

KUKA

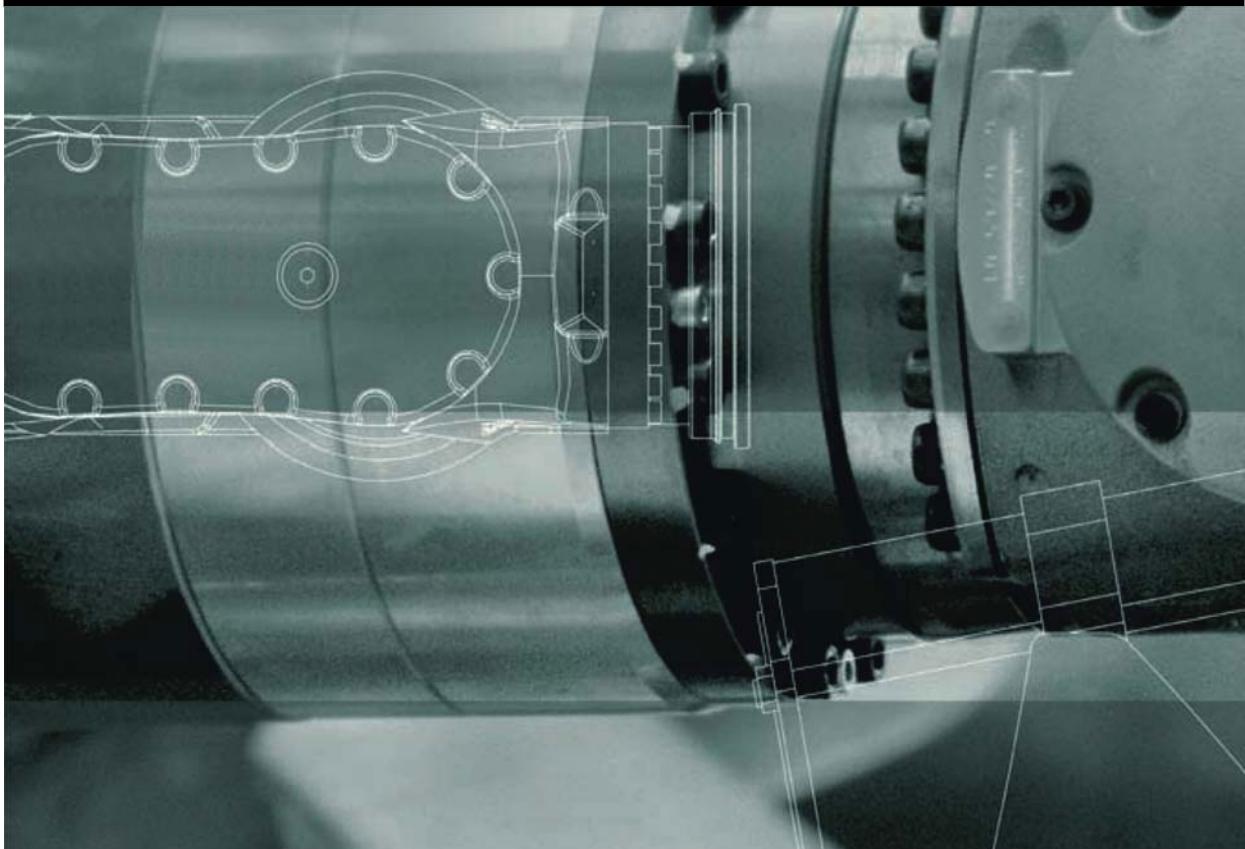
System Software

KUKA Laboratories GmbH

KUKA Sunrise.OS 1.1 KUKA Sunrise.Workbench 1.1

For LBR iiwa

Operating and Programming Instructions



Issued: 07.05.2014

Version: KUKA Sunrise.OS 1.1 V1

© Copyright 2014

KUKA Laboratories 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 the KUKA Laboratories 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.

Translation of the original documentation

KIM-PS5-DOC

Publication: Pub KUKA Sunrise.OS 1.1 en

Book structure: KUKA Sunrise.OS 1.1 V1.1

Version: KUKA Sunrise.OS 1.1 V1

Contents

1	Introduction	13
1.1	Target group	13
1.2	Industrial robot documentation	13
1.3	Representation of warnings and notes	13
1.4	Trademarks	14
1.5	Terms used	14
2	Product description	17
2.1	Overview of the robot system	17
2.2	Overview of the software components	18
2.3	Overview of KUKA Sunrise.OS	18
2.4	Overview of KUKA Sunrise.Workbench	19
2.5	Intended use of the system software	19
3	Safety	21
3.1	Legal framework	21
3.1.1	Liability	21
3.1.2	Intended use of the industrial robot	21
3.1.3	EC declaration of conformity and declaration of incorporation	22
3.2	Safety functions	22
3.2.1	Terms used	23
3.2.2	Personnel	24
3.2.3	Workspace, safety zone and danger zone	25
3.2.4	Safety-oriented functions	26
3.2.4.1	EMERGENCY STOP device	27
3.2.4.2	Enabling device	27
3.2.4.3	Operator safety	28
3.2.4.4	External EMERGENCY STOP device	28
3.2.4.5	External safety stop 1	28
3.2.4.6	External enabling device	29
3.2.4.7	External safe operational stop	29
3.2.5	Triggers for safety-oriented stop reactions	29
3.2.6	Non-safety-oriented functions	30
3.2.6.1	Mode selection	30
3.2.6.2	Software limit switches	31
3.3	Additional protective equipment	31
3.3.1	Jog mode	31
3.3.2	Labeling on the industrial robot	31
3.3.3	External safeguards	32
3.4	Safety measures	32
3.4.1	General safety measures	32
3.4.2	Transportation	33
3.4.3	Start-up and recommissioning	34
3.4.4	Manual mode	35
3.4.5	Automatic mode	36
3.4.6	Maintenance and repair	36
3.4.7	Decommissioning, storage and disposal	37
3.4.8	Safety measures for “single point of control”	37

3.5	Applied norms and regulations	38
4	Installing KUKA Sunrise.Workbench	41
4.1	PC system requirements	41
4.2	Installing KUKA Sunrise.Workbench	41
4.3	Uninstalling KUKA Sunrise.Workbench	41
5	Operation of KUKA Sunrise.Workbench	43
5.1	Starting KUKA Sunrise.Workbench	43
5.2	Overview of the user interface of KUKA Sunrise.Workbench	43
5.2.1	Repositioning the views	45
5.2.2	Displaying different perspectives on the user interface	45
5.2.3	Toolbars	46
5.3	Creating a Sunrise project with a template	46
5.4	Creating a new robot application	49
5.4.1	Creating a new Java package	49
5.4.2	Creating a robot application with a package	49
5.4.3	Creating a robot application for an existing package	50
5.5	Creating a new background task	50
5.5.1	Creating a background task with a package	50
5.5.2	Creating a background task for an existing package	50
5.6	Workspace	51
5.6.1	Creating a new workspace	51
5.6.2	Switching to an existing workspace	51
5.6.3	Switching between the most recently opened workspaces	51
5.6.4	Archiving projects	51
5.6.5	Loading projects from archive to the workspace	52
5.6.6	Loading projects from the directory to the workspace	52
5.7	Sunrise projects with referenced Java projects	52
5.7.1	Creating a new Java project	53
5.7.1.1	Inserting robot-specific class libraries in a Java project	53
5.7.2	Referencing Java projects	54
5.7.3	Canceling the reference to Java projects	54
5.8	Renaming an element in the “Package Explorer”	54
5.9	Removing elements in the “Package Explorer”	54
5.9.1	Removing elements from a project	55
5.9.2	Removing projects	55
6	Operating the KUKA smartPAD	57
6.1	KUKA smartPAD control panel	57
6.1.1	Front view	57
6.1.2	Rear view	59
6.2	Switching the robot controller on/off	60
6.2.1	Switching on the robot controller and starting the system software	60
6.2.2	Switching off the robot controller	60
6.3	KUKA smartHMI user interface	61
6.3.1	Navigation bar	62
6.3.2	Status display	63
6.3.3	Keypad	63

6.3.4	Station view	64
6.3.5	Robots view	65
6.4	Calling the main menu	67
6.5	Changing the operating mode	68
6.6	Coordinate systems	69
6.7	Jogging the robot	70
6.7.1	“Jogging options” window	70
6.7.2	Setting the jog override (HOV)	72
6.7.3	Axis-specific jogging with the jog keys	72
6.7.4	Cartesian jogging with the jog keys	73
6.7.4.1	Null space motion	74
6.8	CRR mode – controlled robot retraction	74
6.9	Teaching and manually addressing frames	75
6.9.1	Displaying frames	75
6.9.2	Teaching frames	77
6.9.3	“Movement mode” window	78
6.9.4	Manually addressing frames	80
6.10	Program execution	80
6.10.1	Selecting a robot application	80
6.10.2	Selecting the program run mode	82
6.10.3	Setting the program override (POV)	83
6.10.4	Starting a program forwards (manually)	83
6.10.5	Starting a program forwards (automatically)	83
6.10.6	Repositioning the robot after leaving the path	84
6.11	Monitor functions	84
6.11.1	Displaying the end frame of the motion currently being executed	84
6.11.2	Displaying the axis-specific actual position	85
6.11.3	Displaying the Cartesian actual position	86
6.11.4	Displaying axis-specific torques	86
6.11.5	Displaying an I/O group and changing the value of an output	87
6.11.6	Displaying the IP address and software version	89
7	Start-up and recommissioning	91
7.1	Position mastering	91
7.1.1	Mastering axes	91
7.1.2	Manually unmastering axes	91
7.2	Determining tool load data	92
8	Brake test	95
8.1	Overview of the brake test	95
8.2	Creating the brake test application from the template	97
8.2.1	Changing the motion sequence for torque value determination	99
8.2.2	Removing torque value determination from the brake test application	100
8.2.3	Changing the starting position for the brake test	101
8.3	Programming interface for the brake test	101
8.3.1	Evaluating measured torques and determining the maximum absolute value	101
8.3.2	Polling the evaluation results of the maximum absolute torques	102
8.3.3	Creating an object for the brake test	104
8.3.4	Starting the execution of the brake test	104

8.3.5	Evaluating the brake test	105
8.3.5.1	Polling the results of the brake test	106
8.4	Performing a brake test	107
8.4.1	Evaluation results of the maximum absolute torques (display)	108
8.4.2	Results of the brake test (display)	108
9	Project management	111
9.1	Sunrise projects – overview	111
9.2	Frame management	111
9.2.1	Creating a new frame	112
9.2.2	Defining a frame as a “base for jogging”	112
9.2.3	Moving frames	113
9.2.4	Deleting frames	113
9.2.5	Displaying and modifying the properties of a frame	114
9.2.6	Inserting a frame in a motion instruction	116
9.3	Object management	116
9.3.1	Geometric structure of tools	116
9.3.2	Geometric structure of workpieces	117
9.3.3	Creating a tool or workpiece	118
9.3.4	Creating a frame for a tool or workpiece	118
9.3.5	Defining a default motion frame	119
9.3.6	Load data	120
9.3.6.1	Manually entering load data	120
9.3.7	Safety-oriented tool	120
9.3.7.1	Defining a safety-oriented tool	121
9.4	Synchronization of projects	122
9.4.1	Transferring the project to the robot controller	123
9.4.2	Updating the project on the robot controller or in Sunrise.Workbench	123
9.5	Loading the project from the robot controller	124
10	Station configuration and installation	127
10.1	Opening the station configuration	127
10.2	Installing the system software	127
11	Bus configuration	129
11.1	Configuration and I/O mapping in WorkVisual – overview	129
11.2	Creating a new I/O configuration	129
11.3	Opening an existing I/O configuration	130
11.4	Creating Sunrise I/Os	130
11.4.1	“Create I/O signals” window	131
11.4.2	Creating an I/O group and inputs/outputs within the group	132
11.4.3	Editing an I/O group	133
11.4.4	Deleting an I/O group	133
11.4.5	Changing an input/output of a group	133
11.4.6	Deleting an input/output of a group	133
11.4.7	Exporting an I/O group as a template	133
11.4.8	Importing an I/O group from a template	134
11.5	Mapping the bus I/Os	134
11.5.1	I/O Mapping window	134

11.5.2 Buttons in the “I/O Mapping” window	135
11.5.3 Mapping Sunrise I/Os	136
11.6 Exporting the I/O configuration to the Sunrise project	136
12 External control	139
12.1 Configuring external control	139
12.1.1 External control inputs	140
12.1.2 External control outputs	140
12.1.3 Configuring external control in the project properties	141
12.2 Selecting a robot application as the default application	142
12.3 Defining the signal outputs for a project that is not externally controlled	143
13 Safety configuration	145
13.1 Overview	145
13.2 Safety concept	145
13.3 Overview of Atomic Monitoring Functions	149
13.3.1 Standard Atomic Monitoring Functions	149
13.3.2 Parameterizable Atomic Monitoring Functions	150
13.3.3 Extended Atomic Monitoring Functions	151
13.4 Safety configuration with KUKA Sunrise.Workbench	151
13.4.1 Overview: changing the safety configuration and activating it on the controller ...	152
13.4.2 Opening the safety configuration	152
13.4.2.1 Evaluating the safety configuration	153
13.4.2.2 Overview of the PSM table “User PSM”	153
13.4.3 Creating and editing safety functions in Sunrise.Workbench	154
13.4.3.1 Creating new safety functions	154
13.4.3.2 Deleting a safety function	154
13.4.3.3 Editing existing safety functions	154
13.5 Activating the safety configuration on the robot controller	155
13.5.1 Deactivating the safety configuration	156
13.5.2 Restoring the safety configuration	156
13.5.3 Changing the password for activating the safety configuration	156
13.6 Use and parameterization of the Atomic Monitoring Functions	156
13.6.1 Evaluating the safety equipment on the KUKA smartPAD	157
13.6.2 Evaluating the operating mode	157
13.6.3 Evaluating the motion enable	157
13.6.4 Evaluating the position referencing	158
13.6.5 Evaluating the torque referencing	158
13.6.6 Monitoring safe inputs	159
13.6.7 Velocity monitoring functions	160
13.6.7.1 Defining axis-specific velocity monitoring	160
13.6.7.2 Defining Cartesian velocity monitoring	161
13.6.8 Monitoring spaces	161
13.6.8.1 Defining Cartesian workspaces	163
13.6.8.2 Defining Cartesian protected spaces	164
13.6.8.3 Defining axis-specific monitoring spaces	166
13.6.9 Standstill monitoring (safe operational stop)	167
13.6.10 Monitoring of forces and torques	168
13.6.10.1 Axis torque monitoring	168
13.6.10.2 Collision detection	169

13.6.10.3	TCP force monitoring	170
13.7	Example of a safety configuration	170
13.8	Position and torque referencing	172
13.8.1	Position referencing	172
13.8.2	Torque referencing	173
13.8.3	Creating an application for position and torque referencing	174
13.9	Safety acceptance overview	175
13.9.1	Checklist for general safety functions	176
13.9.2	Checklists for the safety-oriented tool	177
13.9.2.1	Geometry data of the safety-oriented tool	177
13.9.2.2	Load data of the safety-oriented tool	178
13.9.3	Checklist for PSM rows	178
13.9.4	Checklists for AMFs used	179
13.9.4.1	AMF "Emergency stop smartPAD"	179
13.9.4.2	AMF "Control panel enable smartPAD released"	179
13.9.4.3	AMF "Control panel panic smartPAD"	179
13.9.4.4	AMF "Operating mode Test"	179
13.9.4.5	AMF "Operating mode Automatic"	180
13.9.4.6	AMF "Operating mode with reduced speed"	180
13.9.4.7	AMF "Operating mode with high velocity"	180
13.9.4.8	AMF "Input"	180
13.9.4.9	AMF "Standstill monitoring all axes"	180
13.9.4.10	AMF "Motion enable"	180
13.9.4.11	AMF "Axis torque monitoring"	180
13.9.4.12	AMF "Axis velocity monitoring"	181
13.9.4.13	AMF "Position referencing"	181
13.9.4.14	AMF "Torque referencing"	181
13.9.4.15	AMF "Axis range monitoring"	181
13.9.4.16	AMF "Cartesian velocity"	182
13.9.4.17	AMF "Cartesian workspace monitoring" / "Cartesian protected space monitoring"	182
13.9.4.18	AMF "Collision detection"	182
13.9.4.19	AMF "TCP force monitoring"	182
14	Basic principles of motion programming	185
14.1	Overview of motion types	185
14.2	PTP motion type	185
14.3	LIN motion type	186
14.4	CIRC motion type	186
14.5	SPL motion type	187
14.6	Spline motion type	187
14.6.1	Velocity profile for spline motions	188
14.6.2	Modifications to spline blocks	190
14.6.3	LIN-SPL-LIN transition	192
14.7	Approximate positioning	193
14.8	Orientation control with LIN, CIRC, SPL	195
14.8.1	CIRC – reference system for the orientation control	197
14.8.2	CIRC – combinations of reference system and type for the orientation control ...	197
14.9	Redundancy information	199
14.9.1	Redundancy angle	200
14.9.2	Status	200

14.9.3 Turn	201
14.10 Singularities	202
14.10.1 Kinematic singularities	202
14.10.2 System-dependent singularities	204
15 Programming	205
15.1 Java Editor	205
15.1.1 Opening a robot application in the Java Editor	205
15.1.2 Structure of a robot application	205
15.1.3 Edit functions	206
15.1.3.1 Renaming a variable	206
15.1.3.2 Auto-complete	206
15.1.3.3 Templates – Fast entry of Java statements	207
15.1.3.4 Creating user-specific templates	207
15.1.3.5 Extracting methods	208
15.1.4 Displaying information in Javadoc	208
15.2 Symbols and fonts	209
15.3 Data types	210
15.4 Variables	210
15.5 RoboticsAPI version information	210
15.5.1 Displaying the RoboticsAPI version	211
15.5.2 Structure of the RoboticsAPI version number:	211
15.6 Motion programming: PTP, LIN, CIRC	211
15.6.1 Structure of a motion command (move/moveAsync)	211
15.6.2 PTP	212
15.6.3 LIN	213
15.6.4 CIRC	213
15.6.5 LIN REL	214
15.6.6 MotionBatch	215
15.7 Motion programming: spline	216
15.7.1 Programming tips for spline motions	216
15.7.2 Creating a CP spline block	217
15.7.3 Creating a JP spline block	218
15.7.4 Using spline in a motion instruction	218
15.8 Motion parameters	218
15.8.1 Programming axis-specific motion parameters	220
15.9 Using tools and workpieces in the program	221
15.9.1 Declaring tools and workpieces	221
15.9.2 Initializing tools and workpieces	222
15.9.3 Attaching tools and workpieces to the robot	222
15.9.3.1 Attaching a tool to the robot flange	222
15.9.3.2 Attaching a workpiece to other objects	223
15.9.3.3 Detaching objects	225
15.9.4 Moving tools and workpieces	225
15.10 Inputs/outputs	226
15.10.1 Creating a data array for an I/O group	227
15.10.2 Initializing a data array for an I/O group	227
15.10.3 Reading inputs/outputs	228
15.10.4 Setting outputs	228

15.11 Polling axis torques	229
15.12 Reading Cartesian forces and torques	230
15.12.1 Polling calculated force/torque data	230
15.12.2 Polling individual force/torque values	231
15.12.3 Checking the reliability of the calculated force/torque values	232
15.12.4 Polling individual values of a vector	233
15.13 Polling the robot position	233
15.13.1 Polling the axis-specific actual or setpoint position	234
15.13.2 Polling the Cartesian actual or setpoint position	235
15.13.3 Polling the Cartesian setpoint/actual value difference	236
15.14 HOME position	237
15.14.1 Changing the HOME position	237
15.15 Polling system states	238
15.15.1 Polling the HOME position	238
15.15.2 Polling the mastering state	239
15.15.3 Polling “ready for motion”	239
15.15.3.1 Reacting to changes in the “ready for motion” signal	240
15.15.4 Polling activity	240
15.15.5 Polling and evaluating safety signals	241
15.15.5.1 Polling the state of the safety signals	241
15.15.5.2 Reacting to a change in state of safety signals	243
15.16 Changing and polling the program run mode	243
15.17 Changing and polling the program override	244
15.18 Conditions	245
15.18.1 Conditions in the RoboticsAPI	245
15.18.2 Complex conditions	246
15.18.3 Axis torque condition	247
15.18.4 Force condition	248
15.18.4.1 Force condition for spatial force from all directions	249
15.18.4.2 Force condition for normal force	249
15.18.4.3 Force condition for shearing force	250
15.18.5 Force component condition	251
15.18.6 Path-related condition	253
15.18.7 Condition for Boolean signals	255
15.18.8 Condition for the range of values of a signal	255
15.19 Break conditions for motion commands	256
15.19.1 Defining break conditions	256
15.19.2 Evaluating the break conditions	256
15.19.2.1 Polling for the triggered break conditions	257
15.19.2.2 Polling for the robot position at the time of termination	258
15.20 Path-related switching actions (Trigger)	258
15.20.1 Programming triggers	258
15.20.2 Programming a path-related switching action	259
15.20.3 Evaluating trigger information	260
15.21 Monitoring processes (Monitoring)	262
15.21.1 Listener for monitoring conditions	262
15.21.2 Creating a listener object to monitor the condition	263
15.21.3 Registering a listener for notification of change in state	264
15.21.4 Activating or deactivating the notification service for listeners	265

15.21.5 Programming example for monitoring	265
15.22 Blocking wait for condition	266
15.23 Recording and evaluating data	267
15.23.1 Creating an object for data recording	267
15.23.2 Specifying data to be recorded	268
15.23.3 Starting data recording	270
15.23.4 Ending data recording	271
15.23.5 Polling states from the DataRecorder object	271
15.23.6 Example program for data recording	272
15.24 Message programming	273
15.24.1 Programming user messages	273
15.24.2 Programming user dialogs	274
15.25 Program execution control	275
15.25.1 Stopping an application	275
15.25.2 FOR loop	275
15.25.3 WHILE loop	276
15.25.4 DO WHILE loop	277
15.25.5 IF ELSE branch	278
15.25.6 Examples of nested loops	279
16 Background tasks	281
16.1 Using background tasks	281
16.2 Cyclic background task	281
16.3 Non-cyclic background task	283
17 Programming with a compliant robot	285
17.1 Sensors and control	285
17.2 Available controllers – overview	285
17.3 Using controllers in robot applications	285
17.3.1 Creating a controller object	286
17.3.2 Defining controller parameters	286
17.3.3 Transferring the controller object as a motion parameter	286
17.4 Position controller	286
17.5 Cartesian impedance controller	287
17.5.1 Calculation of the forces on the basis of Hooke's law	287
17.5.2 Parameterization of the impedance controller	289
17.5.2.1 Representation of Cartesian degrees of freedom	290
17.5.2.2 Defining controller parameters for individual degrees of freedom	290
17.5.2.3 Controller parameters specific to the degrees of freedom	291
17.5.2.4 Controller parameters independent of the degrees of freedom	292
17.6 Cartesian impedance controller with overlaid force oscillation	295
17.6.1 Overlaying a simple force oscillation	295
17.6.2 Overlaying superposed force oscillations (Lissajous curves)	296
17.6.3 Parameterization of the impedance controller with overlaid force oscillation	297
17.6.3.1 Controller parameters specific to the degrees of freedom	298
17.6.3.2 Controller parameters independent of the degrees of freedom	299
17.7 Static methods for impedance controller with superposed force oscillation	301
17.7.1 Overlaying a constant force	301
17.7.2 Overlaying a simple force oscillation	302

17.7.3	Overlaying a Lissajous oscillation	303
17.8	Holding the position under servo control	304
18	Diagnosis	307
18.1	Displaying field bus errors	307
18.1.1	General field bus errors	307
18.1.2	Error state of I/Os and I/O groups	307
18.2	Displaying the protocol	307
18.2.1	“Protocol” view	308
18.2.2	Filtering protocol entries	309
18.3	Collecting diagnostic information for error analysis at KUKA	310
18.3.1	Creating a diagnosis package with the smartHMI	311
18.3.2	Creating a diagnosis package with the smartPAD	311
18.3.3	Creating a diagnostic package with Sunrise.Workbench	311
18.3.4	Loading existing diagnosis packages from the robot controller	312
19	KUKA Service	313
19.1	Requesting support	313
19.2	KUKA Customer Support	313
Index	321

1 Introduction

1.1 Target group

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

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



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

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

- Documentation for the manipulator
- Documentation for the robot controller
- Operating and programming instructions for the System Software
- Instructions for options and accessories
- Parts catalog on storage medium

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

These warnings are relevant to safety and **must** be observed.



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



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



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



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:



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

Notes

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 Technology Group is a trademark of Beckhoff Automation GmbH.

1.5 Terms used

Term	Description
AMF	Atomic Monitoring Function Smallest unit of a monitoring function
API	Application Programming Interface Interface for programming applications.
Application server	The application server manages the robot applications and executes them in its Java Virtual Machine. The robot applications access Sunrise via the RoboticsAPI and thus command the real-time controller. Services for integration of the smartHMI operator control software are also managed in the application server.
Frame	A frame is a 3-dimensional coordinate system that is described by its position and orientation relative to a reference system. Points in space can be easily defined using frames. Frames are often arranged hierarchically in a tree structure.
Javadoc	Javadoc is a documentation generated from specific Java comments.
JRE	Java Runtime Environment Runtime environment of the Java programming language
CRR	Controlled Robot Retraction CRR is an operating mode that is available if a collision has been detected, or if the robot has violated a safely monitored space, a safely monitored Cartesian velocity limit or a safely monitored force or torque limit and is stopped by the safety controller. In CRR mode, the robot can be jogged and moved back to a position in which the monitoring function that triggered the stop is no longer violated.
KUKA smartHMI	KUKA smart human-machine interface Name of the graphical user interface of the robot controller
KUKA smartPAD	The smartPAD is the hand-held control panel for the robot cell (station). It has all the operator control and display functions required for operation of the station.
KUKA Sunrise Cabinet	Control hardware for operating the LBR iiwa industrial robot
KUKA Sunrise.OS	KUKA Sunrise.Operating System System software for industrial robots which are operated with the robot controller KUKA Sunrise Cabinet
HRC	Human-robot cooperation

Term	Description
PSM	Permanent Safety Monitoring Safety monitoring functions which are permanently active
RoboticsAPI	Java programming interface for KUKA robots RoboticsAPI is an object-oriented Java interface for controlling robots and peripheral devices.
TCP	Tool Center Point The TCP is the working point of a tool. Multiple working points can be defined for a tool.

2 Product description

2.1 Overview of the robot system

A robot system (**>>>** Fig. 2-1) comprises all the assemblies of an industrial robot, including the manipulator (mechanical system and electrical installations), controller, connecting cables, end effector (tool) and other equipment.

The industrial robot consists of the following components:

- Manipulator
- KUKA Sunrise Cabinet robot controller
- KUKA smartPAD control panel
- Connecting cables
- Software
- Options, accessories



Fig. 2-1: Description of lightweight robot

- 1 Connecting cable to the smartPAD
- 2 KUKA smartPAD control panel
- 3 Manipulator
- 4 Connecting cable
- 5 KUKA Sunrise Cabinet robot controller

2.2 Overview of the software components

The following software components are used:

- KUKA Sunrise.OS 1.1
- KUKA Sunrise.Workbench 1.1
- WorkVisual 3.0

2.3 Overview of KUKA Sunrise.OS

Description	With KUKA Sunrise.OS, the operator control and programming of the robot system are strictly separated.
	<ul style="list-style-type: none"> ■ A station currently consists of a robot controller, a manipulator and further devices. ■ A station may carry out multiple applications (tasks). ■ Robot applications are programmed with KUKA Sunrise.Workbench. ■ A robot cell (station) is operated using the KUKA smartPAD control panel.

