, displaying.....	597		
Log, view.....	598		
Loops, nesting.....	511		
Low Voltage Directive.....	30		
M			
Main menu key.....	77, 80	Object management.....	168
Main menu, calling.....	92	Object properties, displaying/editing.....	171
Maintenance.....	51	Object template, copying.....	176
Manipulator.....	35	Object templates (view).....	58
Manual guidance mode.....	398	ObserverManager.....	475, 478
Manual guidance, axis limitation.....	400	Old project, loading.....	202
Manual guidance, motion type.....	353	onIsReadyToMoveChanged(...).....	431
Manual guidance, programming.....	398	onKeyEvent(...), IUserKeyListener.....	491
Manual guidance, velocity limitation.....	402	onSafetyStateChanged(...).....	436
Manual mode.....	50	onTriggerFired(...).....	470
Manual override.....	101, 118, 120, 438	Open-source.....	23
Mapping, inputs/outputs.....	219	Operating hours meter.....	128
Mass of the heaviest workpiece.....	342	Operating mode, changing.....	97
masterAxis(...).....	525	Operating time.....	128
Mastering.....	137	Operation, KUKA smartPAD.....	75
Mastering state, requesting.....	430	Operation, KUKA Sunrise.Workbench.....	57
Mastering(...).....	525	Operator.....	95, 184, 185
Mastering, deleting.....	138	Operator safety.....	37, 39
Med.....	22	Operators.....	34, 442
Media flange Touch.....	277, 278	Option package, installing.....	204
Medical device manufacturer.....	23	Option package, removing from robot	
Menu bar.....	58	controller.....	207
Message programming.....	498	Option package, uninstalling.....	206
Message window.....	116	Option packages.....	203
Methods, extracting.....	370	Options.....	25, 29
Mode selection.....	41	Orientation control.....	396, 397
Mode selector switch.....	80	Orientation control, LIN, CIRC, SPL.....	356
Monitoring of processes.....	442, 473	Output, change.....	125
Monitoring spaces.....	288	OverallBrakeTestResult(Enum).....	157
		Overload.....	45

Override.....	105, 118, 119	Properties (view).....	59
Override (button).....	86, 100	Protected space.....	288, 292
Override, changing and requesting.....	438	Protective equipment.....	43
Overview, motion parameters.....	394, 399	PSM.....	22
Overview, robot status.....	520	PSM mechanism.....	264
Overview, robot system.....	25	PTP.....	387
Overview, servo controllers.....	567	PTP, motion type.....	345
Overview, Sunrise project.....	161	PTPRecoveryStrategy (class).....	513
Overview, Sunrise.RolesRights.....	185		

P

Package Explorer (view).....	58
Panic position.....	39
Password, changing.....	186
Path-related condition.....	458
Path-related switching actions.....	442, 469
Pausing, robot application.....	119, 120
PDS firmware update.....	136
PEMS.....	22
Performance Level.....	37
Permanent Safety Monitoring.....	246
Personal protective equipment.....	33
Perspective, selection.....	58
Perspectives, displaying.....	59
Plant integrator.....	23, 32
PLC.....	22
Point-to-point.....	345
Point-to-point motion.....	387
Position and torque referencing.....	312
Position controller.....	567, 569
Position referencing.....	312
Position referencing (AMF).....	249, 251, 280
positionHold(...).	595
Post-test loop.....	505
postponeBrakeTest().	154
PPE.....	33
Preventive maintenance work.....	52
Primitive data types.....	378
Processor.....	55
Product description.....	25
PROFINET.....	22
PROFIsafe.....	22
Program execution.....	115
Program execution control.....	501
Program run mode, changing and requesting.....	437
Program run mode, setting.....	117
Program run modes.....	118
Programming.....	367
Programming (perspective).....	59
Programming, compliant robot.....	567
Project management.....	161
Project synchronization.....	187
Project, loading from the robot controller....	190
Project, synchronization.....	187
Project, transferring to the robot controller.	188
Project, updating.....	189
Projects, archiving.....	68
Projects, loading to workspace.....	68, 69

R

RAM.....	55
Reaction distance.....	31
Ready for motion signal, reacting to changes.....	431
Recommissioning.....	47, 135
Reduced-velocity mode (AMF)....	248, 250, 276
Redundancy angle.....	362
Redundancy information.....	166, 361
Reference, canceling.....	71
Referencing state, requesting.....	435
Rejecting loop.....	504
Release notes, displaying.....	73
Remote debugging.....	607
removeMonitoringListener().....	155
Renaming variables.....	368
Repair.....	51
Requesting, robot position.....	424
Requirements on SOUP components.....	239
Reset (button).....	120
Resetting, robot application.....	120
Retraction, robot.....	96
Robot.....	22, 25, 29
Robot activity, checking.....	432
Robot application, pausing.....	119, 120
Robot application, resetting.....	120
Robot application, selecting.....	115
Robot application, starting automatically....	119
Robot application, starting manually.....	119
Robot base coordinate system.....	99
Robot controller.....	29
Robot controller, switching off.....	135
Robot controller, switching on.....	135
Robot controller, switching on/off.....	135
Robot level.....	90
Robot position, requesting.....	424
Robot, repositioning.....	120
RoboticsAPI.....	21
RobotStateAggregator().....	520
run().....	368, 538
runCyclic().....	535