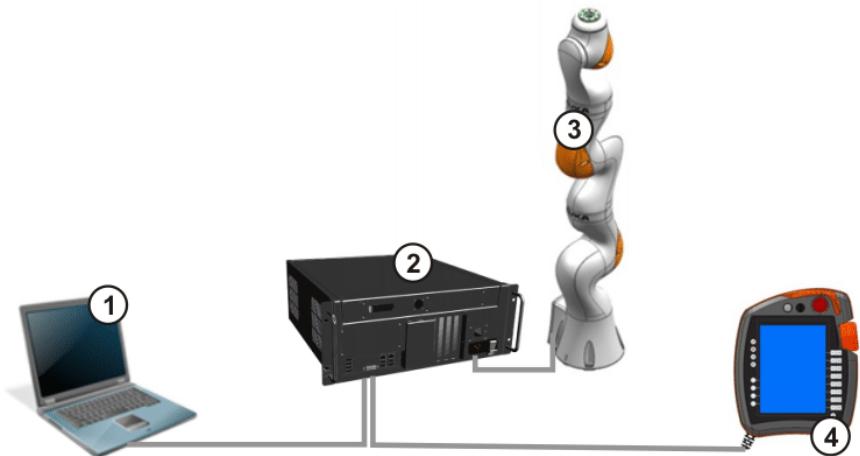


Fig. 2-2: Separation of operator control and programming

- 1 Development computer with KUKA Sunrise.Workbench
- 2 KUKA Sunrise Cabinet robot controller
- 3 Manipulator
- 4 KUKA smartPAD control panel

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

The smartPAD is only required in the start-up phase for tasks which for practical or safety reasons cannot be carried out using KUKA Sunrise.Workbench, e.g. for mastering, calibration and teaching points.

After start-up and application development, the operator can carry out simple servicing 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		✓	

Task	WorkVisual	Workbench	smartPAD
Bus configuration/diagnosis	✓		
Bus mapping	✓		
Configuring safety settings		✓	
Activating the safety configuration			✓
Programming		✓	
Debugging		✓	
Managing/editing runtime data		✓	
Teaching frames			✓
Operating mode selection			✓
Jogging			✓
Mastering			✓
Setting/polling outputs			✓
Polling inputs			✓
Starting/stopping robot applications			✓
Creating a diagnostic package		✓	✓

2.4 Overview of KUKA Sunrise.Workbench

KUKA Sunrise.Workbench is the development environment for the robot cell (station). It offers the following functionalities for start-up and application development:

- | | |
|--------------------------------|--|
| Start-up | <ul style="list-style-type: none"> ■ Installing the system software ■ Configuring the robot cell (station) ■ Editing the safety configuration ■ Creating the I/O configuration ■ Transferring the project to the robot controller |
| Application development | <ul style="list-style-type: none"> ■ Programming robot applications in Java ■ Managing projects and programs ■ Editing and managing runtime data ■ Project synchronization |

2.5 Intended use of the system software

Use The system software is intended exclusively for the operation of a KUKA LBR iiwa in conjunction with KUKA Sunrise Cabinet.

Each version of the system software may be operated exclusively in accordance with the specified system requirements.

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

- Operation of a kinematic system that is not a KUKA lightweight robot

- Operation of the system software not in accordance with the specified system requirements

3 Safety

3.1 Legal framework

3.1.1 Liability

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

Components of the industrial robot:

- Manipulator
- Robot controller
- Hand-held control panel
- Connecting cables
- Software
- Options, accessories

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

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

Safety information

Safety information cannot be held against KUKA Laboratories GmbH. Even if all safety instructions are followed, this is not a guarantee that the industrial robot will not cause personal injuries or material damage.

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

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

3.1.2 Intended use of the industrial robot

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

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

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

The user is responsible for the performance of a risk analysis. This indicates the additional safety equipment that is required, the installation of which is also the responsibility of the user.

Misuse

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

- Transportation of persons and animals
- Use as a climbing aid
- Operation outside the specified operating parameters
- Use in potentially explosive environments
- Operation without the required additional safety equipment
- Outdoor operation
- Underground operation

3.1.3 EC declaration of conformity and declaration of incorporation

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

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

Declaration of conformity

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

The robot controller is CE certified under the EMC Directive and the Low Voltage Directive.

Declaration of incorporation

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

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

3.2 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 industrial robot meet the following safety requirements:
 - **Category 3 and Performance Level d** in accordance with EN ISO 13849-1:2008
 - **SIL 2** according to EN 62061

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

- The EMERGENCY STOP device is pressed and tested for correct functioning during start-up and at least once every 6 months.
 - Non-safety-oriented functions for the protection of machines
- The non-safety-oriented functions of the industrial robot do not meet specific safety requirements:



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



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



In the case of a safety stop 1, the drives are switched off and the brakes are applied after one second at the latest. Use of the safety stop 0 is recommended in situations in which there could be persons in the danger zone of the robot, e.g. in test mode or collaborative operation.

3.2.1 Terms used

Term	Description
Axis range	Range of each axis, in degrees or millimeters, within which it may move. The axis range must be defined for each axis.
Stopping distance	Stopping distance = reaction distance + braking distance The stopping distance is part of the danger zone.
Workspace	The manipulator is allowed to move within its workspace. The workspace is derived from the individual axis ranges.
Automatic (AUT)	Operating mode for program execution. The manipulator moves at the programmed velocity.
Operator (User)	The user of the industrial robot can be the management, employer or delegated person responsible for use of the industrial robot.
Danger zone	The danger zone consists of the workspace and the stopping distances.
Service life	The service life of a safety-relevant component begins at the time of delivery of the component to the customer. The service life is not affected by whether the component is used in a robot controller or elsewhere or not, as safety-relevant components are also subject to aging during storage.
CRR	Controlled Robot Retraction CRR is an operating mode that is available if a collision has been detected, or if the robot has violated a safely monitored space, a safely monitored Cartesian velocity limit or a safely monitored force or torque limit and is stopped by the safety controller. In CRR mode, the robot can be jogged and moved back to a position in which the monitoring function that triggered the stop is no longer violated.
KUKA smartPAD	The smartPAD is the hand-held control panel for the robot cell (station). The smartPAD has all the operator control and display functions required for operation.
Manipulator	The robot arm and the associated electrical installations

Term	Description
Safety zone	The manipulator is not allowed to move within the safety zone. The safety zone is the area 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 or Stop category 1.</p> <p>Note: In this document, a safety stop of Stop category 0 is referred to as safety stop 0 and a safety stop of Stop category 1 as safety stop 1.</p>
Stop category 0	The drives are deactivated immediately and the brakes are applied. The manipulator is stopped with path-oriented braking.
Stop category 1	The manipulator is braked and stays on the programmed path. The drives are deactivated after 1 s and the brakes are applied.
System integrator (plant integrator)	System integrators are people who safely integrate the industrial robot into a complete system and commission it.
T1	Test mode, Manual Reduced Velocity (<= 250 mm/s)
T2	Test mode, Manual High Velocity (> 250 mm/s permissible)

3.2.2 Personnel

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

- User
- Personnel



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

User

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

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

Personnel

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

Personnel includes:

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



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

System integrator

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

The system integrator is responsible for the following tasks:

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

Operator

The operator must meet the following preconditions:

- The operator must be trained for the work to be carried out.
- Work on the industrial robot 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 manipulator may only be carried out by KUKA Laboratories GmbH.

3.2.3 Workspace, safety zone and danger zone

Working zones 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 configuring safe axis range limitations is contained in the "Safety configuration" chapter of the operating and programming instructions. ([>>> 13 "Safety configuration" Page 145](#))

The danger zone consists of the workspace and the stopping distances of the manipulator. In the event of a stop, the manipulator 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 in accordance with EN ISO 10218 must be met. There must be no shearing or crushing hazards at the loading and transfer areas.

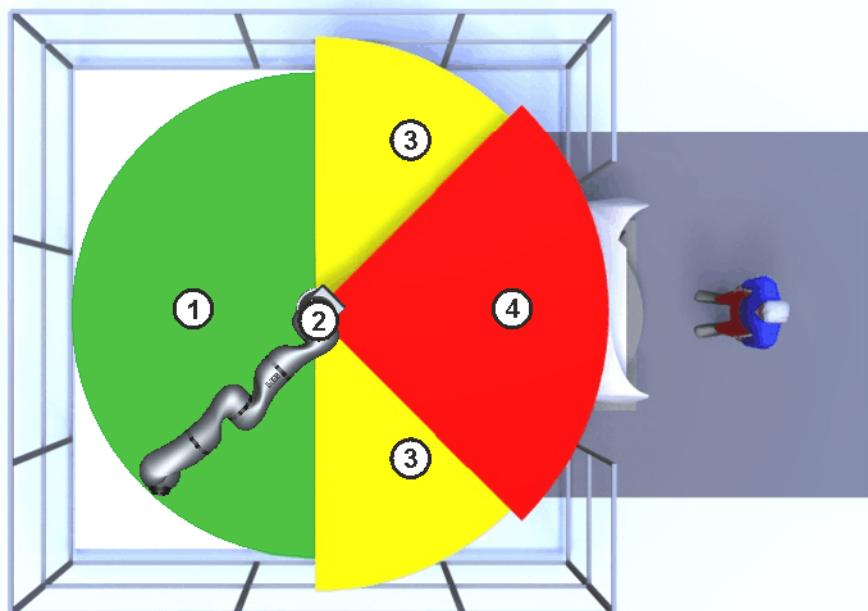


Fig. 3-1: Example: axis range A1

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

3.2.4 Safety-oriented functions

The following safety-oriented functions are present and permanently defined in the industrial robot:

- EMERGENCY STOP device
- Enabling device
- Locking of the operating mode (by means of a keyswitch)

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

- Operator safety (= connection for the guard interlock)
- External EMERGENCY STOP device
- External safety stop 1

Other safety-oriented functions that are not present by default 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. (>>> 13 "Safety configuration" Page 145)

The default configuration of the preconfigured safety functions is described in the following sections on safety.

3.2.4.1 EMERGENCY STOP device

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

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

- The manipulator stops with a safety stop. ([>>> 3.2.5 "Triggers for safety-oriented stop reactions" Page 29](#))

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



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

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

([>>> 3.2.4.4 "External EMERGENCY STOP device" Page 28](#))

3.2.4.2 Enabling device

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

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

- Not pressed
- Center position
- Panic position

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

- Releasing the enabling switch triggers a safety stop. ([>>> 3.2.5 "Triggers for safety-oriented stop reactions" Page 29](#))
- Pressing the enabling switch down fully (panic position) triggers a safety stop 0.
- It is possible to hold 2 enabling switches in the center position simultaneously for several seconds. This makes it possible to adjust grip from one enabling switch to another one. If 2 enabling switches are held simultaneously in the center position for longer than 15 seconds, this triggers a safety stop 1.

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

- Press the enabling switch down fully
- Actuate the EMERGENCY STOP device
- Release the Start/pause key



The enabling switches must not be held down by adhesive tape or other means or tampered with in any other way.

Death, injuries or damage to property may result.

3.2.4.3 Operator safety

The operator safety signal is used for interlocking physical safeguards, e.g. safety gates. In the default configuration, automatic operation is not possible without this signal. Alternatively, the requirements for collaborative operation in accordance with EN ISO 10218 must be met.

Reaction of the industrial robot in the event of a loss of signal during automatic operation, e.g. safety gate is opened (default configuration):

- The manipulator stops with a safety stop 1.

By default, operator safety is not active in the modes T1 (Manual Reduced Velocity) and CRR, i.e. the signal is not evaluated. Operator safety is active in mode T2 (Manual High Velocity).



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 permissible.
- Failure to observe this may result in death to persons, severe injuries or considerable damage to property.

3.2.4.4 External EMERGENCY STOP device

There must be EMERGENCY STOP devices available at every operator station that can initiate a robot motion or other potentially hazardous situation. The system integrator is responsible for ensuring this.

Reaction of the industrial robot if the external EMERGENCY STOP device is pressed (default configuration):

- The manipulator stops with a safety stop 0 (T1, CRR) or with a safety stop 1 (T2, AUT).

Multiple external EMERGENCY STOP devices can be connected via the safety interface of the robot controller. External EMERGENCY STOP devices are not included in the scope of supply of the industrial robot.

3.2.4.5 External safety stop 1

Safety stop 1 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 manipulator can be moved again. No acknowledgement is required.

3.2.4.6 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 industrial robot.

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 industrial robot.

3.2.4.7 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 manipulator can be moved again. No acknowledgement is required.

3.2.5 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 KUKA Sunrise 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. ([>>> 13 "Safety configuration" Page 145](#))

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	
Enabling switch released	Safety stop 1	-
Enabling switch pressed fully down (panic position)	Safety stop 0	-
Local E-STOP pressed	Safety stop 1	
Error in safety controller	Safety stop 0	

User-specific triggers

The robot controller is shipped with a safety configuration that is active on initial start-up. 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

Trigger	T1, CRR	T2, AUT
Enabling switch released	Safety stop 0	-
Local E-STOP pressed	Safety stop 0	-

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).



When the Sunrise project is transferred to the robot controller, the factory-set safety configuration is overwritten by the project-specific safety configuration. This makes it necessary for the safety configuration to be activated.

Further information about activating the safety configuration is contained in the "Safety configuration" chapter of the operating and programming instructions. ([>>> 13 "Safety configuration" Page 145](#))

Trigger	T1, CRR	T2, AUT
Safety gate opened (operator safety)	-	Safety stop 1
Enabling switch released	Safety stop 0	-
Local E-STOP pressed	Safety stop 0	-
External E-STOP pressed	Safety stop 0	Safety stop 1
External safety stop	Safety stop 1	

3.2.6 Non-safety-oriented functions

3.2.6.1 Mode selection

The industrial robot 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
T2	Testing of programs Only possible with safety gate closed	<ul style="list-style-type: none"> ■ Program verification: Programmed velocity ■ Jog mode: not possible

Operating mode	Use	Velocities
AUT	Automatic execution of programs For industrial robots with and without higher-level controllers	<ul style="list-style-type: none"> ■ Program mode: Programmed velocity ■ Jog mode: not possible
CRR	Motion taking the industrial robot out of a violated space and retraction following collisions CRR is an operating mode that is available if a collision has been detected, or if the robot has violated a safely monitored space, a safely monitored Cartesian velocity limit or a safely monitored force or torque limit and is stopped by the safety controller.	<ul style="list-style-type: none"> ■ Program mode: Reduced programmed velocity, maximum 250 mm/s ■ Jog mode: Jog velocity, maximum 250 mm/s

3.2.6.2 Software limit switches

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

3.3 Additional protective equipment

3.3.1 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 that it is necessary to hold down an enabling switch and the Start/pause key in order to execute a program.

- Releasing the enabling switch on the smartPAD triggers a safety stop.
(>>> 3.2.5 "Triggers for safety-oriented stop reactions" Page 29)
- Pressing the enabling switch on the smartPAD fully down triggers a safety stop 0.
- Releasing the Start/pause key triggers a stop of Stop category 1.

3.3.2 Labeling on the industrial robot

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

Labeling on the industrial robot consists of:

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



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

3.3.3 External safeguards

The access of persons to the danger zone of the industrial robot must be prevented by means of safeguards. Alternatively, the requirements for collaborative operation in accordance with EN ISO 10218 must be met. It is the responsibility of the system integrator to ensure this.

Physical safeguards must meet the following requirements:

- They meet the requirements of EN 953.
- 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 risk situation: the safety gate is additionally safeguarded by means of a locking mechanism that only allows the gate to be opened if the manipulator is safely at a standstill.
- The device for setting the signal for operator 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 953.

Other safety equipment

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

3.4 Safety measures

3.4.1 General safety measures

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

It is important to be prepared for possible movements of the industrial robot even after the robot controller has been switched off and locked out. Incorrect installation (e.g. overload) or mechanical defects (e.g. brake defect) can cause the manipulator to sag. If work is to be carried out on a switched-off industrial robot, the manipulator 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 manipulator must be secured by appropriate means.



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



WARNING Standing underneath the robot arm can cause death or serious injuries. Especially if the industrial robot is moving objects that can become detached (e.g. from a gripper). For this reason, standing underneath the robot arm is prohibited!

smartPAD	The user must ensure that the industrial robot is only operated with the smart-PAD by authorized persons.
Modifications	<p>After modifications to the industrial robot, checks must be carried out to ensure the required safety level. The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.</p> <p>New or modified programs must always be tested first in Manual Reduced Velocity mode (T1).</p> <p>After modifications to the industrial robot, existing programs must always be tested first in Manual Reduced Velocity mode (T1). This applies to all components of the industrial robot and includes modifications to the software and configuration settings.</p> <p>Following reconnection of the industrial robot, the robot controller must be rebooted in order to ensure that the correct machine data are loaded.</p>
Faults	<p>The following tasks must be carried out in the case of faults in the industrial robot:</p> <ul style="list-style-type: none"> ■ Switch off the robot controller and secure it (e.g. with a padlock) to prevent unauthorized persons from switching it on again. ■ Indicate the fault by means of a label with a corresponding warning (tag-out). ■ Keep a record of the faults. ■ Eliminate the fault and carry out a function test.

3.4.2 Transportation

Manipulator	The prescribed transport position of the manipulator must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the robot. Avoid vibrations and impacts during transportation in order to prevent damage to the manipulator.
Robot controller	The prescribed transport position of the robot controller must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the robot controller. Avoid vibrations and impacts during transportation in order to prevent damage to the robot controller.

3.4.3 Start-up and recommissioning

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

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



The password to activate the safety configuration must be changed before start-up. This password may only be communicated to trained safety maintenance technicians who are authorized to activate the safety configuration.

(>>> 13.5.3 "Changing the password for activating the safety configuration" Page 156)



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



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



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

Function test

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

General test:

It must be ensured that:

- The industrial robot is correctly installed and fastened in accordance with the specifications in the documentation.
- There are no foreign bodies or loose parts on the industrial robot.
- All required safety equipment is correctly installed and operational.
- The power supply ratings of the industrial robot correspond to the local supply voltage and mains type.
- The ground conductor and the equipotential bonding cable are sufficiently rated and correctly connected.
- The connecting cables are correctly connected and the connectors are locked.

Test of the safety functions:

A function test must be carried out for all the safety-oriented functions to ensure that they are working correctly:

(>>> 13.9 "Safety acceptance overview" Page 175)



In the case of incomplete start-up of the system, additional substitute measures for minimizing risk must be taken and documented, e.g. installation of a safety fence, attachment of a warning sign, locking of the main switch, etc. 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:

The KUKA LBR iiwa (all variants) must undergo a brake test on a regular basis in order to check whether the brakes on all axes apply sufficient braking torque.

(>>> 8 "Brake test" Page 95)

The brake test must be carried out during start-up and recommissioning of the industrial robot.

Irrespective of the period of operation and type of application, the brake test must be performed daily during operation.

Machine data

It must be ensured that the rating plate on the robot controller has the same machine data as those entered in the declaration of incorporation. The machine data on the rating plate of the manipulator must be entered during start-up.

3.4.4 Manual mode

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

- Jog mode
- Teaching
- Program verification

The following must be taken into consideration in manual mode:

- New or modified programs must always be tested first in Manual Reduced Velocity mode (T1).
- The manipulator 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 industrial robot motion, nor must they lead to short-circuits or be liable to fall off.
- All setup work must be carried out, where possible, from outside the safeguarded area.

If the setup work has to be carried out inside the safeguarded area, the following must be taken into consideration:

In Manual Reduced Velocity mode (T1):

- If it can be avoided, there must be no other persons inside the safeguarded area.

If it is necessary for there to be several persons inside the safeguarded area, the following must be observed:

- Each person must have an enabling device.
- All persons must have an unimpeded view of the industrial robot.
- Eye-contact between all persons must be possible at all times.
- The operator must be so positioned that he can see into the danger area and get out of harm's way.

In Manual High Velocity mode (T2):

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

3.4.5 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. Alternatively, the requirements for collaborative operation in accordance with EN ISO 10218 have been met.
- The defined working procedures are adhered to.

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

3.4.6 Maintenance and repair

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

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

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

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

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

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

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

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

Robot controller

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

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

Voltages in excess of 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 industrial robot in this time.

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

3.4.7 Decommissioning, storage and disposal

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

3.4.8 Safety measures for “single point of control”

Overview

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

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; this is the responsibility of the user of the system.

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

T1, T2, CRR

In modes T1, T2 and CRR, a robot motion can only be initiated if an enabling switch on the smartPAD 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.

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

Safety measures:

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

3.5 Applied norms and regulations

Name	Definition	Edition
2006/42/EC	Machinery Directive: Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast)	2006
2004/108/EC	EMC Directive: Directive 2004/108/EC of the European Parliament and of the Council of 15 December 2004 on the approximation of the laws of the Member States relating to electromagnetic compatibility and repealing Directive 89/336/EEC	2004
EN ISO 13850	Safety of machinery: Emergency stop - Principles for design	2008
EN ISO 13849-1	Safety of machinery: Safety-related parts of control systems - Part 1: General principles of design	2008
EN ISO 13849-2	Safety of machinery: Safety-related parts of control systems - Part 2: Validation	2012
EN ISO 12100	Safety of machinery: General principles of design, risk assessment and risk reduction	2010
EN ISO 10218-1	Industrial robots: Safety Note: Content equivalent to ANSI/RIA R.15.06-2012, Part 1	2011
EN 614-1	Safety of machinery: Ergonomic design principles - Part 1: Terms and general principles	2009
EN 61000-6-2	Electromagnetic compatibility (EMC): Part 6-2: Generic standards; Immunity for industrial environments	2005
EN 61000-6-4 + A1	Electromagnetic compatibility (EMC): Part 6-4: Generic standards; Emission standard for industrial environments	2011

EN 60204-1 + A1**Safety of machinery:**

2009

Electrical equipment of machines - Part 1: General requirements

4 Installing KUKA Sunrise.Workbench

4.1 PC system requirements

Hardware	Minimum requirements
	<ul style="list-style-type: none"> ■ PC with Pentium IV processor, min. 1500 MHz ■ 1 GB RAM ■ 1 GB free hard disk space ■ DirectX8-compatible graphics card with a resolution of 1024x768 pixels
Software	Recommended specifications
	<ul style="list-style-type: none"> ■ PC with Pentium IV processor and 2500 MHz ■ 2 GB RAM ■ DirectX8-compatible graphics card with a resolution of 1280x1024 pixels

The following software is required for bus configuration:

- WorkVisual 3.0

4.2 Installing KUKA Sunrise.Workbench

Precondition	<ul style="list-style-type: none"> ■ Local administrator rights
Procedure	<ol style="list-style-type: none"> 1. Start installation via the file Setup.exe. 2. Select the language and confirm with OK. The installation wizard Sunrise Workbench Setup opens. 3. Press Next > to switch to the next page. 4. Accept the license agreement. Press Next > to switch to the next page. 5. The default folder for installation is specified in the Folder box. A different installation folder can be selected by pressing Browse.... Press Next > to switch to the next page. 6. If no desktop link for Sunrise.Workbench should be created: deactivate the Desktop option (uncheck the box). 7. Press Next > to switch to the next page and click on Install. Sunrise.Workbench is installed. 8. Once installation is completed, click on Finish to close the installation wizard. Sunrise.Workbench is then started directly. If this is not desired, deactivate the Launch Sunrise Workbench option (uncheck the box).

4.3 Uninstalling KUKA Sunrise.Workbench

 It is advisable to archive all relevant data before uninstalling a software package.

Precondition	<ul style="list-style-type: none"> ■ Local administrator rights
Procedure	<ol style="list-style-type: none"> 1. Display the list of installed programs in Windows.

Alternative procedure

- Windows 7: In the Windows Start menu, select **Control Panel (> Programs)** > **Programs and Functions**.
 - Windows XP: In the Windows Start menu, select **Settings > Control Panel > Software**.
2. Select the entry **Sunrise Workbench**. Click on **Uninstall** and answer the request for confirmation with **Yes**.
- In the Windows Start menu, open the installation directory specified during installation and double-click on **Uninstall**. (Windows 7: single click)

5 Operation of KUKA Sunrise.Workbench

5.1 Starting KUKA 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 KUKA Sunrise.Workbench" Page 43)
- Click on the **New Sunrise Project** button to create a new Sunrise project directly. The project creation wizard opens.
(>>> 5.3 "Creating a Sunrise project with a template" Page 46)

5.2 Overview of the user interface of KUKA 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.

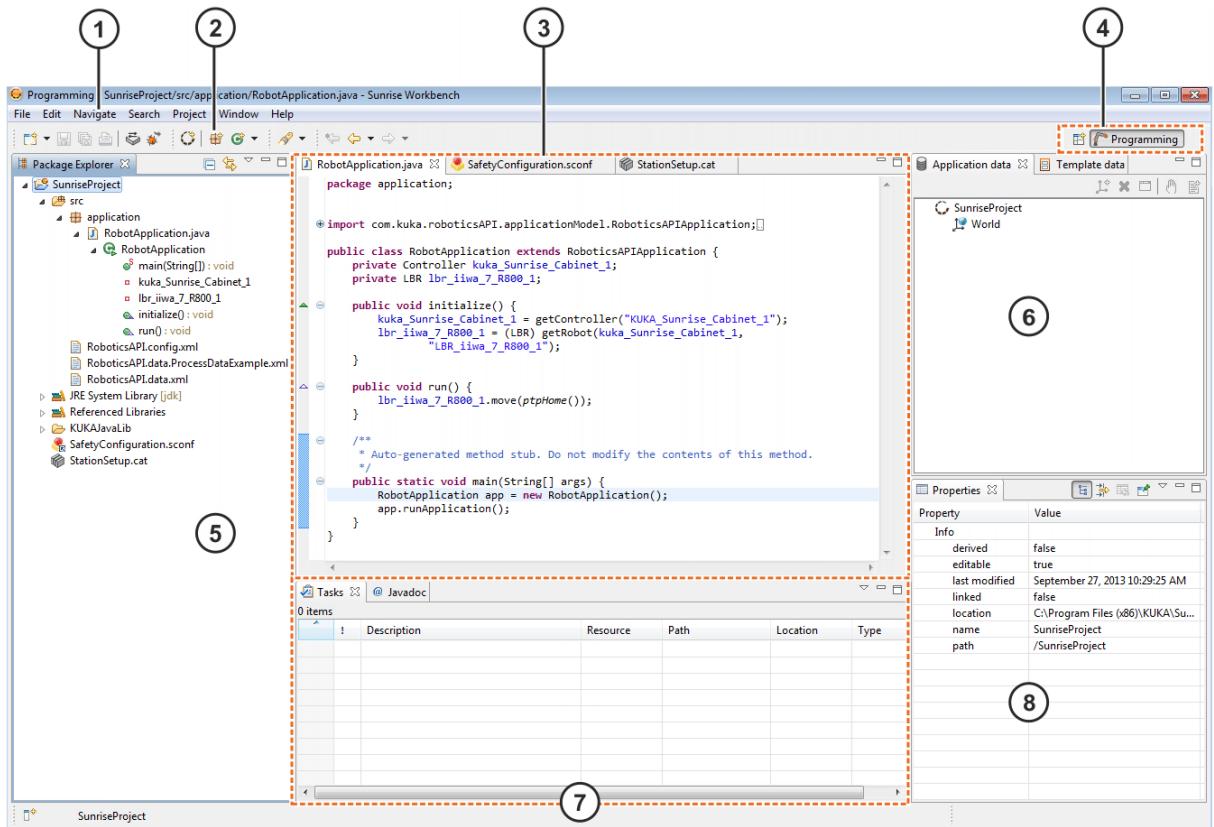


Fig. 5-1: Overview of user interface – “Programming” perspective

Item	Description
1	Menu bar
2	Toolbars (>>> 5.2.3 "Toolbars" Page 46)
3	Editor area Opened files, e.g. robot applications, are displayed and edited here.
4	Perspective selection Here it is possible to switch between various previously-used perspectives by clicking on the name of the appropriate perspective or on the Open Perspective icon. (>>> 5.2.2 "Displaying different perspectives on the user interface" Page 45)
5	Package Explorer view This view contains the projects created and their corresponding files.
6	Application data and Template data views <ul style="list-style-type: none"> ■ Application data: This view displays the frames created for a project in a tree structure. ■ Template data: This view displays the geometrical objects, tools and workpieces created for a project in a tree structure.

Item	Description
7	Tasks and Javadoc views <ul style="list-style-type: none"> ■ Tasks: Displays tasks which a user has created ■ Javadoc: Displays Javadoc information about the selected elements of a Java application
8	Properties view <p>If an object, e.g. a project, frame or tool, is selected in a view, its properties are displayed here.</p>

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 Displaying different perspectives on the user interface

- 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	Type of work
Programming	This perspective has views suitable for editing Sunrise projects. For example, for 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.

5.2.3 Toolbars

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

Programming	Icon	Name / description
		New Opens the wizard for creating new documents.
		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.
		Sunrise Project Opens the wizard for creating a new Sunrise project.
		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.
		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 project creation wizard opens.
2. Enter the IP address of the robot controller to be created for the project in the **Controller IP Address** box.



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

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

3. Retain the **Create new project (offline)** setting. Press **Next >** to switch to the next page.

4. Enter a name for the project in the **Project name** box.

5. The default directory for projects is given in the **Location** box.

A different directory can be selected if desired: To do this, deactivate the **Use default location** option (remove check mark) and select **Browse....**. Press **Next >** to switch to the next page.

6. Select the robot for the station configuration, e.g. LBR iiwa 7 R800, from the **Topology template** list. Press **Next >** to switch to the next page.

7. In order to complete the station configuration, select the media adapter module with which the robot is equipped (see identification plate of the robot).

8. By default, the direction of installation of the 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 ° (angle A):** 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 ° (angle B):** 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 ° (angle C):** 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.

(>>> 6.6 "Coordinate systems" Page 69)

9. Press **Next >** to switch to the next page. A summary of information on the project is displayed.

Deactivate the **Create Application** option (remove check mark) and click on **Finish**. The project is created and added to the **Package Explorer**.

If the **Create Application** option remains active (check mark set), the wizard for application creation opens. The first robot application for this newly created project can then be created directly.

(>>> 5.4.2 "Creating a robot application with a package" Page 49)

Description

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

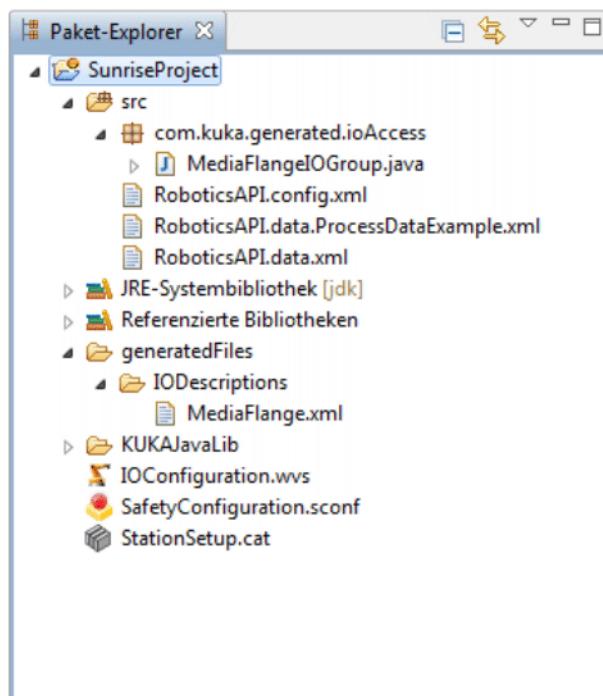


Fig. 5-2: Overview of the project structure

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 Java package com.kuka.generated.ioAccess contains the Java class MediaFlangeIOGroup.java. The class already contains the methods required for programming in order to access the inputs/outputs of the media adapter module.</p> <p>Further information on the use of inputs/outputs in programming can be found here: (>>> 15.10 "Inputs/outputs" Page 226)</p> <p>The source folder also contains various XML files in which, in addition to the configuration data, the runtime data are saved; for example, the frames and tools created by the user.</p> <p>The XML files can be displayed but not edited.</p>
JRE System Library	<p>Java Runtime Environment system library</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. By default, the robot-specific Java class libraries are automatically added when a Sunrise project is created. The user has the option of adding further libraries.</p>
generatedFiles	<p>Folder with subfolder IODescriptions</p> <p>The data for the inputs/outputs configured for the media adapter module are saved in an XML file.</p> <p>The XML file can be displayed but not edited.</p>
KUKAJavaLib	Folder with special libraries required for robot programming.

Element	Description
IOConfiguration.wvs	I/O configuration for the media adapter module The I/O configuration contains the complete bus structure of the media adapter module, including the I/O mapping. The I/O configuration can be opened, edited and re-exported into the Sunrise project in WorkVisual. (>>> 11 "Bus configuration" Page 129)
SafetyConfiguration.sconf	The file contains the safety functions preconfigured by KUKA. The configuration can be displayed and edited. (>>> 13 "Safety configuration" Page 145)
StationSetup.cat	The file contains the station configuration for the station (controller) selected when the project was created. The configuration can be displayed and edited. The system software can be installed on the robot controller via the station configuration. (>>> 10 "Station configuration and installation" Page 127)

5.4 Creating a new robot application

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

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

5.4.1 Creating a new Java package

- Procedure**
1. Select the project in the **Package Explorer**.
 2. Select the menu sequence **File > New > Package**. The **New Java Package** window opens.
 3. Enter a name for the package in the **Name** box.
 4. Click on **Finish**. The package is created and added to the "src" folder for the project.
- The package does not yet contain any files. An empty package is indicated by a white package icon. As soon as a package contains files, the icon turns brown.

5.4.2 Creating a robot application with a package

- Procedure**
1. Select the project in the **Package Explorer**.
 2. Select the menu sequence **File > New > Robotics Application**. The **New robotics application** window opens.
 3. In the **Package** box, enter the name of the package in which the application should be created.
 4. Enter a name for the package in the **Name** box.
 5. Click on **Finish**. The application and package are created and inserted into the project.
- The *Name.java* application is opened in the editor area.

5.4.3 Creating a robot application for an existing package

- | | |
|------------------|---|
| Procedure | <ol style="list-style-type: none">1. Select the package in the Package Explorer.2. Select the menu sequence File > New > Robotics Application. The New robotics application window opens.3. Enter a name for the package in the Name box.4. Click on Finish. The application is created and inserted into the package. The <i>Name.java</i> application is opened in the editor area. |
|------------------|---|

5.5 Creating a new background task

Background tasks 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 tasks are described here:
(>>> 16 "Background tasks" Page 281)

The following properties are defined when the task is created:

- Start type of the task
 - **Automatic**
The task is automatically started after the robot controller has booted (default).
 - **Manual**
The task must be started manually via the smartPAD. (This function is not yet supported.)
- Task template
 - **Cyclic background task**
Template for tasks that are to be executed cyclically (default)
 - **Non-cyclic background task**
Template for tasks that are to be executed once

5.5.1 Creating a background task with a package

- | | |
|------------------|--|
| Procedure | <ol style="list-style-type: none">1. Select the project in the Package Explorer.2. Select the menu sequence File > New > Background task. The New background task window is opened.3. In the Package box, enter the name of the package in which the task is to be created.4. Enter a name for the task in the Name box.5. Click on Next > and select the start type of the task.6. Click on Next > and select the task template.7. Click on Finish. The task and package are created and inserted into the project.
The <i>Name.java</i> task is opened in the editor area. |
|------------------|--|

5.5.2 Creating a background task for an existing package

- | | |
|------------------|--|
| Procedure | <ol style="list-style-type: none">1. Select the package in the Package Explorer.2. Select the menu sequence File > New > Background task. The New background task window is opened.3. Enter a name for the task in the Name box.4. Click on Next > and select the start type of the task. |
|------------------|--|

5. Click on **Next >** and select the task template.
6. Click on **Finish**. The task is created and inserted into the package.
The *Name.java* task is opened in the editor area.

5.6 Workspace

The directory in which the created projects and user-defined settings for 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 Sunrise.Workbench and to switch between them.

5.6.1 Creating a new workspace

- | | |
|------------------|--|
| Procedure | <ol style="list-style-type: none"> 1. Select the menu sequence File > Switch Workspace > Other.... The Workspace Launcher window opens. 2. In the Workspace box, manually enter the path to the new project directory. <p>Alternative:</p> <ul style="list-style-type: none"> ■ 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. Click on OK to confirm. <p>The path to the new project directory is inserted in the Workspace box.</p> <ol style="list-style-type: none"> 3. Click on OK to confirm the new workspace. Sunrise.Workbench restarts and the welcome screen opens. |
|------------------|--|

5.6.2 Switching to an existing workspace

- | | |
|---------------------|--|
| Precondition | <ul style="list-style-type: none"> ■ Other workspaces are available. |
| Procedure | <ol style="list-style-type: none"> 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. |

5.6.3 Switching between the most recently opened workspaces

- | | |
|---------------------|---|
| Precondition | <ul style="list-style-type: none"> ■ Other workspaces are available. |
| Procedure | <ol style="list-style-type: none"> 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. Sunrise.Workbench restarts and opens the selected workspace. |

5.6.4 Archiving projects

- | | |
|------------------|--|
| Procedure | <ol style="list-style-type: none"> 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 created.

5.6.5 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 (check mark must be set).
6. Click on **Finish**. The selected projects are loaded.

5.6.6 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.

5.7 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.

5.7.1 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.

5.7.1.1 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. By default, 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 your 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.

5.7.2 Referencing Java projects

- | | |
|---------------------|--|
| Precondition | <ul style="list-style-type: none">■ 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 | <ol style="list-style-type: none">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. |

5.7.3 Canceling the reference to Java projects

- | | |
|--------------------|--|
| Description | References to inadvertently added projects or projects that are not required (any longer) can be removed. |
| Procedure | <ol style="list-style-type: none">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. |

5.8 Renaming an element in the “Package Explorer”

- | | |
|--------------------|--|
| Description | In the Package Explorer view, the names of inserted elements can be changed, e.g. the names of Sunrise projects, Java packages or robot applications. |
| Procedure | <ol style="list-style-type: none">1. Right-click on the element. Select Refactoring > Rename... in the context menu. The ... rename window opens. (The exact name of the window depends on the selected element type.)2. Enter the desired name in the New name: box. Click on Finish.3. Possible conflicts are indicated before the renaming is completed. After acknowledging and checking these, click on Finish once more. |

5.9 Removing elements in the “Package Explorer”

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

5.9.1 Removing elements from a project

Description If elements of a project are removed in the **Package Explorer**, these are permanently deleted. This means that they are deleted from the project folder on the data storage medium and cannot be restored.

Procedure

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

5.9.2 Removing projects

Description The following options are available for removing a project in the **Package Explorer**:

- The project is only removed from the **Package Explorer** and is retained in the project folder on the data storage medium. (Default setting)
- The project is deleted from the project folder on the data storage medium and cannot be restored.

Procedure

1. Right-click on the project and select **Delete** from the context menu.
2. A request for confirmation is displayed, asking if the project should really be deleted from the workspace.
 - If the project should be deleted from the data storage medium, activate the **Delete project content on disk ...** option (set check mark).
 - If the project should remain on the data storage medium, do not activate the option. (Default setting)
3. Answer the request for confirmation with **OK**. The element is deleted.

6 Operating the KUKA smartPAD

6.1 KUKA smartPAD control panel

6.1.1 Front view

Function

The smartPAD is the hand-held control panel for the industrial robot. The smartPAD has all the operator control and display functions required for operation.

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

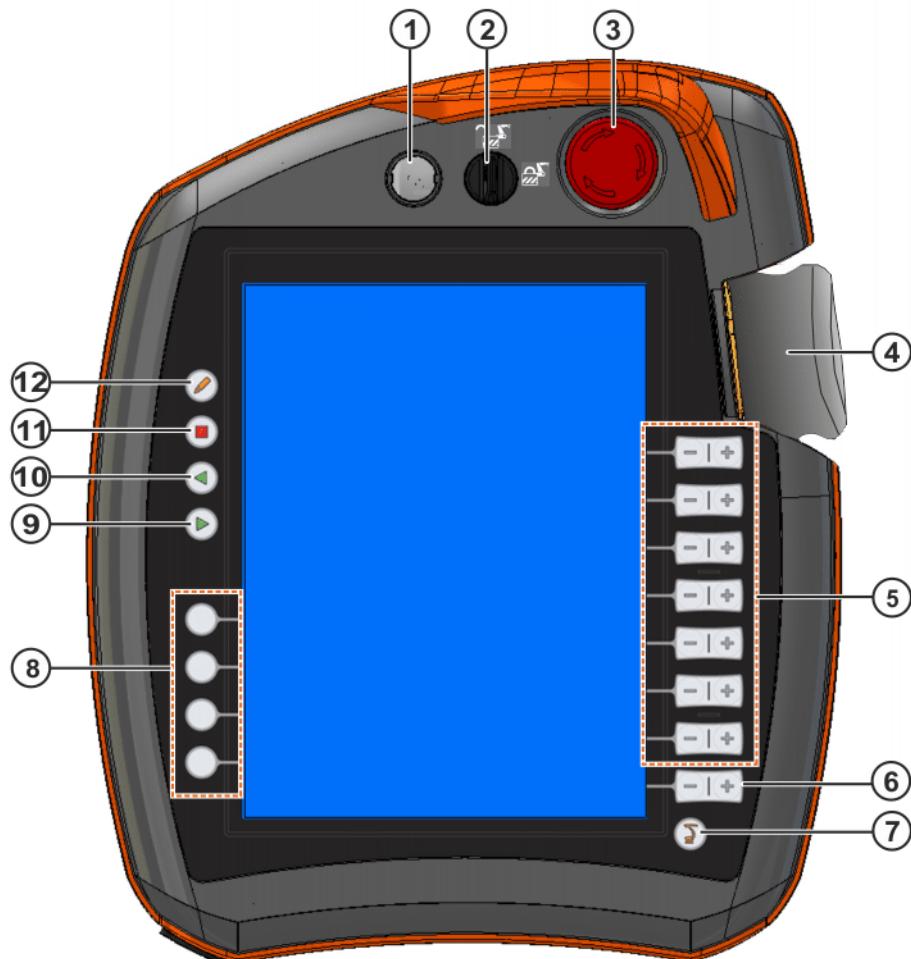


Fig. 6-1: KUKA smartPAD, front view

Item	Description
1	Button for disconnecting the smartPAD Currently without function
2	Keyswitch The connection manager is called by means of the keyswitch. The connection manager is used to change the operating mode. (>>> 6.5 "Changing the operating mode" Page 68)

Item	Description
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 Currently without function
5	Jog keys The jog keys are used to move the robot manually. (>>> 6.7 "Jogging the robot" Page 70)
6	Key for setting the jog override
7	Main menu key The main menu key shows and hides the main menu on the smartHMI.
8	User-defined status keys Currently without function
9	Start/pause key The Start/pause key is used to start a program or to pause a program during execution. The Start/pause key is also used to manually address frames and to move the robot back onto the path. (>>> 6.10 "Program execution" Page 80) (>>> 6.9.4 "Manually addressing frames" Page 80) (>>> 6.10.6 "Repositioning the robot after leaving the path" Page 84)
10	Start backwards key Currently without function
11	STOP key The STOP key is used to fully abort and reset a paused application.
12	Keyboard key Currently without function



The following applies to the jog keys, the user-defined status keys as well as the Start/pause, 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 view

Overview



Fig. 6-2: KUKA smartPAD, rear view

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

Description

Element	Description
Identification plate	Identification plate
Start/pause key	The Start/pause key is used to start a program or to pause a program during execution. The Start/pause key is also used to manually address frames and to move the robot back onto the path.

Element	Description
Enabling switch	<p>The enabling switch has 3 positions:</p> <ul style="list-style-type: none">■ Not pressed■ Center position■ Panic position <p>The enabling switch must be held in the center position in operating modes T1, T2 and CRR in order to be able to jog the manipulator.</p> <p>By default, the enabling switch has no function in Automatic mode.</p>
USB connection	The USB connection is used e.g. for archiving data. Only for FAT32-formatted USB sticks.

6.2 Switching the robot controller on/off

6.2.1 Switching on the robot controller and starting the system software

Procedure

- Turn the main switch on the robot controller to the “I” position.
The system software starts automatically.

The robot controller is ready for operation when the status display for the boot state of the robot controller lights up green (station view / **KUKA_Sunrise_Cabinet** tile).

6.2.2 Switching off the robot controller

Procedure

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

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.

6.3 KUKA smartHMI user interface

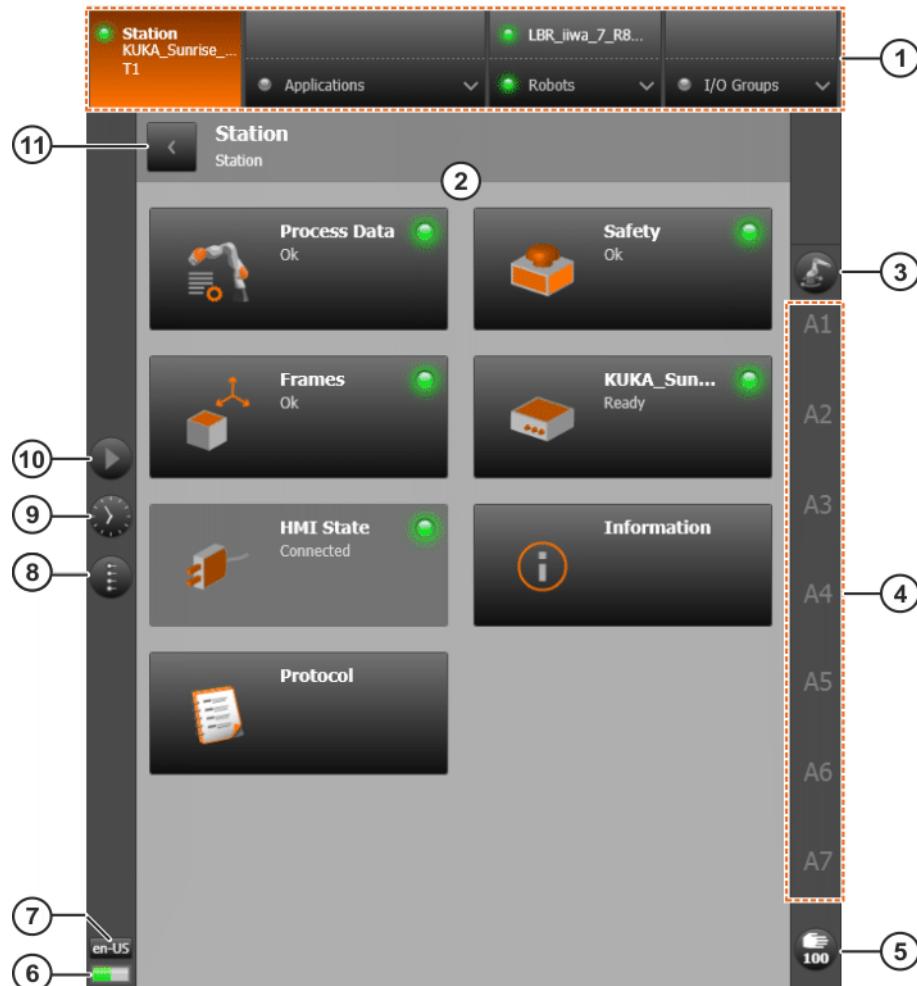


Fig. 6-3: KUKA smartHMI user interface

Item	Description
1	Navigation bar: Main menu and status display (>>> 6.3.1 "Navigation bar" Page 62)
2	Display area Display of the level selected in the navigation bar, here the station view
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. (>>> 6.7.1 ""Jogging options" window" Page 70)
4	Jog keys display If axis-specific jogging is selected, the axis numbers are displayed here (A1, A2, etc.). If Cartesian jogging is selected, all the directions of the coordinate system (X, Y, Z, A, B, C) as well as the elbow angle (R) for executing a null space motion are displayed here. (>>> 6.7 "Jogging the robot" Page 70)

Item	Description
5	HOV button Indicates the current jog override. Touching the button opens the Jog override window, in which the jog override can be set. (>>> 6.7.2 "Setting the jog override (HOV)" Page 72)
6	Life sign display A steadily flashing life sign indicates that the smartHMI is active.
7	Select language button Touching the button opens the Select language menu, in which the display language of the smartHMI can be changed.
8	Display of user-defined status keys Currently without function
9	Clock button The clock displays the system time. Touching the button displays the system time in digital format, together with the current date.
10	Jogging type button Displays the currently set mode of the Start/pause key. Touching the button opens the Jogging type window, in which the mode can be changed. (>>> 6.9.3 "Movement mode" window" Page 78)
11	Back button Return to the previous view by touching this button.

6.3.1 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 shows the corresponding application view, robot view 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.

(>>> 6.4 "Calling the main menu" Page 67)

Overview



Fig. 6-4: KUKA smartHMI navigation bar

Item	Description
1	Station level Displays the controller name and the selected operating mode (>>> 6.3.4 "Station view" Page 64)
2	Applications level Displays the selected applications (>>> 6.10.1 "Selecting a robot application" Page 80)
3	Robots level Displays the selected robot (>>> 6.3.5 "Robots view" Page 65)
4	I/O Groups level Displays the selected I/O group. (>>> 6.11.5 "Displaying an I/O group and changing the value of an output" Page 87)

6.3.2 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 (>>> Fig. 6-4). 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.

6.3.3 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. 6-5: Example of keypad

SYM must be pressed to enter 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.

6.3.4 Station view

The station view gives access to information and functionalities which affect the entire station.



Fig. 6-6: Station view

Item	Description
1	Process Data tile Information regarding the process data is displayed here. The configuration of process data is currently not possible.
2	Safety tile Opens the Safety view and displays the safety status of the station. (>>> 13.5 "Activating the safety configuration on the robot controller" Page 155)
3	Frames tile Opens the Frames view. The view contains the frames from the Sunrise project created for the station. (>>> 6.9 "Teaching and manually addressing frames" Page 75)
4	KUKA_Sunrise_Cabinet tile Displays the state of the robot controller (boot state and state of the field buses being used).
5	HMI State tile Displays the connection status between the smartHMI and the application server.
6	Information tile Displays system information, e.g. the IP address of the robot controller.
7	Protocol tile Opens the Protocol view and displays the logged events and changes in state of the system. The display can be filtered based on various criteria. (>>> 18.2 "Displaying the protocol" Page 307)

6.3.5 Robots view

The Robots view gives access to information and functionalities which affect the selected robot.

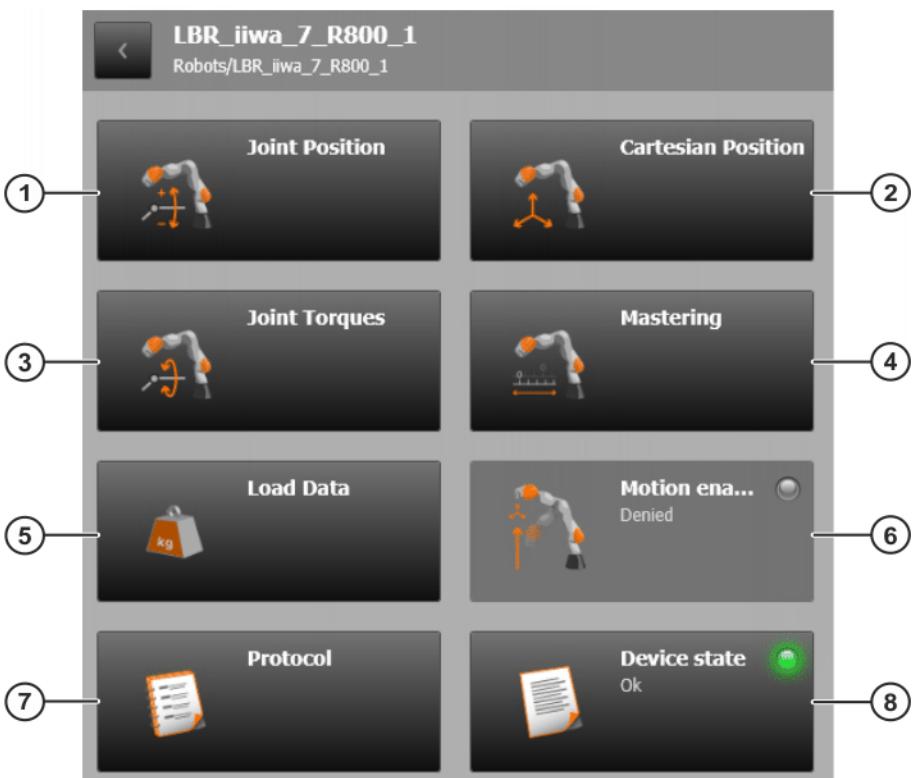


Fig. 6-7: Robots view

Item	Description
1	Joint Position tile Opens the Joint Position view. The axis-specific actual position of the robot is displayed. (>>> 6.11.2 "Displaying the axis-specific actual position" Page 85)
2	Cartesian Position tile Opens the Cartesian Position view. The Cartesian actual position of the robot is displayed. (>>> 6.11.3 "Displaying the Cartesian actual position" Page 86)
3	Joint Torques tile Opens the Joint Torques view. The joint torques of the robot are displayed. (>>> 6.11.4 "Displaying axis-specific torques" Page 86)
4	Mastering tile Opens the Mastering view. The mastering status of the robot axes is displayed. The axes can be mastered or unmastered individually. (>>> 7.1 "Position mastering" Page 91)
5	Load Data tile Opens the Load Data view for automatic load data determination. (>>> 7.2 "Determining tool load data" Page 92)
6	Motion enabling tile Displays whether the robot has received the motion enable.

Item	Description
7	Protocol tile Opens the Protocol view and displays the logged events and changes in state of the system. The display can be filtered based on various criteria. By default, the Source(s) filter is already set on the robot in question. (=> 18.2 "Displaying the protocol" Page 307)
8	Device state tile The status of the robot drive system is displayed.

6.4 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. 6-8: 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.

6.5 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 industrial 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. On the smartPAD, turn the switch for the connection manager to the right.
The connection manager is displayed.
2. Select the operating mode.
3. Turn the switch for the connection manager to the left.

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

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
T2	Testing of programs Only possible with safety gate closed	<ul style="list-style-type: none"> ■ Program verification: Programmed velocity ■ Jog mode: not possible

Operating mode	Use	Velocities
AUT	Automatic execution of programs For industrial robots with and without higher-level controllers	<ul style="list-style-type: none"> ■ Program mode: Programmed velocity ■ Jog mode: not possible
CRR	Motion taking the industrial robot out of a violated space and retraction following collisions CRR is an operating mode that is available if a collision has been detected, or if the robot has violated a safely monitored space, a safely monitored Cartesian velocity limit or a safely monitored force or torque limit and is stopped by the safety controller.	<ul style="list-style-type: none"> ■ Program mode: Reduced programmed velocity, maximum 250 mm/s ■ Jog mode: Jog velocity, maximum 250 mm/s

6.6 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 original 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 the robot base coordinate system is identical to the world coordinate system.

Base coordinate systems

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

By default, the world coordinate system is used as the base coordinate system for a motion. However, additional base coordinate systems can be defined relative to the world coordinate system.

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 116)

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

6.7 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.

6.7.1 "Jogging options" window

Procedure

To open the **Jogging options** window:

- Touch the **Jogging options** button.

To 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 **Jogging options** window.

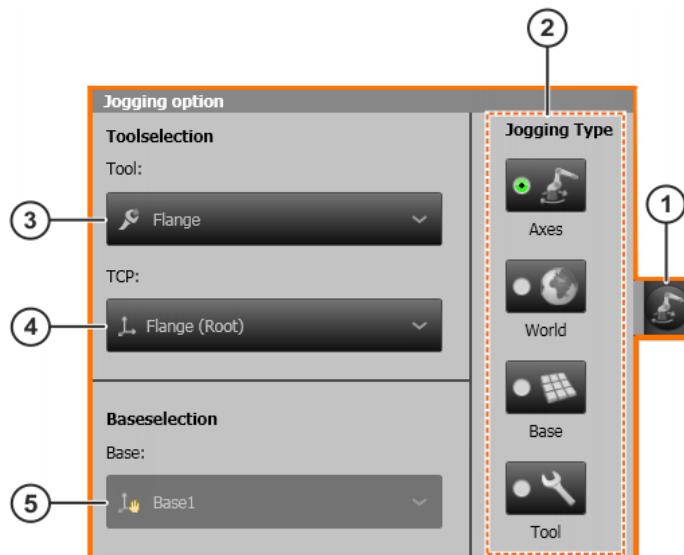


Fig. 6-9: “Jogging options” window

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 in different coordinate systems are possible.
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 . (>>> "Application tool" Page 71)
4	Select the TCP. All the frames of the selected tool are available as the TCP.
5	Select the base. Only possible when the jogging type Base is selected. All frames which were designated in Sunrise.Workbench as a base for jogging 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 frames of a tool or workpiece attached to the robot flange with the `attachTo` command as well as frames created in the application and linked directly or indirectly to the flange.