S

Safe operational stop.....	299
Safe operational stop, external.....	38, 41
Safeguards, external.....	44
Safety.....	29
Safety-oriented functions.....	37
Safety-oriented reactions.....	244

Safety-oriented stop reactions.....	35	setJointJerkRel(...)	396, 397
Safety-oriented tool, configuring.....	176	setJointLimitsEnabled(...)	399
Safety-oriented tools, mapping.....	271	setJointLimitsMax(...)	399
Safety acceptance overview.....	317	setJointLimitsMin(...)	400
Safety concept.....	243	setJointLimitViolationFreezesAll(...)	400
Safety configuration.....	237	setJointVelocityLimit(...)	400
Safety configuration, activating.....	274	setJointVelocityRel(...)	395, 397
Safety configuration, deactivating.....	274	setLED(...), IUserKey	495
Safety configuration, opening.....	261	setMaxCartesianVelocity(...)	578
Safety configuration, restoring.....	274	setMaxControlForce(...)	577
Safety controller, resuming.....	98	setMaxPathDeviation(...)	578
Safety function, new for ESM.....	269	setNullSpaceDamping(...)	576
Safety function, new for PSM.....	265	setNullSpaceStiffness(...)	576
Safety functions.....	36	setOrientationReferenceSystem(...)	358, 397
Safety functions, configuration.....	264	setOrientationType(...)	356, 396
Safety functions, configuring.....	267	setPermanentPullOnViolationAtStart(...)	400
Safety functions, deactivation via an input.	258	setPhaseDeg(...)	583
Safety instructions.....	19	setPositionLimit(...)	585
Safety interfaces.....	245	setRelativeVelocityForInitialBrakeTestMotion()	156
Safety maintenance.....	185	setResponseTime(time)	154
Safety maintenance technicians.....	95, 185	setRiseTime(...)	586
Safety signal, state, requesting.....	433	setSafetyWorkpiece(...)	412
Safety stop.....	32	setStayActiveUntilPatternFinished(...)	586
Safety stop 0.....	32	setStiffness(...)	575, 593
Safety stop 1.....	32	setStiffnessForAllJoints(...)	593
Safety stop 1 (path-maintaining).....	32	setTcp(...)	529
Safety stop, external.....	38, 40	setText(...), IUserKey	494
Safety zone.....	32, 34, 35	setTotalTime(...)	586
Safety, general.....	29	Shear force	445
SafetyConfiguration.sconf (file).....	63, 261	Simple force oscillation, overlaying	588
Serial number, robot.....	128	Single fault safety	242
Serial number, robot controller.....	128	Single point of control	53
Service life.....	31	Singletons	382, 410
Service phases.....	43	Singularities	363
Servo controllers, overview.....	567	System-dependent	365
Set base for jogging (button).....	111	Singularity	357
Set methods.....	394, 399	smartHMI	22, 85
setAdditionalControlForce(...)	576	smartPAD	22, 32, 45
setAmplitude(...)	583	smartPAD enabling switch deactivated (AMF)	249, 275
setApplicationOverride(...)	439	smartPAD enabling switch panic activated (AMF)	249, 275
setBase(...)	529	smartPAD unplugging allowed	341
setBias(...)	584	smartPAD, connecting	83
setBlendingCart(...)	396	smartPAD, disconnecting	82
setBlendingOri(...)	396	smartPAD, software update	84, 135
setBlendingRel(...)	396	Software	25, 29
setCartAcceleration(...)	395	Software classification	238, 241
setCartJerk(...)	396	Software components	25
setCartVelocity(...)	395	Software limit switches	42
setCartVelocityLimit(...)	400	Software requirements	240
setCriticalText(...), IUserKey	496, 497	SOUP components	239, 240
setDamping(...)	575, 593	SOUP components, retrofitting	240
setDampingForAllJoints(...)	594	Space Mouse	76
setExecutionMode(...)	437	Spiral-shaped force oscillation, overlaying	591
setFallTime(...)	586	SPL, motion type	347
setForceLimit(...)	584	Spline segment	347
setFrequency(...)	583	Spline, motion type	347
setHoldTime(...)	586	SPOC	53
setHomePosition(...)	428		
setInitialPositionForBrakeTest()	155		
setJointAccelerationRel(...)	395, 397		