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 selected, the frame with which the current motion command is executed is automatically set as the TCP in the jogging options. All other frames located hierarchically under the flange coordinate system during the runtime can also be selected as the TCP for jogging.

The robot flange frame is available under the name **ApplicationTool(Root)** for selection as the TCP for jogging.

6.7.2 Setting the jog override (HOV)

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	<ol style="list-style-type: none">1. Touch the HOV button. The Jog override window opens.2. Set the desired jog override. It can be set using either the plus/minus keys or by means of the slider.<ul style="list-style-type: none">■ 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.3. Touch the HOV 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%.

6.7.3 Axis-specific jogging with the jog keys

Precondition	<ul style="list-style-type: none">■ Operating mode T1
Procedure	<ol style="list-style-type: none">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

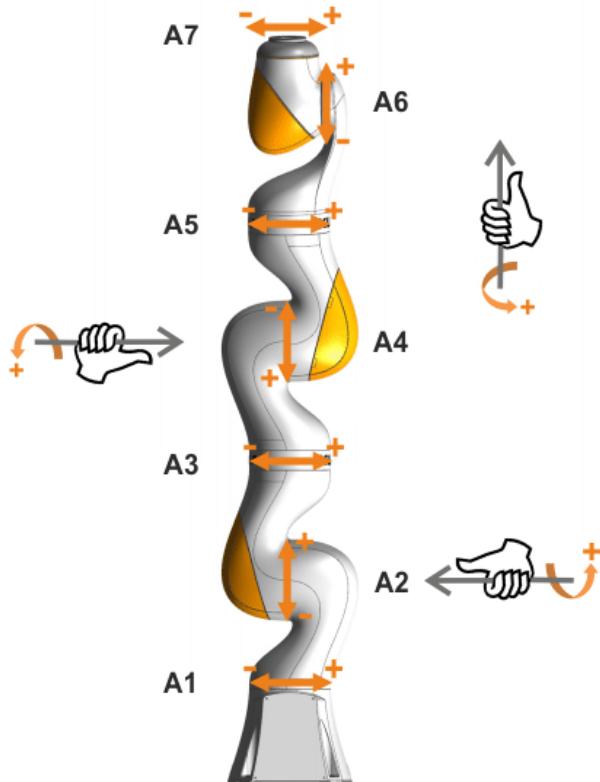


Fig. 6-10: Axis-specific jogging

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 exended 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.

6.7.4 Cartesian jogging with the jog keys

Precondition

- Operating mode T1

Procedure

1. Select the coordinate system for Cartesian jogging. **World**, **Base** and **Tool** are available.

The following designations are displayed next to the jog keys:

- **X, Y, Z**: for the linear motions along the axes of the selected coordinate system
- **A, B, C**: for the rotational motions about the axes of the selected coordinate system
- **R**: for the null space motion

2. Select the desired tool and TCP.
3. If the **Base** coordinate system for Cartesian jogging is selected, select the desired base frame.



The frames available for jogging are selected and enabled in Sunrise.Workbench as a base for jogging.

(>> 9.2.2 "Defining a frame as a "base for jogging"" Page 112)

4. Set the jog override.
5. Hold down the enabling switch.

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

6. Press the plus or minus jog key to move the robot in the positive or negative direction.

6.7.4.1 Null space motion

Description

The lightweight robot has 7 axes, making it 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 zero space motion can be carried out during Cartesian jogging. In the zero 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. 6-11: Null space motion

Characteristics

- 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).
- The position of the elbow angle (R) can be modified using the jog keys during Cartesian jogging.

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.

6.8 CRR mode – controlled robot retraction

Description

CRR is an operating mode which is available when the robot was stopped by the safety controller and the triggering line in the safety configuration contains one of the following monitoring functions:

- Axis range monitoring
- Cartesian velocity monitoring
- Cartesian workspace monitoring
- Cartesian protected space monitoring
- Axis torque monitoring
- Collision detection
- TCP force monitoring

In CRR mode, the robot can be moved out of a violated space or retracted following collisions. The motion velocity of the set working point in CRR mode corresponds to the jog velocity in T1 mode, 250 mm/s maximum.

In CRR mode, the robot can be moved, irrespective of what monitoring functions are active. No stop is triggered if it passes through other monitoring limits. The velocity monitoring functions remain active in CRR mode.

- Procedure**
1. Switch to CRR mode.
 2. Jog the robot and move it back to a position in which the monitoring function that triggered the stop is no longer violated.
If the robot has left the violated space and remains in a permissible range after 4 seconds, the operating mode will automatically change to T1.

6.9 Teaching and manually addressing frames

6.9.1 Displaying frames

- Procedure**
- Select **Frames** in the station view. The Frames view opens.
- Description**
- The view contains the frames created for the Sunrise project. The frames are 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.
- (>>> 6.11.1 "Displaying the end frame of the motion currently being executed" Page 84)

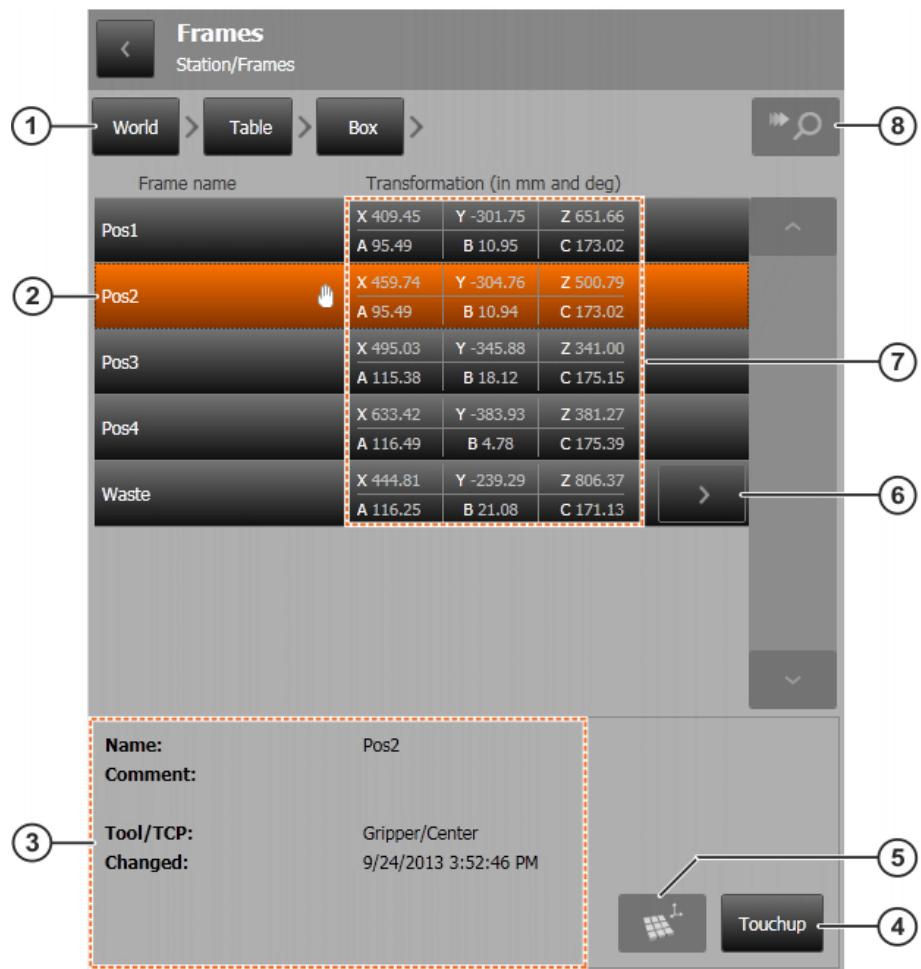


Fig. 6-12: Frames view

Item	Description
1	Frame path Path to the frames of the currently displayed hierarchy level. Goes from World to the direct parent frame (Box here).
2	Frames of the current hierarchy level A frame can be selected by touching it. The selected frame is marked here with a hand icon. The hand icon means that this frame can be used as the base for jogging.
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
4	Touchup button A selected frame can be taught. If no frame is selected, the button is disabled.

Item	Description
5	<p>Button for setting the base for jogging</p> <p>The button sets the selected frame as the base for jogging in the jogging options.</p> <p>(>>> 6.7.1 "Jogging options" window" Page 70)</p> <p>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 for jogging in Sunrise.Workbench.</p>
6	<p>Button for displaying child frames</p> <p>The button is only available if a frame has child elements. The button displays the direct child elements of a frame.</p>
7	Frame coordinates with reference to the parent frame
8	<p>Search button</p> <p>The search 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.</p>

6.9.2 Teaching 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.



Recommendation: Immediately synchronize the project after teaching the frames so that the new frame data will also be updated in the corresponding project in Sunrise.Workbench.

- Precondition**
- The tool with its desired TCP is set.
(>>> 6.7.1 "Jogging options" window" Page 70)
 - Operating mode T1

- 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.



WARNING 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).

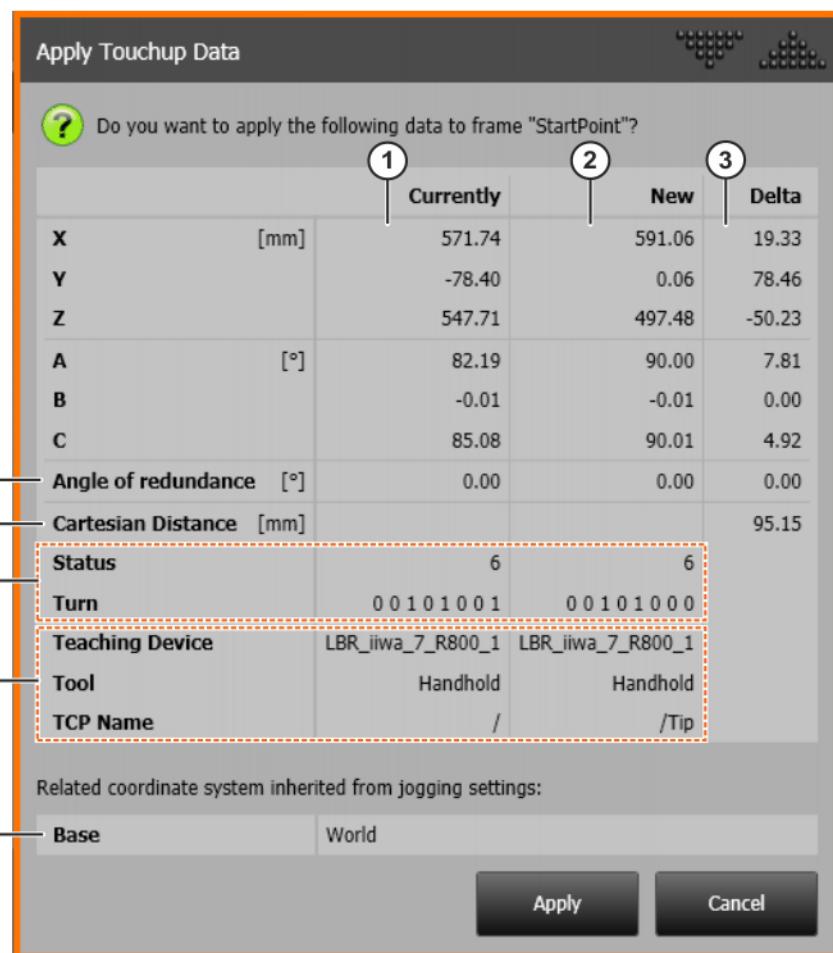


Fig. 6-13: Apply Touchup Data

Item	Description
1	Values saved up to now
2	New values
3	Changes between the old and new values
4	Jogging type from the jogging options The coordinate system currently used for jogging is displayed. (>>> 6.7.1 "Jogging options" window Page 70)
5	Information on the robot and tool used during teaching (>>> 9.2.5 "Displaying and modifying the properties of a frame" Page 114)
6	Redundancy information on the taught point (>>> 9.2.5 "Displaying and modifying the properties of a frame" Page 114)
7	Cartesian distance between the current and new position of the frame

6.9.3 “Movement mode” window

Procedure

To open the **Movement mode** window:

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

To close the **Movement mode** window:

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

Description

The functionality of the Start/pause key can be configured in the **Movement mode** window.

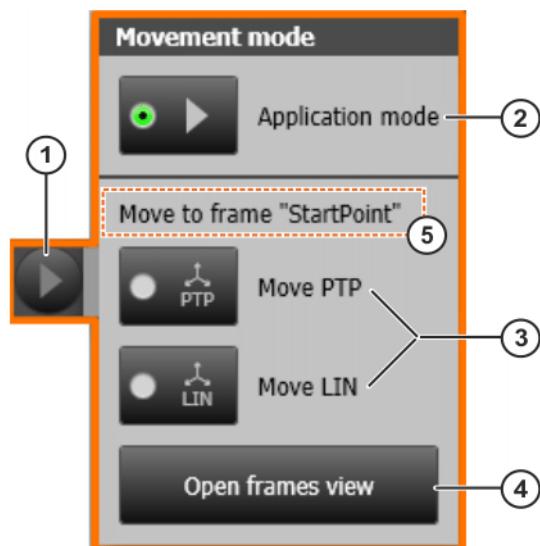


Fig. 6-14: “Movement mode” window

Item	Description
1	Jogging type button The icon displayed depends on the programmed jogging type.
2	Application mode jogging type In this jogging type, the Start/pause key is used to start and pause applications. Note: When switching to Automatic mode, Application mode is set automatically.
3	Move PTP / Move LIN jogging types In these jogging types, the Start/pause key is used to manually address frames with a PTP or LIN motion. These can only be selected if a frame is selected in the frames view and if T1 mode is set. 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. Note: In the Move LIN jogging type, the Status of the end frame is not taken into consideration.
4	Open frames view button. Press the button to switch to the frames view.
5	Frame display The name of the frame currently selected in the frames view is displayed here.

Icons

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

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

6.9.4 Manually addressing frames

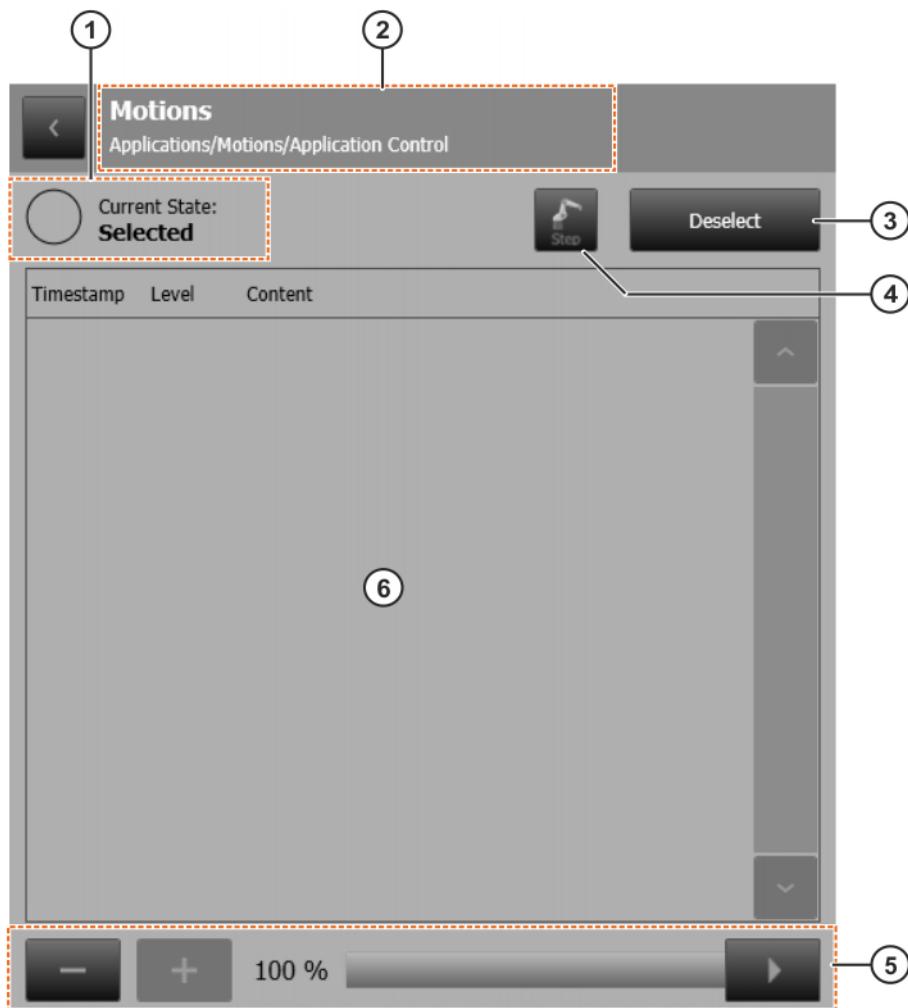
- Description** Taught frames can be manually addressed with a PTP or LIN motion. In a PTP motion, the frame is approached by the quickest route, whereas in a LIN motion it is approached on a predictable path.
- When a frame is being addressed, a warning message is displayed in the following cases:
- 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 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 **Movement mode** window.
 3. Press and hold down the enabling switch.
 4. Press the Start/pause 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.

6.10 Program execution

6.10.1 Selecting a robot application

- Procedure**
- In the navigation bar, select the desired application under **Applications**. The application view opens and the application goes into the **Selected** state.

Description**Fig. 6-15: Application selected**

Item	Description
1	Current status of the application The status is displayed as text and as an icon.
2	Application display The name of the selected application is displayed, here Motions .
3	Deselect button Deselects the selected application and closes the application view. If the application is running or if it is paused, the button is inactive.
4	Step button Pressing the button switches between Step mode and standard mode. (>>> 6.10.2 "Selecting the program run mode" Page 82)
5	Program override (POV) The program override can be set using either the plus/minus keys or by means of the slider.
6	Message window Error messages and user messages programmed in the application are displayed.

Application status display

Icon	State	Description
	Selected	The application is selected.
	Starting	The application is initialized.
	Running	The application is running.
	Motion Paused	The application is paused.
	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.

Functions of the Start/pause key

The following functions are available in the application mode using the Start/pause 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.
	Pause application. A running application can be paused in Automatic mode.



If an application is paused, the robot can be jogged. The tool used in a paused application and the current TCP are not automatically set as the tool and TCP for Cartesian jogging.
(>>> 6.7.1 "Jogging options" window" Page 70)

6.10.2 Selecting the program run mode**Precondition**

- The application is selected.
- T1 or T2 mode

Procedure

- Select the program run mode using the **Step** button.

Description	Program run mode	Description
	Continuous	Standard mode The program is executed through to the end without stopping.
	Step	Step mode The program is executed with a stop after each motion command. The Start/pause key must be pressed again for each motion command. <ul style="list-style-type: none">■ Approximate positioning points are not approximated but rather addressed with exact positioning.■ 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 polled in the source code of the application. ([>>> 15.16 "Changing and polling the program run mode" Page 243](#))

6.10.3 Setting the program override (POV)

- | | |
|---------------------|--|
| Description | Program override is the velocity of the robot during program execution. The program override is specified as a percentage of the programmed velocity.

In T1 mode, the maximum velocity is 250 mm/s, irrespective of the value that is set. |
| Precondition | <ul style="list-style-type: none"> ■ The application is open. |
| Procedure | <ul style="list-style-type: none"> ■ Set the desired program override. It can be set using either the plus/minus keys or by means of the slider. <ul style="list-style-type: none"> ■ 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. |

6.10.4 Starting a program forwards (manually)

- | | |
|---------------------|--|
| Precondition | <ul style="list-style-type: none"> ■ The application is selected. ■ T1 or T2 mode |
| Procedure | <ol style="list-style-type: none"> 1. Set the program override. 2. Select the program run mode. 3. Hold down the enabling switch. 4. Press the Start/pause key and hold it down. The application is executed. <p>To pause a program that has been started manually, release the Start/pause key. If the application is paused, it can be reset by pressing the STOP key.</p> |

6.10.5 Starting a program forwards (automatically)

- | | |
|---------------------|---|
| Precondition | <ul style="list-style-type: none"> ■ The application is selected. ■ Automatic mode ■ The project is not controlled externally. |
|---------------------|---|

Procedure

- Press the Start/pause key. The program is executed.

To pause a program that has been started in Automatic mode, press the Start/pause key again. If the application is paused, it can be reset by pressing the STOP key.

6.10.6 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/pause 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.

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.



In a pause triggered by the Step mode with exact positioning at the end point, it is not possible to reposition to this end point, as it has already been reached. If the motion is continued by pressing the Start/pause key, the next end point will always be addressed directly.

NOTICE

If the robot is jogged off the path after a pause triggered by the Step mode, the Start/pause key may only be pressed if there is no risk of a collision while moving to the next end point. If this is not assured, first move the robot into a suitable position from which it can safely move to the next end point.

Procedure

1. In T1 or T2 mode: press and hold down the enabling switch.
2. Press the Start/pause key and hold it down. The robot returns to the path.

6.11 Monitor functions**6.11.1 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):

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

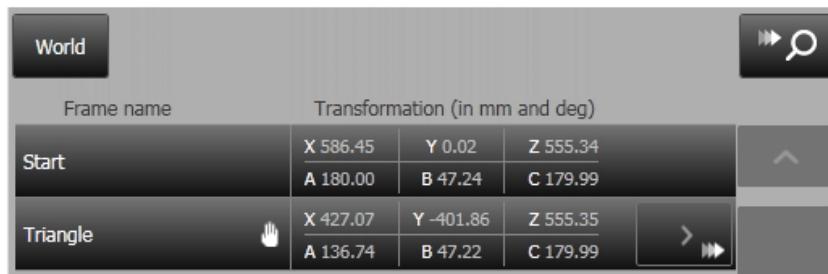
Fig. 6-16: The arrow icon marks the current end frame

If the end frame is located hierarchically below a displayed frame, the button for displaying child frames 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. 6-17: The button switches to the current end frame

You can switch directly to the current end frame using the search button in the upper right-hand area of the frames view:



Frame name	Transformation (in mm and deg)		
Start	X 586.45	Y 0.02	Z 555.34
	A 180.00	B 47.24	C 179.99
Triangle	X 427.07	Y -401.86	Z 555.35
	A 136.74	B 47.22	C 179.99

Fig. 6-18: The search button switches directly to the current end frame

The search button is inactive if no frame is being addressed.

Precondition

- The application is selected.
- Application status **Running** or **Motion Paused**.
- The motion uses an end frame created in the application data.

Procedure

1. Select **Frames** in the station view. The Frames view opens.
2. Switch to the end frame using the button for displaying child frames or using the search button.

6.11.2 Displaying the axis-specific actual position

Procedure

- Select **Joint Position** in the station view.

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.

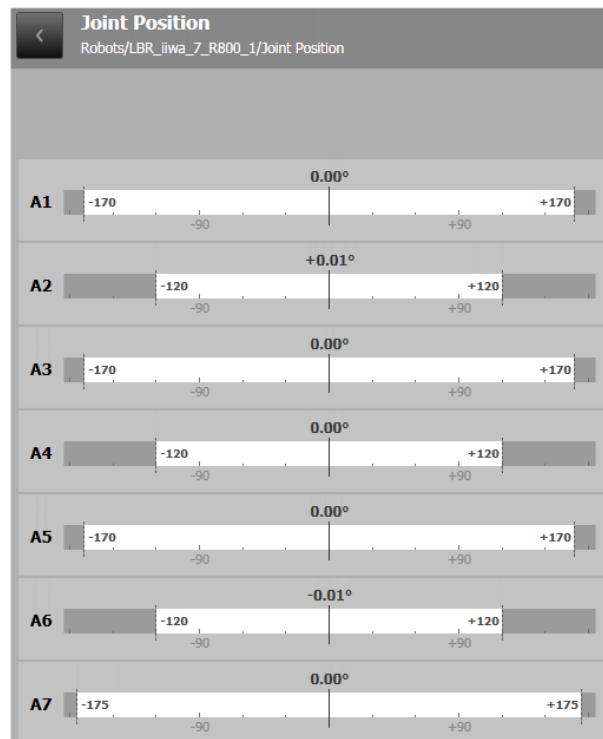


Fig. 6-19: Axis-specific actual position

6.11.3 Displaying the Cartesian actual position

Procedure

1. Select **Cartesian Position** in the Robot view.
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.

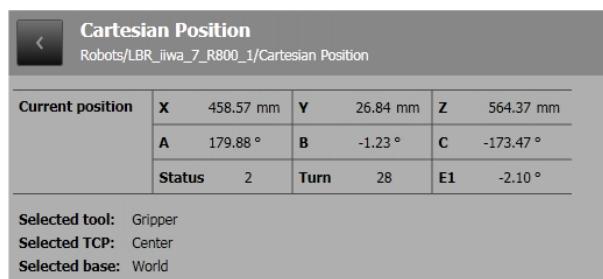


Fig. 6-20: Cartesian actual position

6.11.4 Displaying axis-specific torques

Procedure

- Select **Joint Torques** in the Robot view.

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, i.e. either the negative or positive part.

The display contains the following data:

- Current absolute torques
- Current external torques

i The external torques are only displayed correctly if the robot has been moved. For example, the robot must be moved after the robot controller is booted or after a tool change in order for the external torques to be displayed correctly.

- Current tool

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

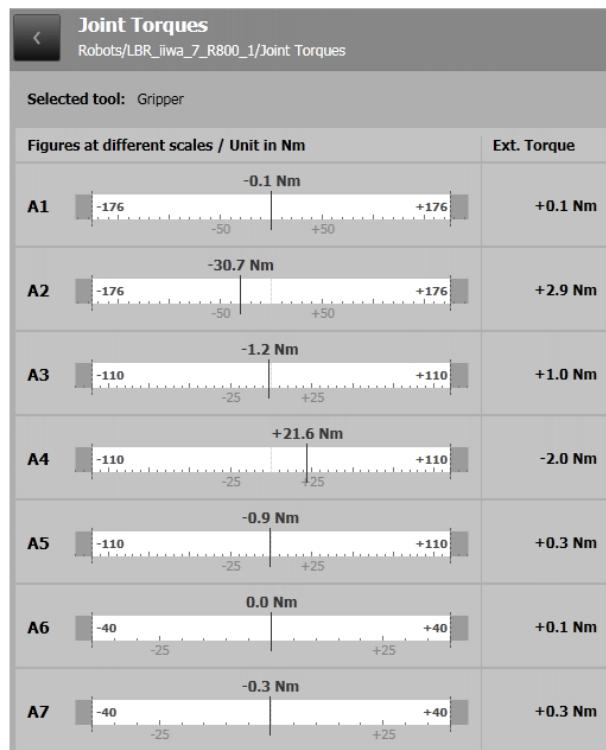


Fig. 6-21: Axis-specific torques

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

Precondition

To change an output:

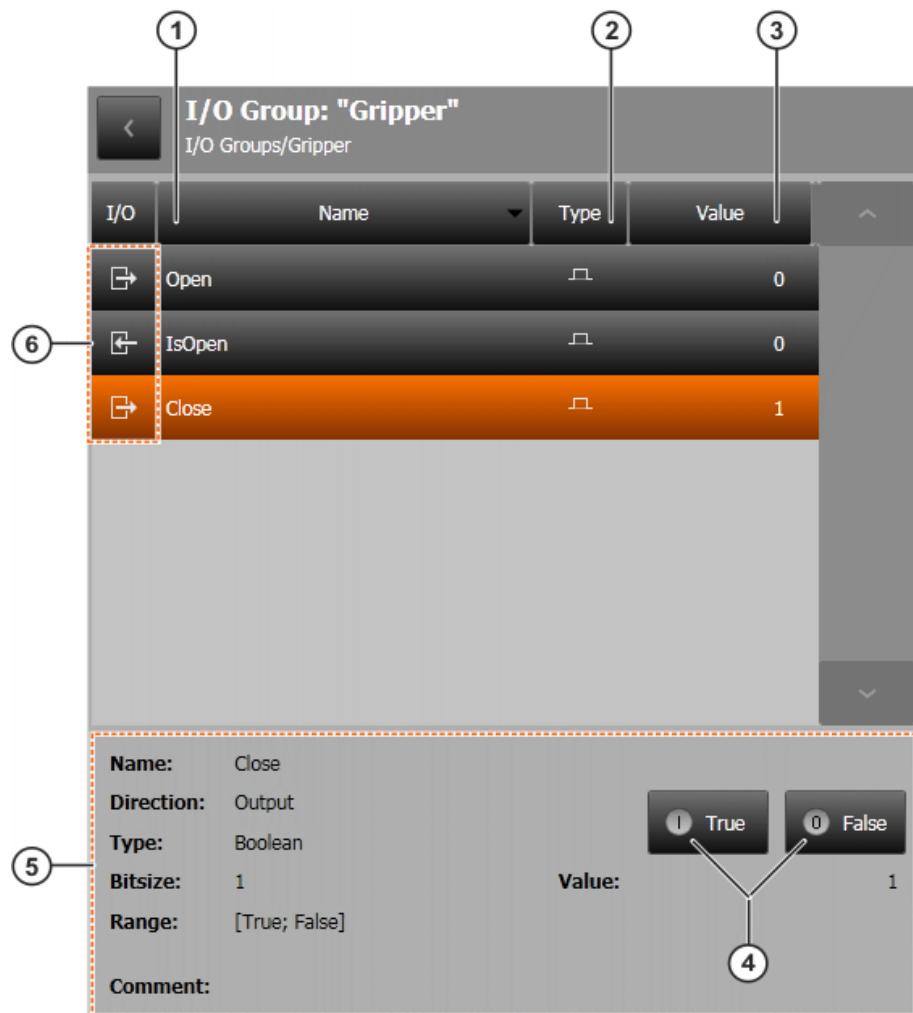
- Operating mode T1, T2 or CRR



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. 6-22: Inputs/outputs of an I/O group**

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

6.11.6 Displaying the IP address and software version**Procedure**

- Select the **Information** tile in the Station view.

The current installed version of the system software and the IP address of the robot controller are displayed under the **Station** node.

7 Start-up and recommissioning

7.1 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).

7.1.1 Mastering axes

Description The LBR iiwa 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.



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.
- Always master the individual axes in the same order.
- Always carry out mastering without a load. Mastering with a load is not currently supported.

Precondition

- Operating mode T1 or CRR

Procedure

1. Select **Mastering** in the Robot view. The **Mastering** view opens.
2. Press and hold down the enabling switch.
3. Press 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).



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

7.1.2 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 **Mastering** in the Robot view. The **Mastering** view opens.
2. Press the **Unmaster** button for the mastered axis. The axis is unmastered.

7.2 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.

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.

- 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 only the center of mass is to be determined on the basis of a specified mass, the criterion 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 slight external torques acting on the load-free robot in this position. This can be checked via the display of the external axis torques.

(>>> 6.11.4 "Displaying axis-specific torques" Page 86)

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.



The application of interference torques during load data determination results in falsified load data. (>>> 9.3.6 "Load data" Page 120)

- 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 Sunrise.Workbench.
 - 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 manipulator during the measurement runs.

It is advisable to carry out load data determination in T1 mode. This does not affect the quality of the measurement results.

- Procedure**
1. Select **Load Data** in the Robot view. 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 the **Determine load data** button.
 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.
 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 dialog **Apply load data**.

Press **Apply** to save the determined load data.

7. Synchronize the project so that the load data are updated in Sunrise.Workbench.

Overview

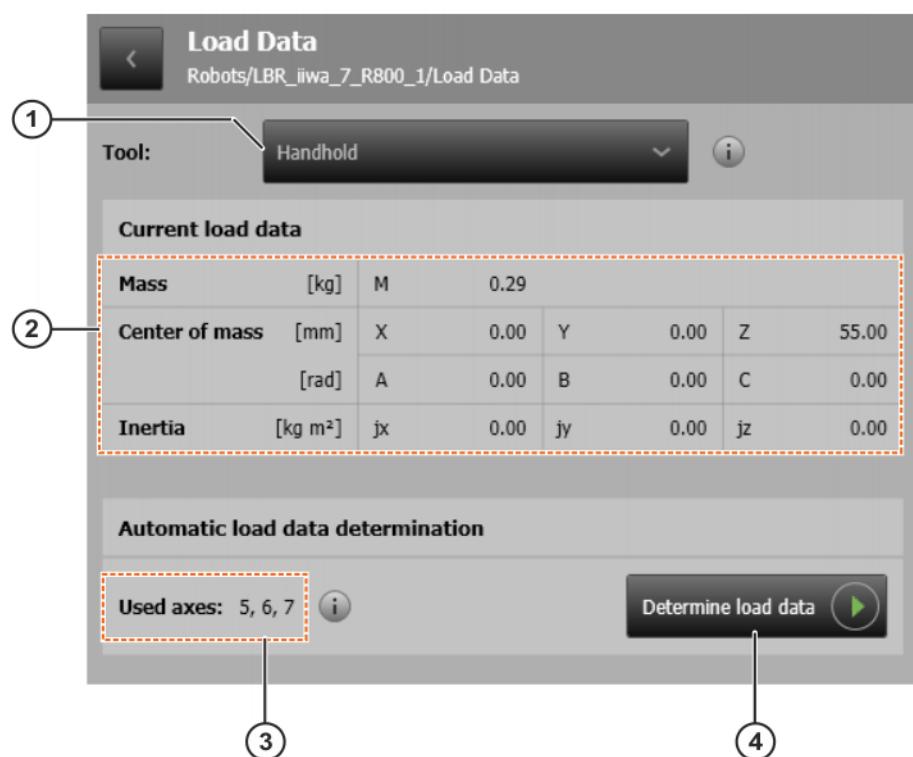


Fig. 7-1: Determining the load data

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	Determine 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 Overview of the brake test

Description Each robot axis has a holding brake integrated into the drive train. The brakes have 2 functions:

- Stopping the robot when the servo control is deactivated or the robot is de-energized.
- Switching the robot to the safe state “Standstill” in the event of a fault.

The brake test checks whether the holding torque applied by each brake is high enough, i.e. whether it exceeds a specific reference torque.

NOTICE

The brake test must be carried out for each axis during start-up and recommissioning of the industrial robot. Irrespective of the period of operation and type of application, the brake test must be performed daily during operation.

Execution A precondition for execution of the brake test is that the robot is at operating temperature.

The brake test is manually executed by means of an application. A prepared brake test application for the LBR iiwa is available from Sunrise.Workbench.

Prior to the actual brake test, the robot is moved to determine the maximum absolute torque for each axis. In the brake test application, this torque is communicated to the brake test as the reference holding torque.



Before the brake test is started, all motions must be completed, and no commands may still be active on the robot controller. No motion commands may be executed during the brake test. Otherwise, the brake test is aborted.

Procedure

When a brake is tested, the following steps are carried out by default:

1. The axis moves at constant velocity in both directions over a small axis angle ($\pm 5^\circ$ on the output side). The gravitation and friction are determined during this motion.
2. When the axis has returned to its starting position and the axis drive is stationary, the brake is closed.
3. With the brake closed, the setpoint torque of the drive is set to 80% of the brake holding torque.

The lowest of the following values is used as the holding torque to be tested: the reference holding torque determined, the brake holding torque or the motor holding torque.



The brake holding torque is saved in the motor data. The motor holding torque is derived from the motor data.

4. The drive torque is gradually increased until a change in position is detected or the maximum permissible brake holding torque is reached. The brake test ends when the maximum torque is reached.
5. The torque applied against the brake when a change in position is detected is measured. This is the measured holding torque.
6. The measured holding torque is evaluated relative to the holding torque to be tested.

The brake test is successful if the measured holding torque lies within the following range:

- \geq holding torque to be tested ... \leq maximum brake holding torque

If the measured holding torque lies below the holding torque to be tested, the brake test has failed, i.e. the brake is identified as being defective.

The test result is displayed on the smartHMI.

([>>> 8.4.2 "Results of the brake test \(display\) " Page 108](#))

7. When the brake test has ended for all axes and the robot is stationary, the brakes are briefly opened and closed again. This releases any remaining tension in the brake and prevents undesired robot motions.



If the brake test for an axis fails, i.e. a brake has been identified as being defective, the brake test can be repeated for confirmation.



The brake test can only be performed in Automatic mode.



If the application is paused during the brake test or if a safety stop is triggered, e.g. by an EMERGENCY STOP, the brake test is aborted.



The brake test does not depend on the loads mounted on the robot, as gravitation and friction are taken into consideration when the test is carried out.

Overview

The following describes the steps for executing the brake test with the template available in Sunrise.Workbench.

The brake test application can be adapted and expanded. The comments contained in the template must be observed.

NOTICE

If a brake is defective, the corresponding axis may slip during the brake test and the robot may sag. The brake test must be executed in a position in which no damage could result from potential sagging. The starting position for the brake test must be selected accordingly.

NOTICE

If the brake test fails for an axis, i.e. if a brake has been identified as being defective, the application must ensure that the robot is automatically moved to a safe position. A safe position is a position in which the robot is supported in such a way that it either cannot sag or cannot cause damage in the event of sagging.

Step	Description
1	Create the brake test application from the template. (>>> 8.2 "Creating the brake test application from the template" Page 97)
2	Integrate the application-specific maximum absolute torques determined into the brake test application. At the start of the brake test application, 2 pre-defined axis positions are addressed by default. The maximum absolute torque for each axis is thus determined and communicated to the brake test as the reference torque. As a rule, the brake test requires the maximum absolute torques which occur when a user-specific robot application is executed. The user-specific robot application can be inserted into the brake test application. (>>> 8.2.1 "Changing the motion sequence for torque value determination" Page 99) The brake test application can also be executed without determining the reference torques and communicating these to the brake test. (>>> 8.2.2 "Removing torque value determination from the brake test application" Page 100)
3	Change the starting position for the brake test. The starting position is the vertical stretch position by default. If required, a different starting position can be selected. (>>> 8.2.3 "Changing the starting position for the brake test" Page 101)
4	If necessary, make further user-specific adaptations in the brake test application. Examples: <ul style="list-style-type: none">■ Setting the output for a failed brake test.■ Saving the test results in a file. (>>> 8.3 "Programming interface for the brake test" Page 101)
5	Synchronize the project in order to transfer the brake test application to the robot controller.
6	Execute the brake test application in Automatic mode. (>>> 8.4 "Performing a brake test" Page 107)

8.2 Creating the brake test application from the template

- Procedure**
1. Select the Sunrise project in the **Package Explorer**.
 2. Select the menu sequence **File > New > Other....**
 3. In the **Sunrise** folder, select the **Application for Brake Test of LBR iiwa** option and click on **Finish**.
- The **BrakeTestApplication.java** application is created in the source folder of the project and opened in the editor area of Sunrise.Workbench.
- Description**
- In the run() method of the **BrakeTestApplication.java** application (limited here to the relevant command lines), the execution of the brake test is implemented for all axes of the LBR iiwa.

A data evaluation preceding the actual brake test is also implemented. 2 pre-defined axis positions are addressed in order to determine the maximum absolute torque for each axis.

```

1 public void run() {
2 ...
3 lbr_iwa.move(ptpHome());
4 ...
5 MeasuredTorqueEvaluator evaluator =
6     new MeasuredTorqueEvaluator(lbr_iwa);
7 evaluator.startEvaluation();
8 ...
9 lbr_iwa.move(new PTP(new JointPosition(
10     0.5, 0.8, 0.2, 1.0, -0.5, -0.5, -1.5))
11     .setJointVelocityRel(relVelocity));
12 lbr_iwa.move(new PTP(new JointPosition(
13     -0.5, -0.8, -0.2, -1.0, 0.5, 0.5, 1.5))
14     .setJointVelocityRel(relVelocity));
15 ...
16 MeasuredTorqueStatistic maxTorqueData =
17     evaluator.stopEvaluation();
18 ...
19 lbr_iwa.move(ptpHome());
20 ...
21 try {
22     TimeUnit.SECONDS
23         .sleep(((Double) RoboticsAPIDefaults
24             .getRequiredParameter("MAINTAIN_TIMEOUT_SECS"))
25             .longValue() + 1);
26 } catch (InterruptedException ex) {
27     ...
28     return;
29 }
30 ...
31 BrakeTest brakeTest = new (lbr_iwa);
32 ...
33 for (int axis : axes) {
34     try {
35         BrakeTestResult result = brakeTest.execute(axis,
36             maxTorqueData.getMaxAbsMsrTorqueValues()[axis]);
37
38         if (LogLevel.Info == result.getState().getLogLevel()) {
39             getLogger().info(result.toString());
40         } else if LogLevel.Warning == result
41             .getState().getLogLevel()) {
42             getLogger().warn(result.toString());
43         } else {
44             getLogger().error(result.toString());
45         }
46     } catch (BrakeTestException ex) {
47     ...
48 }
49 }
50 ...
51 lbr_iwa.move(positionHold(new PositionControlMode(),
52     1000, TimeUnit.MILLISECONDS));
53 }
```

Line	Description
3	Address the starting position from which the robot is moved to determine the maximum absolute torque for each axis. The starting position is the vertical stretch position by default.
5 ... 6	Prepare the data evaluation. In order to perform an axis-specific evaluation of the torques measured during a motion sequence, an instance of the MeasuredTorqueEvaluator class must be created.

Line	Description
7	<p>Start the data evaluation.</p> <p>The data evaluation is started with the startEvaluation() command. The command belongs to the MeasuredTorqueEvaluator class.</p>
9 ... 14	<p>Carry out the motion sequence to determine the maximum absolute torques</p> <p>2 pre-defined axis positions are each addressed with a PTP motion.</p>
16 ... 17	<p>End the data evaluation and poll the data.</p> <p>The stopEvaluation() command ends the data evaluation and returns the result as an object of type MeasuredTorqueStatistic. The command belongs to the MeasuredTorqueEvaluator class.</p>
19	<p>Address the starting position for the brake test.</p> <p>The starting position is the vertical stretch position by default.</p>
21 ... 29	<p>Wait time</p> <p>A wait time is inserted before the start of the brake test. This is to ensure that no commands are still active on the robot controller when the brake test is started. Otherwise, the brake test would be aborted.</p>
31	<p>Create an instance for the brake test.</p> <p>The brake test can be started and evaluated with an instance of the BrakeTest class.</p>
33 ... 49	<p>Execute the brake test</p> <p>The brakes are tested one after the other, starting with axis A1. The test execution is started via the BrakeTest method execute(...) (lines 35 ... 36).</p> <p>The maximum absolute torque determined is communicated to the brake test as the reference torque. The test results are displayed individually for each axis on the smartHMI.</p>
51 ... 52	<p>Release any tension.</p> <p>The brakes are opened once and closed again. This releases any remaining tension in the brake and prevents undesired robot motions.</p>

8.2.1 Changing the motion sequence for torque value determination

Description	<p>The brake test application created from the template contains a prepared motion sequence for determining the maximum absolute torques generated in each axis. The determination of the maximum absolute torques is referred to in the following as torque value determination.</p> <p>The robot is moved from the vertical stretch position by default. A different starting position can be selected.</p> <p>2 pre-defined axis positions are each addressed from the starting position with a PTP motion. As a rule, the brake test requires the maximum absolute torques which occur when a user-specific robot application is executed. The user-specific robot application can be inserted into the brake test application.</p>
--------------------	---

```

    public void run() {
        // initial position
        getLogger().info("Moving into initial position.");
        lbr_iwa.move(ptpHome());
    }

    // start monitoring for maximum torques
    getLogger().info("Start evaluation of torque statistic.");
    MeasuredTorqueEvaluator evaluator = new MeasuredTorqueEvaluator(lbr_iwa);
    evaluator.startEvaluation();

    getLogger().info("Perform desired robot motion.");
    lbr_iwa.move(new PTP(new JointPosition(0.5, 0.8, 0.2, 1.0, -0.5, -1.5)).setJointVelocityRel(relVelocity));
    lbr_iwa.move(new PTP(new JointPosition(-0.5, -0.8, -0.2, -1.0, 0.5, 0.5, 1.5)).setJointVelocityRel(relVelocity));

    // stop monitoring for maximum torques and store results
    getLogger().info("Stop evaluation of torque statistic.");
    MeasuredTorqueStatistic maxTorqueData = evaluator.stopEvaluation();
    getLogger().info("The result of evaluation:" + maxTorqueData.toString());
}

```

Fig. 8-1: Motion sequence for torque value determination

- 1 ptpHome() motion to starting position
- 2 Pre-defined motion sequence for torque value determination

Procedure

1. Open the brake test application in Sunrise.Workbench.
2. If necessary, make the following changes to the run() method of the application:
 - Replace the ptpHome() motion that brings the robot to the starting position with a motion to the desired starting position.
 - Replace the pre-defined motion sequence with the appropriate application code.
3. Save changes.

8.2.2 Removing torque value determination from the brake test application

Description

If the brake test is to be executed without reference torques being determined and made available to the brake test, all the command lines relevant for torque value determination must be removed from the brake test application. As a consequence, the brake test application then starts with the motion to the starting position for the brake test.

Furthermore, the parameter with which the reference torque is specified must be removed from the execute(...) method call which starts the brake test for an axis.

```

    // initialize and start brake test
    getLogger().info("Executing brake test.");
    BrakeTest brakeTest = new BrakeTest(lbr_iwa);
    // if execution of application is paused, brake test will be aborted for all axes
    for (int axis : axes) {
        try {
            BrakeTestResult result = brakeTest.execute(axis, maxTorqueData.getMaxAbsMsrTorqueValues()[axis]);
            if (LogLevel.Info == result.getState().getLogLevel()) {
                getLogger().info(result.toString());
            }
        }
    }
}

```

Fig. 8-2: Transferring the reference torque for the brake test

- 1 Call of execute(...) with transfer of reference torque

Procedure

1. Open the brake test application in Sunrise.Workbench.
2. Make the following changes to the run() method of the application:
 - Delete all command lines which are relevant for torque value determination.
 - Delete the following parameter from the execute(...) method call:
`maxTorqueData.getMaxAbsMsrTorqueValues() [axis]`
 The following code remains in the line:

```
BrakeTestResult result = brakeTest.execute(axis);
```

3. Save changes.

8.2.3 Changing the starting position for the brake test

Description

The brake test application created from the template executes the brake test in the vertical stretch position by default. If this is not possible, for example due to space constraints, the starting position can be changed.



In the starting position for the brake test, each axis must have a range of motion of at least $\pm 5^\circ$ on the output side.

```

MeasuredTorqueStatistic maxTorqueData = evaluator.stopEvaluation();
getLogger().info("The result of evaluation:\n" + maxTorqueData.toString());

// now perform brake test with evaluated torques

// initial position
getLogger().info("Moving into brake test position.");
① lbr_iwa.move(ptpHome()); // Motion to starting position

// assure the motion has finished
getLogger().info("Waiting for brakes to be closed.");
try {
    TimeUnit.SECONDS
        .sleep(((Double) RoboticsAPIDefaults
            .getRequiredParameter("MAINTAIN_TIMEOUT_SECS"))
            .longValue() + 1);
}

```

Fig. 8-3: Starting position for the brake test

- 1 ptpHome() motion to starting position

Procedure

1. Open the brake test application in Sunrise.Workbench.
2. In the run() method of the application, replace the ptpHome() motion that brings the robot into the starting position with a motion to the desired starting position.
3. Save changes.

8.3 Programming interface for the brake test

With the BrakeTest class, the RoboticsAPI offers a programming interface for the execution of the brake test.

In addition, using the MeasuredTorqueEvaluator class, the torques measured during a motion sequence can be evaluated and the maximum absolute torque for each axis can be determined. This torque can be used as the reference torque for the brake test.

8.3.1 Evaluating measured torques and determining the maximum absolute value

Description

In order to perform an axis-specific evaluation of the torques measured during a motion sequence, an object of the MeasuredTorqueEvaluator class must first be created. The LBR instance for whose axes the maximum absolute torque values are to be determined is transferred to the constructor of the MeasuredTorqueEvaluator class.

The evaluation can be started and then ended with the following methods of the MeasuredTorqueEvaluator class:

- **startEvaluation():** Starts the evaluation.
Once the method has been called, the motion sequence to be evaluated must be commanded.
- **stopEvaluation():** Ends the evaluation.
The method returns an object of type MeasuredTorqueStatistic. The results of the evaluation can be polled via this object.

Syntax

```
MeasuredTorqueEvaluator evaluator =
new MeasuredTorqueEvaluator(lbr_iwa);

evaluator.startEvaluation();

// Motion sequence

MeasuredTorqueStatistic maxTorqueData =
evaluator.stopEvaluation();
```

Explanation of the syntax

Element	Description
<i>evaluator</i>	Type: MeasuredTorqueEvaluator Variable to which the created MeasuredTorqueEvaluator instance is assigned. The evaluation of the torques during a motion sequence is started and ended via the variable.
<i>lbr_iwa</i>	Type: LBR LBR instance of the application. Represents the robot for which the maximum absolute torque values are to be determined.
<i>maxTorque Data</i>	Type: MeasuredTorqueStatistic Variable for the return value of stopEvaluation(). The return value contains the determined maximum absolute torque values and further information for the evaluation.

8.3.2 Polling the evaluation results of the maximum absolute torques

When the evaluation of the maximum absolute torque values has ended, the results of the evaluation can be polled.

Overview

The following methods of the MeasuredTorqueStatistic class are available:

Method	Description
getMaxAbsMsrTorqueValues()	Return value type: double[]; unit: Nm Returns a double array containing the determined maximum absolute torque values (output side) for all axes.
getSingleMaxAbsMsrTorqueValue(...)	Return value type: double; unit: Nm Returns the maximum absolute torque value (output side) for the axis which is transferred as the parameter (type: JointEnum).
areDataValid()	Return value type: boolean The system polls whether the determined data are valid (= true). The data are valid if no errors occur during command processing.
getStartTimestamp()	Return value type: java.util.Date Returns the time at which the evaluation was started.
getStopTimestamp()	Return value type: java.util.Date Returns the time at which the evaluation was ended.

Example

The maximum torques which occur during a joining task are to be used as reference torques in a brake test. For this purpose, the torques which are measured during the execution of the joining task are evaluated, and the maximum absolute measured torque for each axis is determined.

Once the evaluation has been started, the motion commands of the joining process are executed. When the joining process is completed, the evaluation is ended and the results of the evaluation for axes A2 and A4 are saved in the process data. If the determined data are invalid, an output is set.

```

private LBR testLBR;
private BrakeTestIOGroup brakeTestIOS;
private Tool testGripper;
private Workpiece testWorkpiece;
...
public void run() {

    testGripper.attachTo(testLBR.getFlange());
    testWorkpiece.attachTo(testGripper.getFrame("/GripPoint"));

    // create MeasuredTorqueEvaluator
    MeasuredTorqueEvaluator testEvaluator =
        new MeasuredTorqueEvaluator(testLBR);

    // start evaluation
    testEvaluator.startEvaluation();

    // performs assembly task
    testAssemblyTask();

    // finish evaluation and store result in variable testMaxTrqData
    MeasuredTorqueStatistic testMaxTrqData =
        testEvaluator.stopEvaluation();

    // get maximum absolute measured torque value for joint 2
    double maxTrqA2 = testMaxTrqData
        .getSingleMaxAbsMsrTorqueValue(JointEnum.J2);

    // save result
    getApplicationData().getProcessData("maxTrqA2").setValue(maxTrqA2);

    // get maximum absolute measured torque value for joint 4
    double maxTrqA4 = testMaxTrqData
        .getSingleMaxAbsMsrTorqueValue(JointEnum.J4);

    // save result
    getApplicationData().getProcessData("maxTrqA4").setValue(maxTrqA4);

    // check if evaluated data is valid
    boolean areDataValid = testMaxTrqData.areDataValid();
    if(areDataValid == false){
        // if data is not valid, set output signal
        brakeTestIOS.setEvaluatedTorqueInvalid(true);
    }
    ...
}

public void exampleAssemblyTask(){

    testLBR.move(ptp(getFrame("/StartAssembly")));

    ForceCondition testForceCondition =
        ForceCondition.createNormalForceCondition
            (testWorkpiece.getDefaultMotionFrame(), CoordinateAxis.Z, 15.0);
    testWorkpiece.move(linRel(0.0, 0.0, 100.0)
        .breakWhen(testForceCondition));

    CartesianSineImpedanceControlMode testAssemblyMode =
        CartesianSineImpedanceControlMode.createLissajousPattern(
            CartPlane.XY, 5.0, 10.0, 500.0);

    testWorkpiece.move(positionHold(
        testAssemblyMode, 3.0, TimeUnit.SECONDS));

    openGripper();
    testWorkpiece.detach();

    testGripper.move(linRel(0.0, 0.0, -100.0));
}

```

8.3.3 Creating an object for the brake test

Description	In order to be able to execute the brake test, an object of the BrakeTest class must first be created. The LBR instance for which the brake test is to be executed is transferred to the constructor of the BrakeTest class.						
Syntax	<pre>BrakeTest brakeTest = new BrakeTest (lbr_iwia);</pre>						
Explanation of the syntax	<table border="1"> <thead> <tr> <th>Element</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>brakeTest</i></td><td>Type: BrakeTest Variable to which the created BrakeTest instance is assigned. The execution of the brake test is commanded via the variable.</td></tr> <tr> <td><i>lbr_iwia</i></td><td>Type: LBR LBR instance of the application. Represents the robot for which the brake test is to be carried out.</td></tr> </tbody> </table>	Element	Description	<i>brakeTest</i>	Type: BrakeTest Variable to which the created BrakeTest instance is assigned. The execution of the brake test is commanded via the variable.	<i>lbr_iwia</i>	Type: LBR LBR instance of the application. Represents the robot for which the brake test is to be carried out.
Element	Description						
<i>brakeTest</i>	Type: BrakeTest Variable to which the created BrakeTest instance is assigned. The execution of the brake test is commanded via the variable.						
<i>lbr_iwia</i>	Type: LBR LBR instance of the application. Represents the robot for which the brake test is to be carried out.						

8.3.4 Starting the execution of the brake test

Description	The execution of the brake test is started via the execute(...) method. The axis whose brake is to be tested is transferred when the method is called. In addition, a reference holding torque can be transferred, e.g. the maximum absolute axis torque which occurs in a specific application. The method returns an object of type BrakeTestResult via which the results of the brake test can be polled. When execute(...) is called, the brake test is started at the actual position of the robot. execute(...) must be called within a try/catch block so that it can react appropriately to an aborted brake test.
--------------------	--



Before the brake test is started, all motions must be completed, and no commands may still be active on the robot controller. No motion commands may be executed during the brake test. Otherwise, the brake test is aborted.

When the brake test has ended for all axes and the robot is stationary, the brakes should be opened once and closed again, e.g. by sending the position-Hold(...) motion command. This releases any remaining tension in the brake and prevents undesired robot motions.

Syntax

```
BrakeTest brakeTest = ...;  
  
try{  
  
    BrakeTestResult result = brakeTest.execute (axis,  
        <absMaxHoldingTorque>);  
  
    ...  
  
} catch(BrakeTestException ex{  
    ...  
}
```

Explanation of the syntax

Element	Description
<i>result</i>	Type: BrakeTestResult Variable for the return value of execute(...). The return value contains the results of the brake test and further information concerning the brake test which can be polled via the variable.
<i>brakeTest</i>	Type: BrakeTest Variable to which the created BrakeTest instance is assigned. The execution of the brake test is commanded via the variable.
<i>axis</i>	Type: int Index of the axis whose brake is to be tested. ■ 0 ... 6: Axes A1 ... A7
<i>absMaxHoldingTorque</i>	Type: double; unit: Nm Reference holding torque specified by the user (output side) If the value <i>absMaxHoldingTorque</i> is specified, the lowest of the following values is used as the holding torque to be tested: <i>absMaxHoldingTorque</i> , brake holding torque or motor holding torque. If the value <i>absMaxHoldingTorque</i> is not specified, the lowest of the following values is used as the holding torque to be tested: brake holding torque or motor holding torque. ■ Minimum: <i>absMaxHoldingTorque</i> > 0.0 Nm ■ Maximum: The value of <i>absMaxHoldingTorque</i> must lie below one of the following values: brake holding torque or motor holding torque. Note: The brake holding torque is saved in the motor data. The motor holding torque is derived from the motor data.
<i>ex</i>	Type: BrakeTestException Exception which occurs when the brake test is aborted. The exception is treated within the catch block in such a way that an aborted brake test for a single brake does not cancel the entire application.