Standard frame for motions.....	171	Target group.....	19
Standstill monitoring.....	299	Task, remote debugging.....	608
Standstill monitoring of all axes (AMF)....	249,	Tasks (view).....	59
252,	299	TCP.....	23, 138, 139, 172
Start-up.....	47, 135	TCP force monitoring.....	303
Start backwards key.....	77, 80	TCP force monitoring (AMF)....	249, 252, 303,
Start key.....	77, 78, 80, 81	337	
startCartesianJogging(...).....	528	Teach pendant	25, 29
Starting, robot application.....	119	Teaching.....	111, 112, 114
startJointJogging(...).....	527	Template, for Sunrise project.....	61
startRecording().....	484	Templates.....	369
StartRecordingAction.....	484	Templates, user-specific.....	370
Station configuration.....	193	Terms used.....	20
Station configuration, overview.....	193	Terms, safety.....	31
Station level.....	88	Test mode (AMF).....	248, 250, 276
Station_Error.....	223	Tilting torque.....	453
StationSetup.cat (file).....	63, 193	Time delay (AMF).....	249, 253, 300
Status.....	362	Tool-related velocity component (AMF)....	249,
Status display.....	87	251,	287
Status monitor.....	561	Tool calibration.....	138
Stop category 0.....	32	Tool Center Point.....	138, 139
Stop category 1.....	32	Tool coordinate system.....	99, 138, 139
Stop category 1 (path-maintaining).....	32	Tool frame, creating.....	172
STOP key.....	77, 80	Tool load data, determining.....	147
Stop reactions, safety-oriented.....	35	Tool orientation (AMF).....	249, 251, 297
stopCartesianJogging().....	527	Tool orientation, monitoring.....	296
stopCartesianJogging(...).....	528	Tool, creating.....	169
stopJoggingAtAxisLimitOrSingularity(...).....	529	Tool, integrating.....	403
Stopping distance.....	31, 35, 290	Tool, switching off.....	276
stopRecording().....	485	Toolbars.....	58
StopRecordingAction.....	485	Torque.....	453
Storage.....	52	Torque component condition.....	457
Structure, robot application.....	367	Torque referencing.....	313
Sunrise applications.....	64	Torque referencing (AMF).....	249, 251, 281
Sunrise I/Os, changing.....	215	TorqueComponentCondition.....	441
Sunrise I/Os, creating.....	211	Torques, axis-specific.....	125
Sunrise I/Os, deleting.....	215	Touch screen.....	75
Sunrise project.....	23	Trademarks.....	20
Sunrise project, new.....	61	Training.....	19
Sunrise project, overview.....	161	Transportation.....	47
Sunrise.StatusController.....	557	Trigger.....	442, 469
Sunrise.Workbench, starting.....	57	Trigger information, evaluating.....	471
Sunrise.Workbench, user interface.....	57	Triggers, programming.....	469
SunriseExecutionService.....	437	triggerWhen(...).....	469
SunriseSafetyState (class).....	433	Turn.....	363
Support request.....	643	Type, robot.....	128
Surface normal.....	445		
SWITCH branch.....	508		
Symbols.....	376		
Synchronize project (button).....	60		
Synchronous motion execution.....	386		
System integrator.....	23, 30, 32, 33		
System requirements, PC.....	55		
System Software, installing.....	200		
System states, requesting.....	429		

T

T1 (operating mode).....	33
T2 (operating mode).....	33

U

Uninstallation, option package.....	206
Uninstallation, Sunrise.Workbench.....	55
Unmastering.....	138
updateCartesianJogging(...).....	528
updateJointJogging().....	527
USB connection.....	78
Use, contrary to intended use.....	29
Use, improper.....	29
User.....	31, 33
User dialogs, programming.....	499
User group (button).....	86

User group, changing.....	96
User groups.....	95
User interface, KUKA smartHMI.....	85
User interface, Sunrise.Workbench.....	57
User key bar, creating.....	488
User key selection (button).....	98
User keys.....	77, 80
User keys (button).....	86
User keys, activating.....	97
User keys, defining.....	487
User management.....	184
User messages, programming.....	498
User PSM.....	262
UserKeyAlignment (enum).....	493

V

Variables, renaming.....	368
Velocity.....	105
Velocity monitoring functions.....	282
Velocity monitoring, T1.....	42
Version, System Software.....	128
View, Backup Manager.....	129
View, frames.....	109
View, log.....	598
Views, repositioning.....	59
Virus scanner.....	203
Virus scanner, displaying messages.....	603
Virus scanner, installing.....	205
VxWorks.....	23

W

waitFor(...).	478
Warnings.....	19
WHILE loop.....	504
Workpiece frame, creating.....	172
Workpiece load data, entering.....	183
Workpiece, creating.....	169
Workpiece, integrating.....	403
Workpieces, safety-oriented use.....	181
Workspace.....	31, 34, 35, 288, 290, 295
Workspace, new.....	67
Workspace, Sunrise.Workbench.....	67
Workspace, switching.....	68
Workspaces, switching.....	68
World coordinate system.....	99

X

XYZ 4-point method.....	140
-------------------------	-----