8.3.5 Evaluating the brake test

The execute(...) method which starts the execution of the brake test returns an object of type BrakeTestResult. Various information concerning the executed brake test can be polled from this object.

Overview

The following methods of the BrakeTestResult class are available:

Method	Description
<code>getAxis()</code>	Return value type: int Returns the index of the axis whose brake has been tested. The index starts with 0 (= axis A1).
<code>getMeasuredTorque()</code>	Return value type: double; unit: Nm Returns the holding torque (output side) measured during the brake test. This value is compared with the holding torque to be tested.

Method	Description
getState()	<p>Return value type: Enum of type BrakeTestResult.State Returns the results of the brake test. (>>> 8.3.5.1 "Polling the results of the brake test" Page 106)</p>
getTestedTorque()	<p>Return value type: double; unit: Nm Returns the test holding torque with which the holding torque (output side) applied and measured during the brake test is compared.</p>
getTimestamp()	<p>Return value type: java.util.Date Returns the time at which the brake test was started.</p>

8.3.5.1 Polling the results of the brake test

Description The test results are polled via the BrakeTestResult method getState(). An enum of type BrakeTestResult.State is returned; its values describe the possible test results. In addition, the log level corresponding to the test results can be polled with getLogLevel().

Syntax

```
BrakeTestResult result = . . . ;
BrakeTestResult.State state = result.getState();
BrakeTestResult.LogLevel logLevel = state.getLogLevel();
```

Explanation of the syntax	Element	Description
	<i>result</i>	Type: BrakeTestResult Variable for the return value of execute(...). The return value contains the results of the brake test and further information concerning the brake test which can be polled via the variable.
	<i>state</i>	Type: Enum of type BrakeTestResult.State Variable for the return value of getState(). The return value contains the test results.
	<i>logLevel</i>	Type: Enum of type BrakeTestResult.LogLevel Variable for the return value of getLogLevel(). The return value contains the log level of the test results.

BrakeTestResult.State The enum of type BrakeTestResult.State has the following values:

Value	Description
BrakeUntested	The brake test could not be executed or was aborted during execution due to faults. Log level: BrakeTestResult.LogLevel.Error
BrakeUnknown	The brake test could not be executed because not enough torque could be generated (e.g. due to excessive friction). Log level: BrakeTestResult.LogLevel.Error
BrakeError	The brake test has failed. The measured holding torque falls below the holding torque to be tested. The brake is defective. Log level: BrakeTestResult.LogLevel.Error

Value	Description
BrakeWarning	The measured holding torque is less than 5% above the holding torque to be tested. The brake has reached the wear limit and will soon be identified as defective. Log level: BrakeTestResult.LogLevel.Warning
BrakeMax Unknown	The brake test could not be executed, as it was not possible to test against the maximum brake holding torque. Possible cause: The value is missing from the motor data. Log level: BrakeTestResult.LogLevel.Error
BrakeExcessive	The measured holding torque is greater than the maximum brake holding torque. Stopping using the brake can cause damage to the machine. Log level: BrakeTestResult.LogLevel.Warning
BrakeReady	The measured holding torque exceeds the holding torque to be tested by more than 5 %. The brake is fully operational. Log level: BrakeTestResult.LogLevel.Info

Example

A brake test is executed for axis A2. If the brake test is aborted, this is indicated by a corresponding output signal. If the brake test is fully executed, a message containing the measured holding torque is generated and the test results are polled. Depending on whether the measured holding torque is too low, within the tolerance range or in the ideal range, a corresponding output is also set in each case.

```
private LBR lbr_iiwa_7_R800_1;
private BrakeTestIOGroup brakeTestIOs;
...
run{
    ...
    BrakeTest brakeTest = new BrakeTest(lbr_iiwa_7_R800_1);
    int indexA2 = 1;
    try {
        BrakeTestResult resultA2 = brakeTest.execute(indexA2);

        double measuredTorque = resultA2.getMeasuredTorque();
        getLogger().info("Measured torque for A2: " + measuredTorque);

        BrakeTestResult.State state = resultA2.getState();
        if(state == BrakeTestResult.State.BrakeError)
            brakeTestIOs.setA2_BrakeError(true);
        else if(state == BrakeTestResult.State.BrakeWarning)
            brakeTestIOs.setA2_BrakeWarning(true);
        else if(state == BrakeTestResult.State.BrakeReady)
            brakeTestIOs.setA2_BrakeOK(true);
    } catch (BrakeTestException e) {
        brakeTestIOs.setBrakeTest_Exception(true);
        e.printStackTrace();
    }
}
```

8.4 Performing a brake test

NOTICE

If a brake is identified as being defective and the drives are deactivated, the robot may sag.

Description

If the template for the brake test application is adopted unchanged, the torques measured during a motion sequence are first evaluated for each axis, and the maximum torque for each axis is determined. The result of the evaluation is

displayed on the smartHMI. The brakes are then tested one after the other, starting with axis A1. The brake test results are displayed individually for each axis on the smartHMI.

Precondition

- No persons or objects are present within the motion range of the robot.
- Program run mode **Continuous** (standard mode)
- AUT mode



The torque value determination can also be executed in T1 or T2 mode.

- The robot is at operating temperature.

Procedure

- Select and start the brake test application.

8.4.1 Evaluation results of the maximum absolute torques (display)

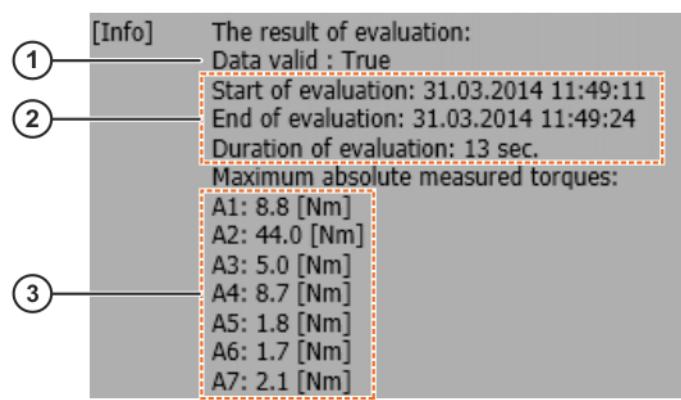


Fig. 8-4: Results of an evaluation of the maximum absolute torques

Item	Description
1	Validity Indicates whether the determined data are valid. The data are valid if no errors occur during command processing.
2	Time indications Start time, end time and overall duration of the evaluation.
3	Determined data The maximum absolute torque determined from the evaluation is displayed for each axis.

8.4.2 Results of the brake test (display)

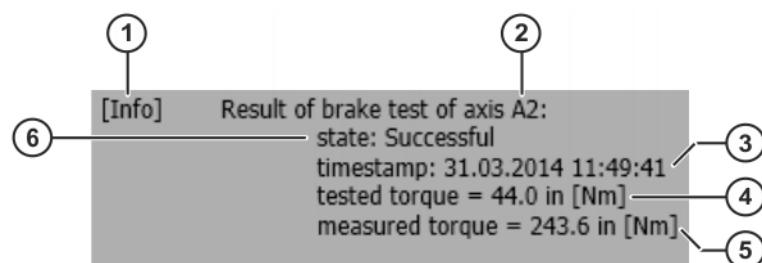


Fig. 8-5: Results of a brake test for axis A2

Item	Description
1	<p>Log level</p> <p>Depending on the results of the brake test, the message is generated with a specific log level.</p> <ul style="list-style-type: none"> ■ Info: The brake test has been executed successfully. ■ Warning: The brake test has been executed successfully, but the measured value does not lie in the ideal range. ■ Error: The brake test could not be executed or has failed.
2	Tested axis
3	Time stamp
	Time stamp at which the brake test was started for the axis.
4	Holding torque to be tested
5	Measured holding torque
6	<p>Result of the brake test</p> <ul style="list-style-type: none"> ■ Untested: The brake test could not be executed or was aborted during execution due to faults. ■ Unknown: The brake test could not be executed because not enough torque could be generated (e.g. due to excessive friction). ■ Failed: The brake test has failed. The measured holding torque falls below the holding torque to be tested. The brake is defective. ■ Warning: The measured holding torque is less than 5% above the holding torque to be tested. The brake has reached the wear limit and will soon be identified as defective. ■ Maximum unknown: The brake test could not be executed, as it was not possible to test against the maximum brake holding torque. Possible cause: The value is missing from the motor data. ■ Excessive: The measured holding torque is greater than the maximum brake holding torque. Stopping using the brake can cause damage to the machine. ■ Successfull: The measured holding torque exceeds the holding torque to be tested by more than 5 %. The brake is fully operational.



If the functional capability of a brake is not guaranteed, the robot can sag. If the brake test shows that the brake on at least one axis of the LBR iiwa cannot apply the desired holding torque, the robot must be taken out of operation immediately.

9 Project management

9.1 Sunrise projects – overview

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.

- Applications

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 applications.

- Runtime data

Runtime data are all the data which are used by the 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 application.

Sunrise projects are created and managed with KUKA Sunrise.Workbench.

(>>> 5.3 "Creating a Sunrise project with a template" Page 46)

There may only be 1 Sunrise project on the robot controller. This is transferred from Sunrise.Workbench to the robot controller by means of project synchronization.

(>>> 9.4 "Synchronization of projects" Page 122)

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 by default is located at 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 with respect to its parent frame.

The position of a frame with respect 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

9.2.1 Creating a new frame

Description

In Sunrise.Workbench, created 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. On the smartHMI, the frames can be taught in order to determine the position of the frames in space. Taught frames can be addressed manually.

(>>> 6.9 "Teaching and manually addressing frames" Page 75)

Procedure

1. Select the project in the **Package Explorer**.
2. Right-click on the desired parent frame in the **Application data** view and select **Insert new empty 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 **Properties** view.

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.

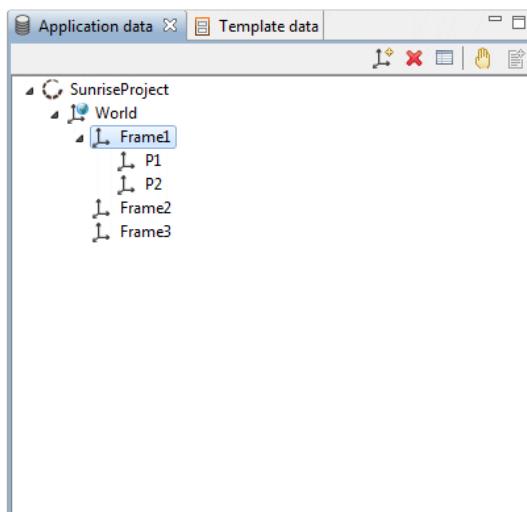
Example

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 Frame2 and are located one level below it.

9.2.2 Defining a frame as a “base for jogging”

Description

Frames can be marked as a base for jogging in the **Application data** view.



To simplify selection of a frame for jogging, it is advisable, in particular for these frames, to assign clear and descriptive names.

Only frames marked in this way can be selected on the smartHMI as a base for jogging after synchronization of the project.

(>>> 6.7.1 "Jogging options" window" Page 70)

- Procedure**
- Right-click on the desired frame and select **Use as Jogging Base** from the context menu.
- Alternative:
- Select the frame and click on the **Use as Jogging Base** hand icon.
- The frame is marked with a hand icon.

Example

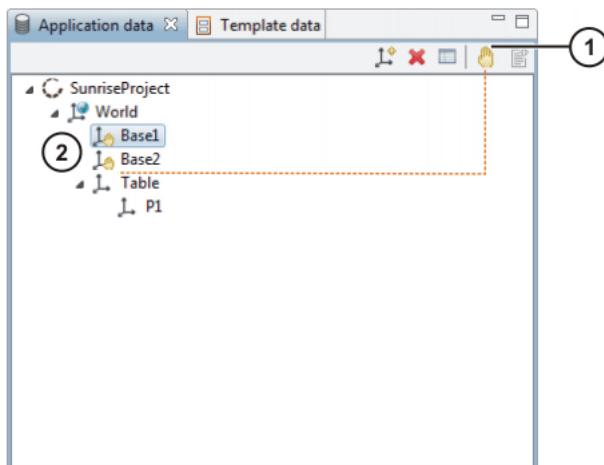


Fig. 9-2: Frame as a base for jogging

- 1 **Use as Jogging Base** hand icon
- 2 Base1 and Base2 available as a jogging base

9.2.3 Moving frames

- Description**
- A frame can be moved in the **Application data** view 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. Since frames are used via this path in the source code of applications, the path specification must be corrected accordingly in the applications.

- Procedure**
1. Click on the desired frame 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 frames

- Description**
- Frames can be removed from the frame tree in the **Application data** view. 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. Since frames are used via this path in the source code of applications, the path specification must be corrected accordingly in the applications.

- **Delete parent and child frames:** Deletes the selected frame and all subordinate frames.

Procedure

1. Right-click on the frame to be deleted and select **Delete** from the context menu. A frame without child elements is deleted immediately.
2. If the frame has child elements, the system asks whether these should also be deleted. Select the desired option.
3. Only with the **Move children to parent** option: 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 and modifying the properties of a frame



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, and it applies to 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).

Procedure

1. Select the frame in the **Application data** view. The properties of the frame are displayed in the **Properties** view.
2. If required, display the properties by category using the  (**Display categories**) button.
3. Enter the new value in the **Value** box of the property and confirm with the **Enter** key.



For physical variables, the value can be entered with the unit. If this is compatible with the preset unit, the value is converted accordingly (e.g. cm into mm or ° into rad). If no unit is entered, the preset unit is used.

Description

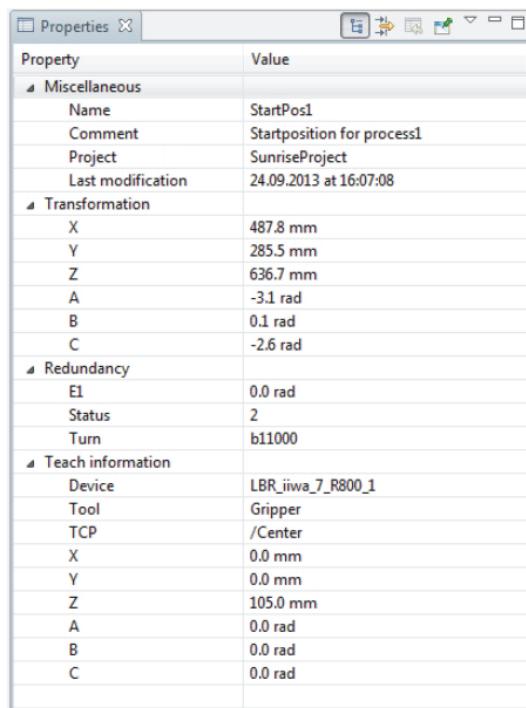


Fig. 9-3: Frame properties by category

The categories contain the following information:

Category	Description
Miscellaneous	General information <ul style="list-style-type: none"> ■ Name of the frame ■ Comment (optional) ■ Corresponding project ■ Date and time of the last modification
Transformation	Transformation data <ul style="list-style-type: none"> ■ Translational offset of the frame relative to its parent frame (X, Y, Z) ■ Rotational offset of the frame relative to its parent frame (A, B, C)
Redundancy	Redundancy information <ul style="list-style-type: none"> ■ Value of the redundancy angle (E1) ■ Status ■ Turn
Teach information	Information about the taught frame <ul style="list-style-type: none"> ■ Robot used (device) ■ Tool used ■ Frame path to TCP used ■ Translational offset of the TCP relative to the origin frame of the tool (X, Y, Z) ■ Rotational offset of the TCP relative to the origin frame of the tool (A, B, C)

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	<ol style="list-style-type: none">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

Tools and workpieces are created and managed 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

Properties:

- 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). By default, 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 workpiece (root) or to one of its child elements.

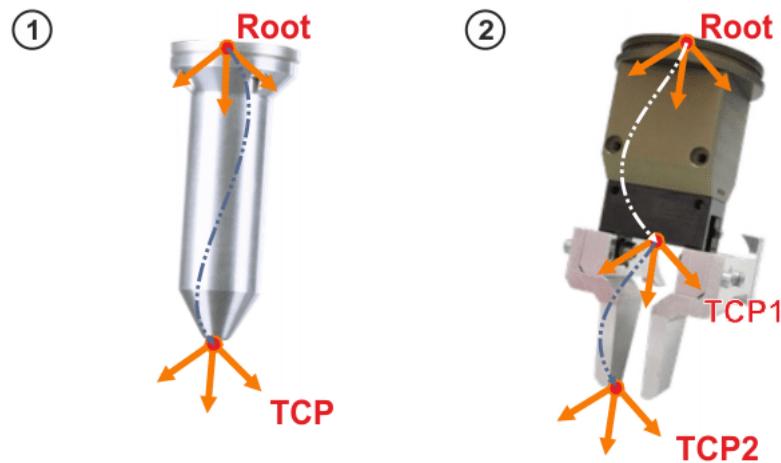


Fig. 9-4: Examples of TCPs of tools

1 Guiding tool with 1 TCP

2 Gripper with 2 TCPs

i 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.

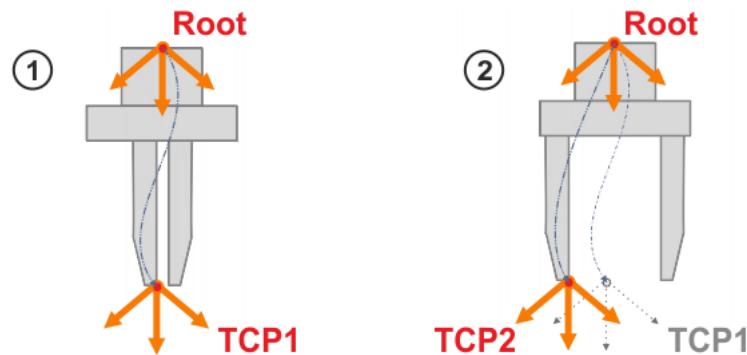


Fig. 9-5: Static TCP on a gripper

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.

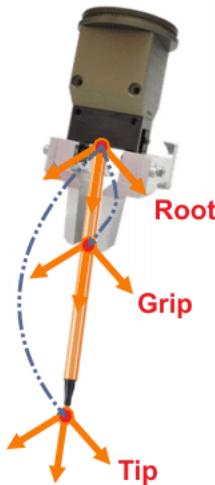


Fig. 9-6: Examples of frames of workpieces

9.3.3 Creating a tool or workpiece

Description In Sunrise.Workbench, created tools and workpieces are project-specific and can be used in every robot application of the project.
The created tools can be selected in the jogging options on the smartHMI after the project is synchronized.
(>>> 6.7.1 "Jogging options" window" Page 70)

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Template data** view, open the list of object templates.
3. To create a tool, right-click on the object type **Template Group for 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 **Template Group for Workpieces** and select **Insert new workpiece** from the context menu. The object template for the workpiece is created.
5. The system automatically generates an object name. It is advisable to change the name in the **Properties** view.
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 **Properties** view.
(>>> 9.3.6 "Load data" Page 120)

9.3.4 Creating a frame for a tool or workpiece

Description Each of the frames 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 created frames of tools can be selected as the TCP for Cartesian jogging on the smartHMI.
(>>> 6.7.1 "Jogging options" window" Page 70)

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Template data** view, open the list of object templates.
3. Right-click on the object template and select **Insert new empty frame** from the context menu. The frame is created.

In the upper hierarchy level, the parent frame of the created frame is the origin frame of the object.

4. To insert a new frame under an existing frame of the object, right-click on this parent frame and select **Insert new empty frame** from the context menu. The frame is created.
5. The system automatically generates a frame name. It is advisable to change the name in the **Properties** view.
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.
6. In the **Properties** view, enter the transformation data of the frame with respect 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 with reference to the parent frame

9.3.5 Defining a default motion frame

Description

If a tool or workpiece has a frame with which a large part of the motions must be executed, this frame can be defined as the default frame for motions.

Defining an appropriate default frame for a tool or workpiece simplifies the motion programming.

(>>> 15.9.4 "Moving tools and workpieces" Page 225)

If no default frame is defined, the origin frame of the tool or workpiece is automatically used as the default frame for motions.

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Template data** view, select the object type **Template Group for Tools** or **Template Group for Workpieces**.
3. Select the desired tool or workpiece.
4. Right-click on the desired frame and select **Default Motion Frame** from the context menu.

Alternative:

Select the frame and click on the **Default Motion Frame** icon.

The frame is marked as the default motion frame.

Example

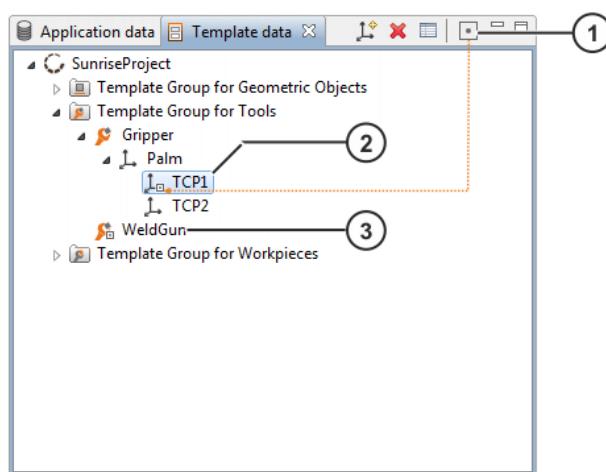


Fig. 9-7: Default motion frame

1 Default motion frame icon

- 2 Default motion frame of the **Gripper** tool: TCP_1
- 3 Default motion frame of the **WeldGun** tool: origin frame

9.3.6 Load data

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 must 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 The robot must not be operated with incorrect load data or unsuitable loads. Failure to observe this precaution may result in severe injuries or considerable damage to property, e.g. because braking the robot takes too long due to incorrect 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.
(>>> 7.2 "Determining tool load data" Page 92)

9.3.6.1 Manually entering load data

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Template data** view, open the list of object templates.
3. Select the desired tool or workpiece.
4. In the **Properties** view, enter the load data:
 - **Mass**: mass of the tool or workpiece
 - Boxes **MS X**, **MS Y**, **MS Z**: position of the center of mass relative to the origin frame
 - Boxes **MS A**, **MS B**, **MS C**: orientation of the principal inertia axes relative to the origin frame
 - Boxes **jX**, **jY**, **jZ**: mass moments of inertia
jX is the inertia about the X axis of the coordinate system that is rotated relative to the origin frame by A, B and C. **jY** and **jZ** are the analogous inertia values about the Y and Z axes.

9.3.7 Safety-oriented tool

Description

A maximum of 1 safety-oriented tool may be defined in a Sunrise project, with up to 6 configurable spheres modeled around it.

The properties of the safety-oriented tool are relevant for the following configurable safety functions:

- Monitoring of Cartesian workspaces and protected spaces
The spheres are monitored against the space limits.
(>>> 13.6.8 "Monitoring spaces" Page 161)
- Monitoring of the translational Cartesian velocity
The velocity of the sphere center points is monitored.
(>>> 13.6.7 "Velocity monitoring functions" Page 160)
- Collision detection and TCP force monitoring
Only correctly specified load data ensure the accuracy of these monitoring functions. The load data of the safety-oriented tool, in particular the mass and center of mass of the tool, must be configured. In the case of tools with comparatively high moments of inertia ($> 0.1 \text{ kg} \cdot \text{m}^2$), these data must also be specified in order to ensure the accuracy of these monitoring functions.

Just like any other tool, a safety-oriented tool can have any number of frames. In order to configure the monitoring spheres, suitable frames must be defined as safety-oriented frames. The origin of a safety-oriented frame is the center of the sphere. The radius of the sphere is defined in the frame properties.

The safety-oriented tool is transferred to the robot controller by means of synchronization and activated once the robot controller has been rebooted, i.e. it is permanently active for the safety controller, irrespective of which tool is used in the application or set in the jogging options.

Example

For a safety-oriented gripper, 3 monitoring spheres are configured.

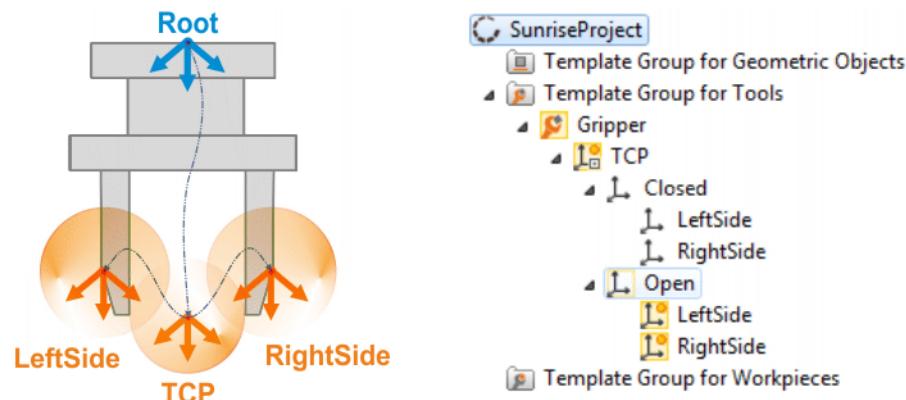


Fig. 9-8: Safety-oriented gripper

9.3.7.1 Defining a safety-oriented tool

Precondition

- Tool and corresponding frames have been created.
- If the AMFs **Collision detection** and **TCP force monitoring** are being used: the correct load data of the tool, particularly the mass and center of mass of the tool, are known.

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Template data** view, select the tool that is to be safety-oriented.
3. Define the properties of the safety-oriented tool in the **Properties** view.
 - If they have not yet been defined, enter the load data of the tool.
Tools with load data outside the specified range of values cannot be used as safety-oriented tools.
(>>> "Tool" Page 122)
 - Set the property **Safety related** to **Yes**.
The tool icon in the **Template data** view is highlighted in yellow.
4. In the **Template data** view, select the frame that is to be safety-oriented.

5. Define the properties of the safety-oriented frame in the **Properties** view.
 - Enter the radius of the monitoring sphere.
 - If they have not yet been defined, enter the 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.
 (=> "Frames" Page 122)

 - Set the property **Safety related** to **Yes**.

The frame icon in the **Template data** view is highlighted in yellow and marked with a sphere symbol.
6. Repeat steps 4 to 5 to define further safety-oriented frames.



Alternatively, to mark a tool or frame as safety-oriented:

- Right-click on the tool or frame in the **Template data** view and select **Safety related** from the context menu.

Tool

Load data of the safety-oriented tool:

Parameter	Description
Mass	Mass of the safety-oriented tool <ul style="list-style-type: none"> ■ ≤2,000 kg
MS X, MS Y, MS Z	Position of the center of mass relative to the origin frame of the safety-oriented tool <ul style="list-style-type: none"> ■ -10,000 mm ... +10,000 mm
MS A, MS B, MS C	Orientation of the principal inertia axes relative to the origin frame of the safety-oriented tool <ul style="list-style-type: none"> ■ Any
jX, jY, jZ	Mass moments of inertia of the safety-oriented tool <ul style="list-style-type: none"> ■ 0 kg*m²... 1,000 kg*m²

Frames

Properties of safety-oriented frames:

Parameter	Description
Radius	Radius of the sphere on the safety-oriented frame <ul style="list-style-type: none"> ■ 25 mm ... 10,000 mm
X, Y, Z	Offset of the safety-oriented frame along the axes of the parent frame <ul style="list-style-type: none"> ■ -10,000 mm ... +10,000 mm
A, B, C	Orientation of the safety-oriented frame relative to the parent frame <ul style="list-style-type: none"> ■ Any

9.4 Synchronization of projects

Overview

In the synchronization of projects, the 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:

- The project only exists in Sunrise.Workbench.

- The project on the robot controller is replaced by another project.
- There are different versions of the project:
 - When the project data is modified in Sunrise.Workbench only
 - When the project data is modified on the robot controller only
 - When the project data is modified on both sides

9.4.1 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	<ul style="list-style-type: none"> ■ Network connection to the robot controller ■ The project to be transferred has at least one application. ■ The system software is now installed. (>>> 10.2 "Installing the system software" Page 127) ■ The installed system software is compatible with the station configuration of the project to be transferred.
Procedure	<p>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.</p> <ol style="list-style-type: none"> 1. Select the project to be transferred in the Package Explorer. 2. Click on the Synchronize project button in the toolbar. The system scans the 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.

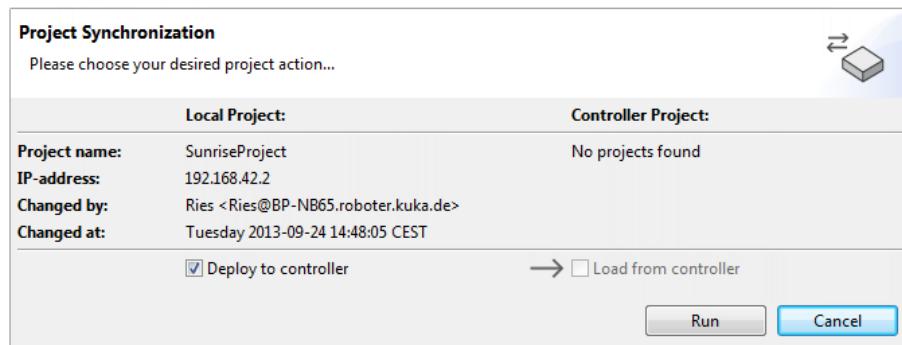


Fig. 9-9: Transferring the project to the controller

4. Click on **Run**. The project is transferred to the robot controller.
5. If the safety configuration or I/O configuration is modified, the robot controller must be rebooted.
Confirm the reboot prompt with **OK**. The robot controller is rebooted.
6. If the safety configuration is modified, activate this on the robot controller.
(>>> 13.5 "Activating the safety configuration on the robot controller" Page 155)

The transferred project is now active on the controller. All created project data are available.

9.4.2 Updating the project on the robot controller or in Sunrise.Workbench

Description	The procedure described here applies if the same project exists on both sides but in different versions.
Precondition	<ul style="list-style-type: none"> ■ Network connection to the robot controller

- No application is running on the robot controller.

Procedure

1. Select the project to be transferred in the **Package Explorer**.
2. Click on the **Synchronize project** button in the toolbar. The system scans the controller for existing project data. If the scan fails, the cause of the error is displayed in a message.
3. If the project is identical to the project in Sunrise.Workbench, no synchronization is necessary. The process is then automatically aborted.
If the scan is successful, the **Project Synchronization** window opens.

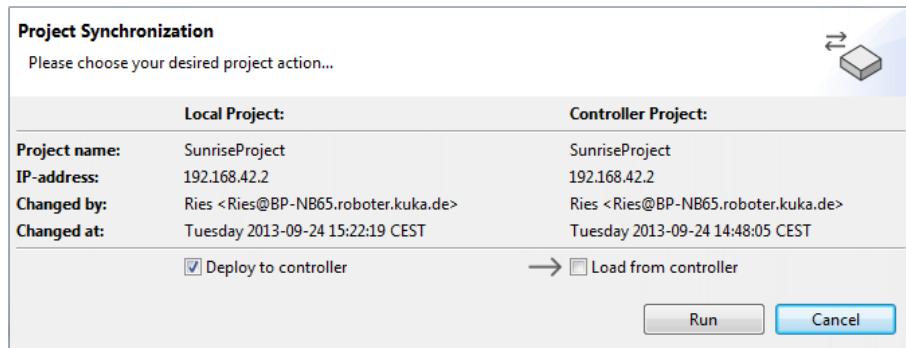


Fig. 9-10: Updating a project

Information regarding both projects is displayed here. The direction of synchronization is set by default to transfer the more current 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:

- If the **Deploy to controller** check box is activated (check mark set):
The project is transferred from Sunrise.Workbench to the robot controller.
 - If the **Load from controller** check box is activated (check mark is set):
The project is transferred from the robot controller to Sunrise.Workbench.
4. Select the desired check box and click on **Run**. The project is transferred. If the older version is to be transferred, a warning is displayed.
 5. Only when transferring to the robot controller:
 - If the safety configuration or I/O configuration is modified, the robot controller must be rebooted.
Confirm the reboot prompt with **OK**. The robot controller is rebooted.
 - If the safety configuration is modified, activate this on the robot controller.
(>>> 13.5 "Activating the safety configuration on the robot controller"
Page 155)

9.5 Loading the project from the robot controller

Description

It is also possible to load a project 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 **Controller IP Address** box.



The IP address of the robot controller can be displayed on the smartHMI. (**>>> 6.11.6 "Displaying the IP address and software version"** Page 89)

3. Activate the **Load project from controller** radio button.
4. Click on **Next >**. The system checks whether there is a project on the controller.
5. If there is a project on the 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 added to the **Package Explorer**.

10 Station configuration and installation

10.1 Opening the station configuration

- Procedure**
- Double-click on the **StationSetup.cat** file in the **Package Explorer**.
The file contains the station configuration of the project. This 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 components of the station. The components to be installed as well as their version can be selected. The components available for selection depend on the specific topology. |
| Configuration | The Configuration tab displays the configuration of the robot controller. The configuration can be changed. |
| Installation | The system software is installed on the robot controller via the Installation tab. |

10.2 Installing the system software

- Precondition**
- The station configuration is completed.
 - Network connection to the robot controller
- Procedure**
1. Select the **Installation** tab.
 2. By default, the **Installation events** window displays the warnings and errors which occur during installation: The **Show only warnings and errors** check box is activated (check mark is set).
To display all events which occur during installation, deactivate the **Show only warnings and errors** check box (uncheck the box).
 3. Click on **Install**. The installation is prepared and the **Installation** window opens. The **Configured IP** box is marked in color:
 - Marked in green: The network connection to the robot controller has been established; installation is possible.
 - Marked in red: The network connection to the robot controller could not be established. Possible causes include: the network cable is not connected correctly or the configured IP address does not match the actual address.

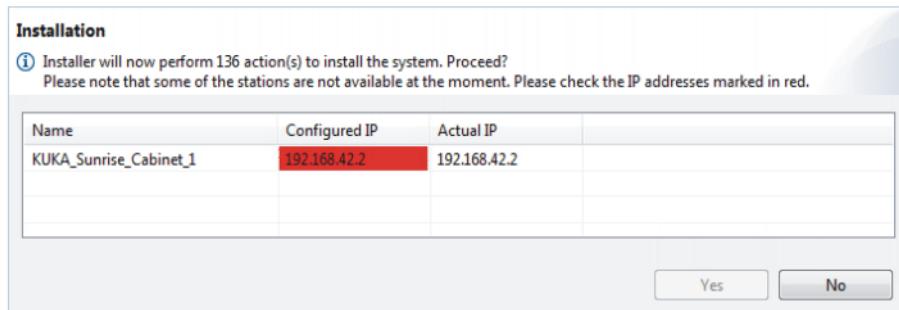


Fig. 10-1: The robot controller cannot be reached

4. Only if the **Configured IP** box is marked in red:
 - If the configured IP address of the robot controller matches the actual address, there is no network connection to the robot controller. Establish a 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.



During installation, the safety configuration is transferred to the robot controller but is not yet activated. In order to be able to move the robot, the safety configuration must be activated via the smartHMI.
(>>> 13.5 "Activating the safety configuration on the robot controller"
Page 155)

11 Bus configuration

11.1 Configuration and I/O mapping in WorkVisual – overview

Step	Description
1	Install the Sunrise option package in WorkVisual.
2	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. (>>> 11.2 "Creating a new I/O configuration" Page 129) (>>> 11.3 "Opening an existing I/O configuration" Page 130)
3	Only necessary if devices are used for which no device description files have yet been imported: 1. Close the WorkVisual project. 2. Import the required device description files. 3. Reopen the WorkVisual project.
4	Configure the field bus.
5	Create the Sunrise I/Os and map them. (>>> 11.4 "Creating Sunrise I/Os" Page 130) (>>> 11.5.3 "Mapping Sunrise I/Os" Page 136)
6	Export the I/O configuration to the Sunrise project. (>>> 11.6 "Exporting the I/O configuration to the Sunrise project" Page 136)
7	Transfer the I/O configuration to the robot controller (Synchronize Project) and reboot the robot controller. (>>> 9.4 "Synchronization of projects" Page 122)



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.



Information about the specific configuration of field buses supported by Sunrise can be found in the corresponding field bus documentation.

11.2 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.

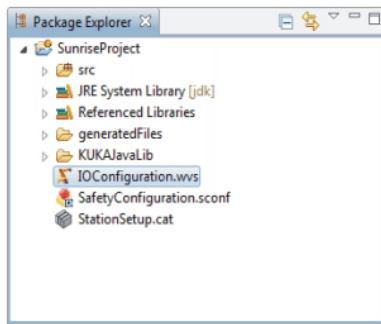


Fig. 11-1: Project with I/O configuration

11.3 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.

11.4 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.
(>>> 11.5.1 "I/O Mapping window" Page 134)
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.
(>>> 11.4.1 "'Create I/O signals' window" Page 131)
4. Create an I/O group and inputs/outputs within the group.
(>>> 11.4.2 "Creating an I/O group and inputs/outputs within the group" Page 132)
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.

(>>> 11.5.3 "Mapping Sunrise I/Os" Page 136)

11.4.1 “Create I/O signals” window

Overview

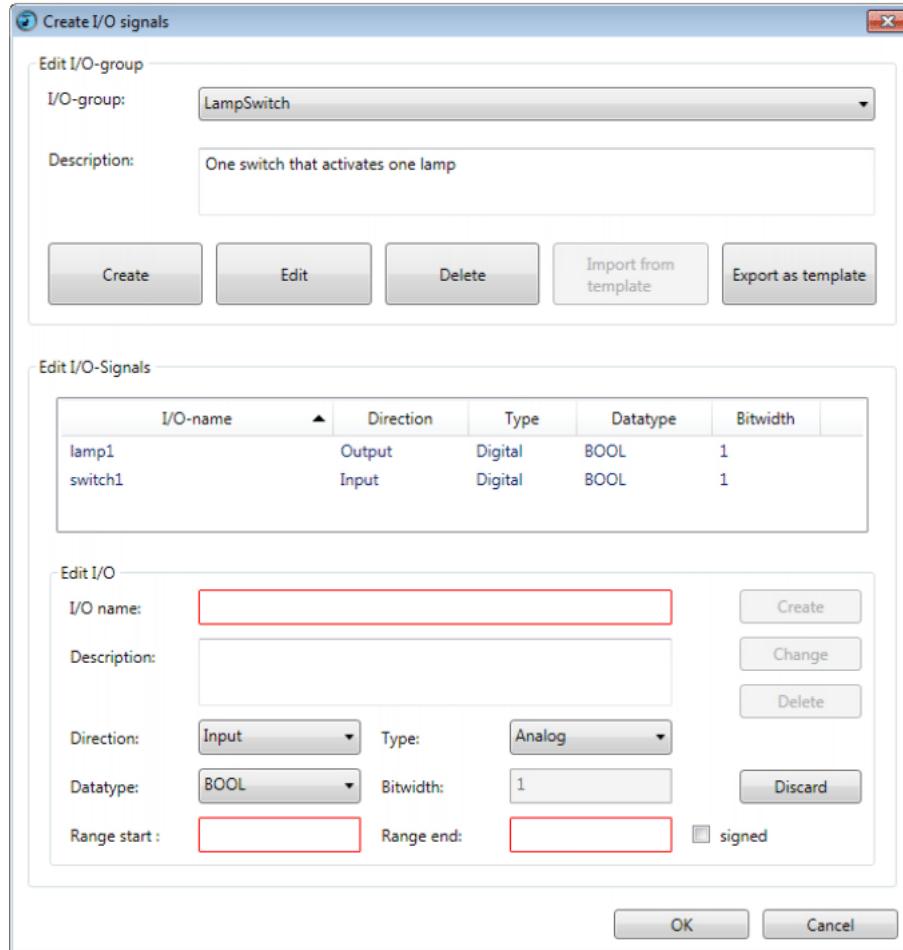


Fig. 11-2: “Create I/O signals” window

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 their properties defined:

Property	Description
I/O name	Enter the name of the input or output.
Description	Enter a description for the input or output (optional).
Direction	Specify whether the signal is an input or output.
	■ Input, Output

Property	Description
Type	Specify whether the signal is an analog or digital signal. ■ Analog, Digital
Datatype	Select the data type of the signal. 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: ■ integer, long, double, boolean
Bitwidth	Enter the number of bits that make up the signal. With the data type BOOL, the bitwidth is always 1. Note: The value must be a positive integer which does not exceed the maximum permissible length of the selected data type.
Range start	Only relevant for analog inputs/outputs!
Range end	Enter the start and end of the range for the analog value and if applicable activate the Signed check box (set the check mark).
Signed	Note: These values can generally be found in the data sheet of the field bus module. The range start must be lower than the range end. It is also possible to enter decimal values.

11.4.2 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.

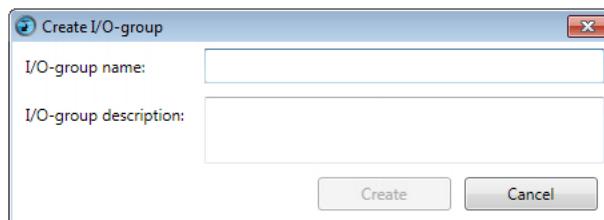


Fig. 11-3: Create I/O group

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 131)
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.

11.4.3 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**.

11.4.4 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.

11.4.5 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 131)



All the changes can be discarded by clicking on the **Discard** button.

4. Click on **Change**. The changes are saved.

11.4.6 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**.

11.4.7 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

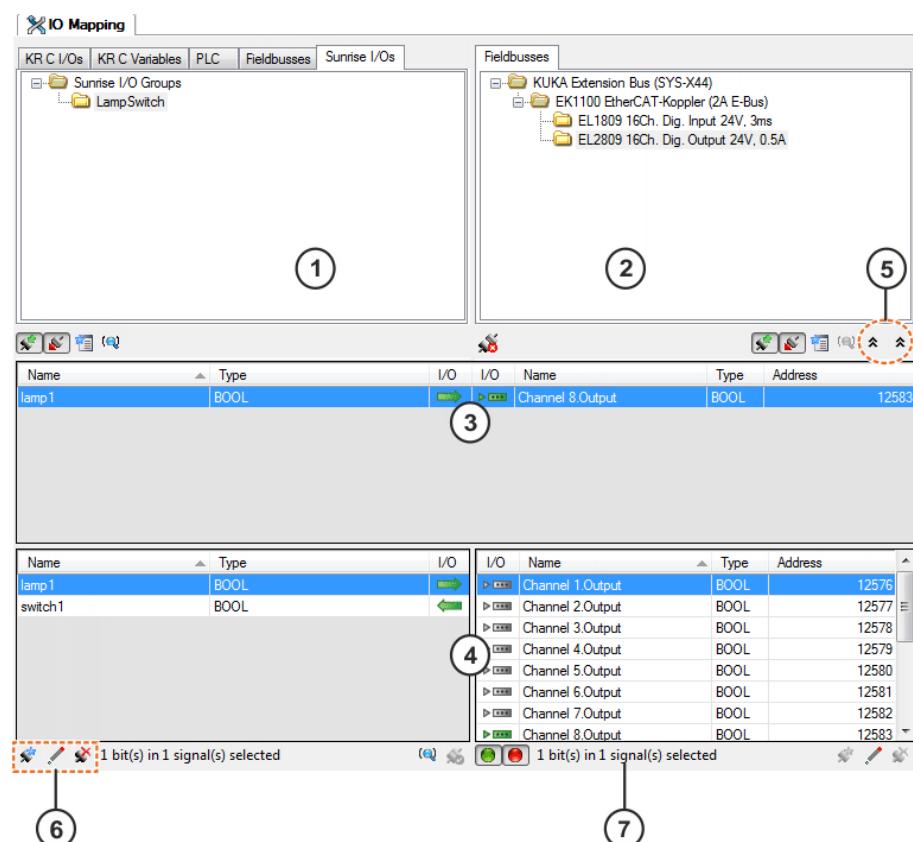
1. In the **Edit I/O group** area, select the I/O group that is to be exported as a template.
 2. Click on **Export as template**. The **Save I/O group as template** window is opened.
 3. Enter a name for template.
- Info:** If a template with the same name already exists in the Sunrise project, it will be overwritten during the export operation.
4. Enter a description for the template (optional).
 5. Click on **Export**. The template is saved.

11.4.8 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.

11.5 Mapping the bus I/Os**11.5.1 I/O Mapping window****Overview****Fig. 11-4: "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. (>>> 11.5.3 "Mapping Sunrise I/Os" Page 136)
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.

11.5.2 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. (>>> 11.4.1 "“Create I/O signals” window" Page 131) 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.

Button	Name / description
	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.

11.5.3 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

1. 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.
2. 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.
3. 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.

11.6 Exporting the I/O configuration to the Sunrise project**Description**

Exporting the I/O configuration automatically creates and inserts a Java package in the source folder of the Sunrise project. This package contains the Java classes and methods required for programming, in order to enable read and write access to the inputs/outputs.

The project structure after exporting the I/O configuration is described here:

(>>> 15.10 "Inputs/outputs" Page 226)



The first time the I/O configuration is exported, the folder **generated-Files** is created in the Sunrise project. This is used by the system and must not be used for saving files created by the user.

Precondition

- The robot controller has been set as the active controller.

Procedure

1. Select the menu sequence **File > Import / Export**. The import/export wizard for files opens.
2. Select **Export I/O configuration to 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**.



In order to transfer the I/O configuration and mapping to the robot controller, the Sunrise project must be synchronized.

(>>> 9.4.1 "Transferring the project to the robot controller"

Page 123)

12 External control

12.1 Configuring external control

Description If the processes of the station are to be controlled by a higher-level controller in Automatic mode, the Sunrise project on the robot controller must be configured for external control.

Every Sunrise project that is configured for external control uses a default application that is automatically selected when switching to Automatic mode. The default application of the externally controlled project cannot be deselected again in Automatic mode.

The default application cannot be started using the Start/pause key on the smartPAD, but only via an external input.

The robot controller and the higher-level controller communicate via pre-defined input/output signals:

- The higher-level controller can start, pause and resume the default application via the input signals.
- The output signals can be used to provide information about the status of the default application and the station to the higher-level controller.

Overview

The following steps are required in order to be able to use external control:

Step	Description
1	Configure and map inputs/outputs for communication with the higher-level controller in WorkVisual. (>>> 12.1.1 "External control inputs" Page 140) (>>> 12.1.2 "External control outputs" Page 140)
2	Export I/O configuration from WorkVisual to Sunrise.Workbench.
3	Create the default application for external control of the Sunrise project in Sunrise.Workbench.
4	Configure external control of the Sunrise project in Sunrise.Workbench. (>>> 12.1.3 "Configuring external control in the project properties" Page 141)
5	Transfer the Sunrise project to the robot controller by means of synchronization.



The physical inputs/outputs used for communication with the higher-level controller must not be multiply mapped.

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
- The input App_Start has been configured.
- If configured: the input App_Enable has the level HIGH (TRUE).
- The motion enable signal is present.

12.1.1 External control inputs

App_Start	The input App_Start is absolutely vital for an externally controlled project. The default application is started and resumed in Automatic mode by the higher-level controller by means of a rising edge of the input signal (change from FALSE to TRUE).
App_Enable	The input App_Enable can optionally be configured. The default application can be paused by the higher-level controller in Automatic mode by means of a LOW level at this input.
System response	The input App_Enable has a higher priority than the input App_Start. If the input App_Enable is configured, the default application can only be started if there is a high-level signal at App_Enable.

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. Application is then paused.
Any	TRUE --> FALSE	Running	Application is paused.
Any	TRUE --> FALSE	Repositioning	Application is paused.

12.1.2 External control outputs

The configuration of these outputs is optional.

AutExt_Active	A high level at this output signals to the higher-level controller that Automatic mode is active and the project on the robot controller can be controlled externally.
AutExt_AppReadyToStart	A high level at this output signals to the higher-level controller that the default application is ready to start (status Selected or Motion Paused).
DefaultApp_Error	A high level at this output signals to the higher-level controller that an error occurred when the default application was run (status Error).
Station_Error	A high level at this output signals to the higher-level controller that the station is in an error state. The error state is active if one of the following conditions is met: <ul style="list-style-type: none"> ■ Motion enable signal 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.

12.1.3 Configuring external control in the project properties

Procedure

1. Right-click on the desired Sunrise project in the **Package Explorer** and select **Sunrise > Change project settings** from the context menu.
The **Properties for Project** window opens.
2. Select **Sunrise > External Control** in the directory structure 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.
 - Activate the check box **Project will be controlled externally** (set check mark).
 - In the **Default Application** area, select the desired default application.
 - In the area **Configuration inputs**, configure the inputs for the external communication.
 - **IO Group** column: select the I/O group containing the desired input.
 - **Boolean Input** column: select input from the I/O group.
 - Optionally: In the area **Configuration outputs**, configure the signal outputs for the project.
 - **IO Group** column: select the I/O group containing the desired output.
 - **Boolean Output** column: select output from the I/O group.
4. Click on **Apply** to apply the settings and close the window with **OK**.



The default application is indicated in the **Package Explorer** by the following icon:



If the default application is renamed, the icon is no longer displayed and the application must be selected as the default application once again.

(>>> 12.2 "Selecting a robot application as the default application"

Page 142)

Description

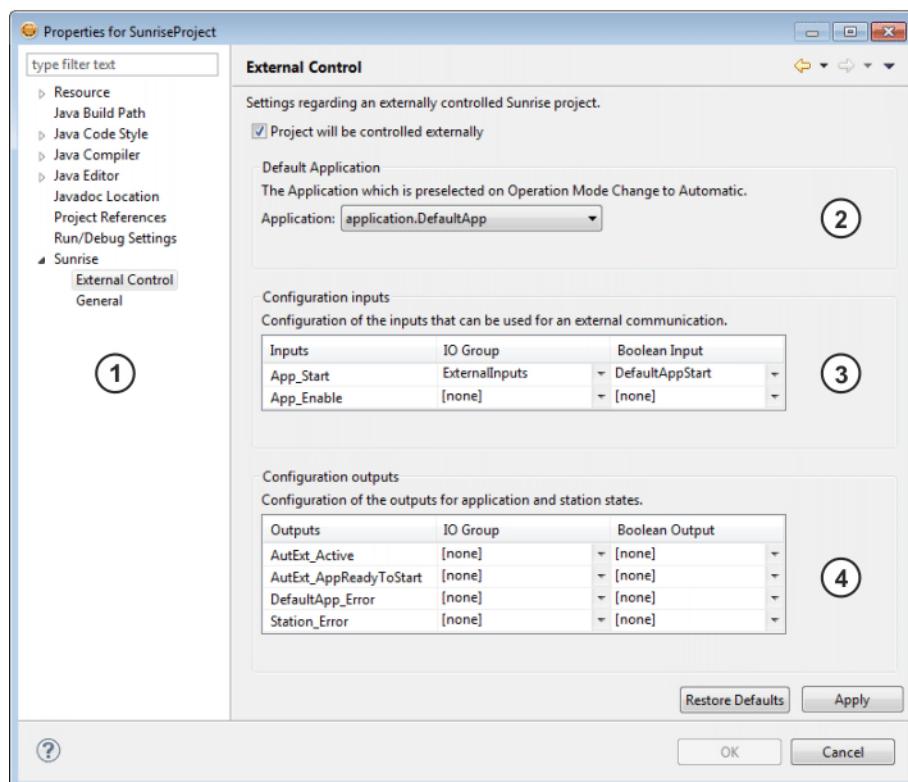


Fig. 12-1: Minimal configuration: external control of a project

Item	Description
1	Project properties directory structure
2	Default Application area All robot applications of the project are available for selection.
3	Configuration inputs area All I/O groups of the I/O configuration of the project and the corresponding inputs are available for selection. The input App_Start is absolutely vital for external control of a project.
4	Configuration outputs area All I/O groups of the I/O configuration of the project and the corresponding outputs are available for selection.

12.2 Selecting a 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** and select **Sunrise > Set as default application** from the context menu.

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

Example**Fig. 12-2: Default application MainApp.java****12.3 Defining the signal outputs for a project that is not externally controlled****Description**

The predefined output signals for communication with the higher-level 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 of the project.

Precondition

- The I/O configuration of the project contains the outputs configured and mapped in WorkVisual.
(>>> 12.1.2 "External control outputs" Page 140)

Procedure

1. Right-click on the desired Sunrise project in the **Package Explorer** and select **Sunrise > Change project settings** from the context menu.
The **Properties for Project** window opens.
2. Select **Sunrise > General** in the directory structure in the left area of the window.
3. Make the general settings for the project in the right-hand area of the window.
 - If application statuses are to be signaled, select the desired default application in the **Default Application** area.
 - In the area **Configuration outputs**, configure the signal outputs for the project.
 - **IO Group** column: select the I/O group containing the desired output.
 - **Boolean Output** column: select output from the I/O group.
4. Click on **Apply** to apply the settings and close the window with **OK**.

13 Safety configuration

13.1 Overview

The safety configuration defines the safety-oriented functions in order to integrate the industrial 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 the form of a table. 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.



In the case of incomplete start-up of the system, additional substitute measures for minimizing risk must be taken and documented, e.g. installation of a safety fence, attachment of a warning sign, locking of the main switch, etc. 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 system integrator must verify that the safety configuration sufficiently reduces risks during collaborative operation (HRC). It is advisable to perform this verification in accordance with the information and instructions for operating collaborative robots in ISO/TS 15066.

13.2 Safety concept

Overview

The safety configuration must implement all safety functions which are required to operate the industrial robot. Each safety function is described by a line in the table in which it is configured. The individual safety functions are evaluated separately from one another.

The smallest unit of a safety function is designated as the Atomic Monitoring Function (AMF). Each safety function consists of up to 3 AMFs. Each AMF supplies an elementary, safety-relevant piece of information, for example if a safe input is set or if the Automatic operating mode is selected. In addition, the safety function defines a reaction which is triggered in case of violation.

To simplify the diagnosis in case a safety function is violated, the safety function can be assigned a category suitable for the AMFs used and the reaction.

Functional principle

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.

For example, the AMF **EMERGENCY STOP smartPAD** is violated if the EMERGENCY STOP device on the operator panel is pressed.

For a safety function, up to 3 AMFs are linked to one another. The safety function also defines a reaction. This is triggered if all the AMFs of the safety function are violated. Here, 2 AMFs can be interpreted as activation functions and 1 AMF as a monitoring function.

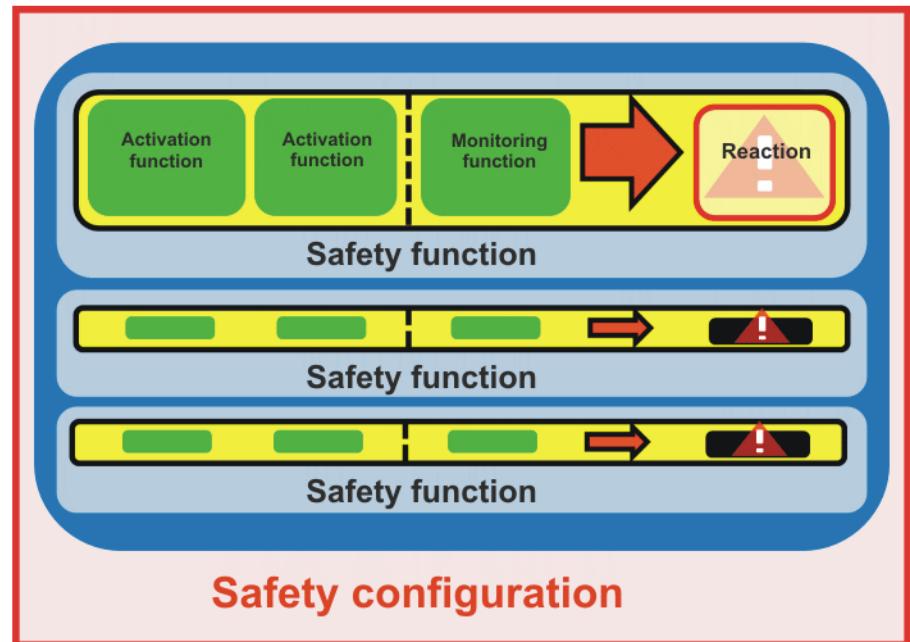


Fig. 13-1: Structure of a safety function

i The AMFs can be distributed in any way to the two activation functions and the monitoring function of a safety function. The distinction only serves to simplify the structure of user-defined safety functions.

Safety functions can be structured in different ways, due to the flexible assignment of the activation and monitoring functions:

Safety function	Description
Only monitoring function	<p>Structure:</p> <ul style="list-style-type: none"> ■ The safety function contains a single AMF, which is defined as a monitoring function. <p>Functional principle:</p> <ul style="list-style-type: none"> ■ The monitoring function is effective without further conditions. As soon as it is violated, the reaction is triggered. <p>Example: Workspace Monitoring</p> <ul style="list-style-type: none"> ■ If a safely monitored space is violated, a safety stop is triggered.
Only activation function	<p>Structure:</p> <ul style="list-style-type: none"> ■ The safety function contains 1 or 2 AMFs, which are defined as activation functions. <p>Functional principle:</p> <ul style="list-style-type: none"> ■ Once all the configured activation functions are violated, the reaction is immediately triggered. <p>Example: External EMERGENCY STOP</p> <ul style="list-style-type: none"> ■ Once an external EMERGENCY STOP device is activated, a safety stop 0 is triggered.
Combined monitoring and activation function	<p>Structure:</p> <ul style="list-style-type: none"> ■ The safety function contains a monitoring function as well as 1 or 2 activation functions. <p>Functional principle:</p> <ul style="list-style-type: none"> ■ When the monitoring function is violated, the configured reaction is only triggered if all the configured activation functions are also violated. <p>Example: Velocity monitoring in test mode</p> <ul style="list-style-type: none"> ■ The velocity is safely monitored. However, a violation only triggers a safety stop if test mode is also active.

Reactions

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.
- Safe output is set to "0".

The safe outputs of the discrete safety interface CIB_SR/X11 or the safe outputs of the Ethernet safety interface PROFINET/PROFIsafe (if configured in WorkVisual) can be used as safe outputs. All the safety-relevant outputs use LOW as a safe state.



Information about interface X11 of the KUKA Sunrise Cabinet robot controller can be found in the operating instructions for **KUKA Sunrise Cabinet**.



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.



When using safety functions with a safe PROFIsafe output as a reaction, it must be noted that the signal edge from the safety controller is only maintained as long as the safety function is in the violation state. A violation state is maintained for at least 12 ms.

The reactions can be used for any number of safety functions. A reaction is triggered once all the configured AMFs are violated for one of the safety functions using this reaction. This makes it possible, for example, to inform a higher-level controller via a safe output when specific errors occur.

If different reactions are to be triggered when a combination of AMFs is violated, e.g. a safety stop and setting an output, 2 identical safety functions must be configured for this. These merely differ in terms of the reaction.

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 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.

Categories

For diagnosis in case of error, a category is assigned to each safety function. 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
None	For safety functions which cannot be assigned a category
Output	For safety functions which use setting an output as a reaction In this category, no diagnostic information is provided in case of violation.
Enabling Device	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.
Local E-STOP	For safety functions which evaluate an EMERGENCY STOP triggered by the EMERGENCY STOP device on the smartPAD
External E-STOP	For safety functions which evaluate an EMERGENCY STOP triggered by an external EMERGENCY STOP device
Operator protection	For safety functions which evaluate the signal for operator safety
Safe operational stop	For safety functions which monitor robot standstill
Collision detection	For safety functions which are used for collision detection or force monitoring
Protective stop	For safety functions which use a safety stop as a reaction and cannot be assigned to another category. Example: external safety stop

Category	Recommended use
Velocity Monitoring	For safety functions which are used for monitoring an axis-specific or Cartesian velocity
Workspace Monitoring	For safety functions which are used for monitoring an axis-specific or Cartesian space

13.3 Overview of 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.

Atomic Monitoring Functions are divided into 3 categories:

- Standard AMFs
- Parameterizable AMFs
- Extended AMFs

13.3.1 Standard Atomic Monitoring Functions

Description Standard Atomic Monitoring Functions provide information about system components or system states, for example 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
Emergency stop smartPAD	Monitors the EMERGENCY STOP device on the smartPAD
Control panel enable smart-PAD released	Monitors the enabling switch on the smartPAD
Control panel panic smart-PAD	Checks whether an enabling switch on the smartPAD is in panic position

(>>> 13.6.1 "Evaluating the safety equipment on the KUKA smartPAD" Page 157)

AMFs for evaluating the operating mode:

AMF	Task
Operating mode Test	Checks whether a test operating mode is active (T1, T2 , CRR)
Operating mode Automatic	Checks whether the Automatic operating mode is active (AUT)
Operating mode with reduced speed	Checks whether an operating mode with reduced velocity is active (T1, CRR)
Operating mode with high velocity	Checks whether an operating mode with programmed velocity is active (T2 , AUT)

(>>> 13.6.2 "Evaluating the operating mode" Page 157)

AMFs for evaluating the motion enable:

AMF	Task
Motion enable	Monitors the motion enable, i.e. checks whether a safety stop is active.

(>>> 13.6.3 "Evaluating the motion enable" Page 157)

AMFs for evaluating the referencing status:

AMF	Task
Position referencing	Monitors the referencing status of the position values for all axes (>>> 13.6.4 "Evaluating the position referencing" Page 158)
Torque referencing	Monitors the referencing status of the joint torque sensors for all axes (>>> 13.6.5 "Evaluating the torque referencing" Page 158)

13.3.2 Parameterizable Atomic Monitoring Functions

Description In contrast to standard AMFs, parameterizable Atomic Monitoring Functions additionally have one or more parameters. These can be configured depending on the values at which the AMF is to be considered violated. Example: monitoring limits.

For parameterizable AMFs, a fixed number of instances is available. The number indicates how many differently parameterizable versions of the AMF can be configured and used. Each instance can have different parameter values. An instance of an AMF may be used multiple times in the table in which the safety functions are configured. If, however, different parameter settings are required, additional instances must be used.

Overview AMFs for evaluating safe inputs:

AMF	Task
Input signal	Monitors a safe input (>>> 13.6.6 "Monitoring safe inputs" Page 159)

AMFs for velocity monitoring:

AMF	Task
Axis velocity monitoring	Monitors the velocity of an axis (>>> 13.6.7.1 "Defining axis-specific velocity monitoring" Page 160)
Cartesian velocity	Monitors the translational Cartesian velocity at all tool spheres and joint center points as well as at the robot flange (>>> 13.6.7.2 "Defining Cartesian velocity monitoring" Page 161)

AMFs for space monitoring:

AMF	Task
Cartesian workspace monitoring	Checks whether a part of the robot structure is located outside of its permissible workspace (>>> 13.6.8.1 "Defining Cartesian workspaces" Page 163)
Cartesian protected space monitoring	Checks whether a part of the robot structure is located within a non-permissible protected space (>>> 13.6.8.2 "Defining Cartesian protected spaces" Page 164)
Axis range monitoring	Monitors the position of an axis (>>> 13.6.8.3 "Defining axis-specific monitoring spaces" Page 166)

AMFs for the safe monitoring of forces and torques:

AMF	Task
Axis torque monitoring	Monitors the torque of an axis (>>> 13.6.10.1 "Axis torque monitoring" Page 168)
Collision detection	Monitors the external torque of all axes (>>> 13.6.10.2 "Collision detection" Page 169)
TCP force monitoring	Monitors the external force acting on the TCP of a tool or on the robot flange (>>> 13.6.10.3 "TCP force monitoring" Page 170)

13.3.3 Extended Atomic Monitoring Functions

Description Extended Atomic Monitoring Functions differ from the standard AMFs and parameterizable AMFs in that monitoring parameters are only defined during operation. The parameters are set at the time of activation. For the AMF **Standstill monitoring 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.

Multiple instances are available for Extended AMFs. It is advisable to use each instance only once in the safety configuration, even with Extended AMFs which cannot be parameterized.

Overview AMF for standstill monitoring:

AMF	Task
Standstill monitoring all axes	Monitors the standstill on all axes (>>> 13.6.9 "Standstill monitoring (safe operational stop)" Page 167)

13.4 Safety configuration with KUKA Sunrise.Workbench

The safety configuration is an integral feature of a Sunrise project. It is managed in tabular form.

When creating a new Sunrise project, the system automatically creates a standard safety configuration with the safety function pre-configured by KUKA.
(>>> 5.3 "Creating a Sunrise project with a template" Page 46)

Further information on the standard safety configuration can be found here:
(>>> 3 "Safety" Page 21)

In KUKA Sunrise.Workbench, the safety configuration is displayed, edited and transferred to the controller when the system software is installed or the project is synchronized. The safety configuration can be activated and deactivated via the smartHMI.

13.4.1 Overview: changing the safety configuration and activating it on the controller

Step	Description
1	Open the safety configuration. (>>> 13.4.2 "Opening the safety configuration" Page 152)
2	Edit the safety functions in the PSM table User PSM or create new safety functions. (>>> 13.4.3 "Creating and editing safety functions in Sunrise.Workbench" Page 154)
3	Save the safety configuration.
4	When using position-based AMFs (>>> "Position-based AMFs" Page 177) If necessary, create the application prepared by KUKA for the position and torque referencing of the LBR iiwa or create a separate reference run application. (>>> 13.8.1 "Position referencing" Page 172)
5	When using axis torque-based AMFs (>>> "Axis torque-based AMFs" Page 177) Create the application prepared by KUKA for the position and torque referencing of the LBR iiwa and integrate the safety-oriented tool into the application. Further adaptations in the application may be necessary. (>>> 13.8.2 "Torque referencing" Page 173)
6	Transfer the project with the safety configuration to the robot controller. <ul style="list-style-type: none"> ■ When the system software is installed or the project is synchronized
7	Reboot the robot controller to apply the changed safety configuration.
8	Activate the safety configuration on the robot controller (>>> 13.5 "Activating the safety configuration on the robot controller" Page 155)
9	When using position-based AMFs (>>> "Position-based AMFs" Page 177) Carry out position referencing.
10	When using axis torque-based AMFs (>>> "Axis torque-based AMFs" Page 177) Carry out torque referencing.
11	Test the safety functions of the activated safety configuration. (>>> 13.9 "Safety acceptance overview" Page 175)

13.4.2 Opening the safety configuration

Procedure ■ In the Sunrise project, double-click on the file **SafetyConfiguration.sconf**.

Description The safety configuration contains 2 tables. The permanently active safety functions are configured in these tables.

■ KUKA PSM

The table contains the safety functions prescribed by KUKA. These cannot be modified or deactivated.

The table documents the system behavior and, in conjunction with the table **User PSM**, provides a full description of the safety functions.

■ User PSM

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.

13.4.2.1 Evaluating the safety configuration

When the safety configuration is evaluated, the **User 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.

13.4.2.2 Overview of the PSM table “User PSM”

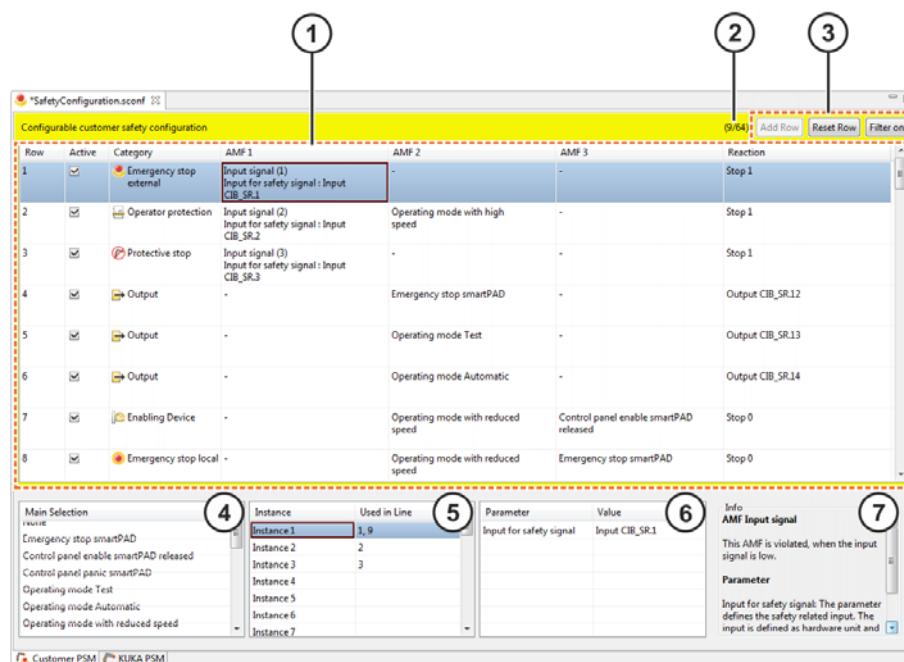


Fig. 13-2: Overview of PSM table “User PSM”

Item	Description
1	PSM table The table contains the configured safety functions.
2	Display of the currently defined PSM rows A total of 64 PSM rows are available for configuring the user-specific safety monitoring functions.
3	Buttons for editing the PSM table <ul style="list-style-type: none"> ■ Filter on/filter off: shows/hides undefined PSM rows ■ Reset Row: deletes the configuration of the selected PSM row. ■ Add Row: adds a new row in the PSM table (only active when the filter is on).

Item	Description
4	Selection table With respect to the cell selected in the PSM row, the category, AMF or reaction of a safety function can be selected here. (>>> 13.4.3.3 "Editing existing safety functions" Page 154)
5	Instance table The instances of the AMF marked in the Selection table, as well as the PSM rows in which they are used, are displayed here.
6	Parameter table The parameter values of the AMF instance selected in the Instance table are displayed here. The values can be changed.
7	Information display Displays the description of the selected category, AMF or reaction

13.4.3 Creating and editing safety functions in Sunrise.Workbench

In the PSM table **User 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.

13.4.3.1 Creating new safety functions

- Procedure**
- If the filter for displaying the safety functions is not active:
1. Select an empty row in the PSM table.
 2. Set the category, the AMFs used and the reaction of the safety function in the corresponding columns of the PSM row.
 3. Set the check mark in the **Active** check box if the row is to be activated.
- If the filter for displaying the safety functions is active:
1. Click on the **New Row** button. An empty PSM row is added. The row is automatically activated (check mark in the **Active** check box).
 2. Set the category, the AMFs used and the reaction of the safety function in the corresponding columns of the PSM row.

13.4.3.2 Deleting a safety function

- Procedure**
1. In the PSM table, select the PSM row containing the safety function to be deleted.
 2. Click on the **Reset Row** button. The PSM row is removed from the table.

13.4.3.3 Editing existing safety functions

- Procedure**
- Changing the category:**
1. Select the **Category** column in the desired PSM row. The available categories are displayed in the Selection table.
 2. Select the desired category from the Selection table. The category is inserted in the PSM row.
- Changing the AMF used:**

1. Select the column **AMF1**, **AMF2** or **AMF3** in the desired PSM row. The available AMFs are displayed in the Selection table.
2. Select the desired AMF from the Selection table. The AMF is inserted in the PSM row.
3. For multiply instanced AMFs: select the desired instance from the Selection table. The instance is inserted in the PSM row.
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 PSM row. The available reactions are displayed in the Selection table.
2. Select the desired reaction from the Selection table. The reaction is inserted in the PSM row.
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 a safety function:

- Activate the check box in the **Active** column in the desired PSM row (set the check mark).



Only the safety functions activated here are available on the robot controller after transferring and activating the safety configurations.

Deactivating a safety function:

- Deactivate the check box in the **Active** column in the desired PSM row (remove the check mark).

13.5 Activating the safety configuration on the robot controller

Description

A safety configuration is effective only after it has been transferred to the robot controller and activated on the smartHMI. 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). This is displayed in the **Safety** level in the **Activation** area. With this ID, the safety maintenance technician can clearly identify the safety configuration activated on the robot controller.

When installing the system software, the robot controller must be rebooted. If a changed safety configuration is transferred along with the installation, it must be activated after the reboot.

If changes to an active safety configuration are transferred to the robot controller by means of project synchronization, the system response depends on whether the synchronization was completed by rebooting the robot controller:

- For synchronization with reboot: the old safety configuration is no longer active and the new configuration is not yet active. The robot can no longer be moved.
- For synchronization without reboot: the old safety configuration remains active until the robot controller has been rebooted. The robot can be moved until then.



In case the newly transferred safety configuration is not activated, the old safety configuration can be restored.

(>>> 13.5.2 "Restoring the safety configuration" Page 156)

A password is required for activation. The default password is "argus".



The password to activate the safety configuration must be changed before start-up. This password may only be communicated to trained safety maintenance technicians who are authorized to activate the safety configuration.
(>>> 13.5.3 "Changing the password for activating the safety configuration" Page 156)

- Procedure**
1. In the Station view, select **Safety > Activation**.
 2. Enter the password and press **Activation**.
 3. Only required if the robot controller was not rebooted after installation of the system software or after project synchronization: reboot the robot controller.

13.5.1 Deactivating the safety configuration

- Description**
- An activated safety configuration can be deactivated again. It is the responsibility of the safety maintenance technician to check that the deactivation has been successful. If the safety configuration is deactivated, the robot can no longer be moved. This is displayed in a message in the Station view (**Safety** tile).
- Procedure**
1. In the Station view, select **Safety > Activation**.
 2. Enter the password and press **Deactivation**.

13.5.2 Restoring the safety configuration

- Description**
- This function is only available if a new safety configuration has been transferred to the robot controller but not activated. This procedure can be used to restore the most recently activated safety configuration.
- Procedure**
1. In the Station view, select **Safety > Activation**.
 2. Enter the password and press **Undo**.
 3. Only required if the robot controller was not rebooted after project synchronization: reboot the robot controller.

13.5.3 Changing the password for activating the safety configuration

- Procedure**
1. In the Station view, select **Safety > Change Password**.
 2. Enter the old password in the **Password** box.
 3. In the boxes below enter the new password twice.
For security reasons, the entries are displayed encrypted. Upper and lower case are distinguished.
 4. Press **Change**. The new password is valid immediately.



If the password has been forgotten, the memory card in the robot controller must be exchanged by KUKA Service. Using the default password, the password for activating the safety configuration must then be changed again.

13.6 Use and parameterization of the Atomic Monitoring Functions

The following describes the use, functional principle and parameterization of the individual AMFs. Configuration examples are added to the description.

13.6.1 Evaluating the safety equipment on the KUKA smartPAD

The KUKA smartPAD has an EMERGENCY STOP device and an enabling device. The corresponding safety-oriented functions are preconfigured in the KUKA table and cannot be changed.

Further safety functions evaluated by the safety equipment on the smartPAD can be configured. The following AMFs are available for this purpose:

AMF	Description
Emergency stop smartPAD	This AMF is violated if the EMERGENCY STOP device on the smartPAD is pressed. Standard AMF
Control panel enable smart-PAD released	This AMF is violated if no enabling switch is pressed on the smartPAD or if an enabling switch is fully pressed (panic position). Standard AMF
Control panel panic smart-PAD	This AMF is violated if an enabling switch is fully pressed (panic position). Standard AMF

13.6.2 Evaluating the operating mode

The set operating mode has a powerful effect on the behavior of the industrial robot and determines which safety precautions are required.

The following AMFs are available for configuring a safety function to evaluate the set operating mode:

AMF	Description
Operating mode Test	This AMF is violated if a test operating mode is active (T1, T2 , CRR). Standard AMF
Operating mode Automatic	This AMF is violated if an Automatic operating mode is active (AUT). Standard AMF
Operating mode with reduced speed	This AMF is violated if an operating mode is active whose velocity is reduced to a maximum of 250mm/s (T1, CRR). Standard AMF
Operating mode with high speed	This AMF is violated if an operating mode is active in which the robot is moved with a programmed velocity (T2, AUT). Standard AMF

13.6.3 Evaluating the motion enable

The robot cannot be moved without the motion enable. The motion enable can be cancelled for various reasons, for example 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
Motion enable	This 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. Standard AMF

ExampleSwitching 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 canceled. It is only to be switched off if the operator safety is violated.

AMF1	AMF2	AMF3	Reaction
Input signal (operator safety)	-	Motion enable	Output (tool)

13.6.4 Evaluating the position referencing

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 safely monitored Cartesian and axis-specific robot positions, Cartesian velocities, forces and collisions.

The AMF **Position referencing** can be used to check whether the position values of all axes are referenced.

AMF	Description
Position referencing	This AMF is violated if the position of at least one of the axes of the kinematic system is not referenced, or if the position referencing test on at least one axis has failed. Standard AMF

ExampleMonitoring the position referencing status (category: **Protective stop**)

To prevent danger in the case of failure or non-execution of a position referencing test, a robot whose axes are not referenced may only be moved at a reduced maximum velocity of 250 mm/s.

In order to guarantee this, the referencing status of all axes in the operating modes with high velocity (T2 and AUT) is monitored and a safety stop 1 is triggered as soon as the position of one axis has not been successfully referenced.

AMF1	AMF2	AMF3	Reaction
Operating mode with high velocity	-	Position referenced	Stop 1

13.6.5 Evaluating the torque referencing

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 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
Torque referencing	This AMF is violated if the joint torque sensor of at least one axis is not referenced, or if the referencing test of at least one joint torque sensor has failed. Standard AMF

Example

Monitoring the referencing status (category: **Protective 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 is to be triggered in high-velocity operating modes (T2 and AUT).

AMF1	AMF2	AMF3	Reaction
Collision detection	-	-	Stop 0
Operating mode with high velocity	-	Torque referencing	Stop 1

13.6.6 Monitoring safe inputs

The safe inputs of the discrete safety interface CIB_SR/X11 or the safe inputs of the Ethernet safety interface PROFINET/PROFIsafe (if configured in WorkVisual) can be used as safe inputs.



Information about interface X11 of the KUKA Sunrise Cabinet robot controller can be found in the operating instructions for **KUKA Sunrise Cabinet**.

Safety equipment can be connected to these inputs, for example external EMERGENCY STOP devices or safety gates. The AMF **Input signal** is used to evaluate the associated input signal.

AMF	Description
Input signal	This AMF is violated if the safe input used is low (state "0"). <ul style="list-style-type: none"> ■ Safe inputs CIB_SR/X11: CIB_SR.1 ... CIB_SR.7 ■ Safe inputs PROFINET/PROFIsafe (if configured in WorkVisual) Parameterizable AMF with 32 available instances



If a connection error, i.e. a communication error, occurs at a safe input, the robot is not automatically stopped, but rather only the corresponding AMF is evaluated as violated. The robot is only stopped if the PSM row with this input AMF is violated and a stop reaction is configured. Only after the connection error has occurred a number of times does the system enter the safe state without the safety function being violated.

Example 1**Operator protection (category: Operator protection)**

A safety gate is connected to a safe input. If the safety gate is opened in Automatic or T2 mode, a safety stop 1 is to be triggered.

AMF1	AMF2	AMF3	Reaction
Input signal	Operating mode with high velocity	-	Stop 1

Example 2**External EMERGENCY STOP (category: Emergency stop external)**

An external EMERGENCY STOP device is connected to a safe input. If an EMERGENCY STOP device is pressed, a safety stop 1 is triggered.

AMF1	AMF2	AMF3	Reaction
Input signal	-	-	Stop 1

Example 3**External enabling switch (category: Enabling device)**

An external enabling device is connected to a safe input. If no external enabling signal has been issued, a safety stop 1 is to be triggered if a Cartesian protected space is violated and Automatic mode is active at the same time.

AMF1	AMF2	AMF3	Reaction
Input signal	Operating mode Automatic	Cartesian protected space monitoring	Stop 1

13.6.7 Velocity monitoring functions

A moving robot 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.

The axis-specific velocities and the Cartesian velocity of the robot can be monitored.

13.6.7.1 Defining axis-specific velocity monitoring

The AMF **Axis velocity monitoring** is used to define axis-specific velocity monitoring.

AMF	Description	
Axis velocity monitoring	This AMF is violated if the absolute velocity of the monitored axis exceeds the configured limit. Parameterizable AMF with 16 available instances	

Parameters of the AMF

Parameter	Description
Monitored axis	Axis to be monitored ■ Axis1 ... Axis16 Note: Axis1 ... Axis7 are used for the KUKA LBR iiwa.
Maximum velocity [°/s]	Maximum permissible velocity at which the monitored axis may move in the positive and negative direction of rotation ■ 1 ... 500 °/s

13.6.7.2 Defining Cartesian velocity monitoring

The AMF **Cartesian velocity monitoring** is used to define Cartesian velocity monitoring. It monitors the translational Cartesian velocity 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 also monitors the velocity at the center points of the spheres which are configured for this tool.

(>>> 9.3.7 "Safety-oriented tool" Page 120)



The system does not monitor the entire structure of the robot and tool against the violation of a velocity limit, but rather only the monitoring spheres. In particular with protruding tools, 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.

AMF	Description
Cartesian velocity	<p>This AMF is violated if the Cartesian velocity at one or more of the monitored points exceeds the configured limit, if one of the robot axes is unmastered or if position referencing of a mastered axis has failed.</p> <p>Parameterizable AMF with 8 available instances</p> <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>

Parameters of the AMF	Parameter	Description
	Maximum velocity [mm/s]	<p>Maximum permissible Cartesian velocity which must not be exceeded at any monitored point</p> <ul style="list-style-type: none"> ■ 1 ... 10,000 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
Cartesian workspace monitoring	-	Cartesian velocity	Stop 1

13.6.8 Monitoring spaces

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 parts of the robot structure are 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 Atomic 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 safely monitored space and been stopped by the safety controller, it can only be moved out of the violated area in CRR mode.

(>>> 6.8 "CRR mode – controlled robot retraction" Page 74)

Spheres on the robot

Spheres are modeled around selected points on the robot, covering and moving with the robot. These spheres are monitored against the 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 sphere center lies on the center point of each axis, of the robot base and of the robot flange.

Radii of the monitored spheres on the LBR iiwa 7 R800:

Base	A1	A2	A3	A4	A5	A6	A7	Flange
135 mm	70 mm	131 mm	70 mm	131 mm	70 mm	55 mm	170 mm	32 mm

Radii of the monitored spheres on the LBR iiwa 14 R820:

Base	A1	A2	A3	A4	A5	A6	A7	Flange
155 mm	90 mm	146 mm	70 mm	131 mm	70 mm	55 mm	170 mm	32 mm

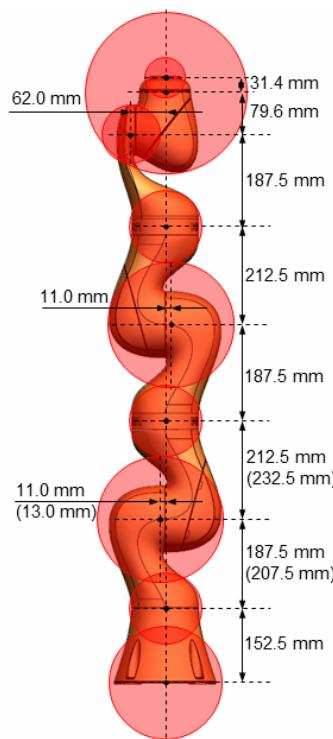


Fig. 13-3: Monitoring spheres on the LBR iiwa 7 R800

Spheres on tool

If a safety-oriented tool is active on the robot controller, the system also monitors the monitoring spheres which are configured for this tool in addition to the spheres on the robot.

(>>> 9.3.7 "Safety-oriented tool" Page 120)



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, 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.

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



The stopping distance of the robot depends primarily on the dynamic characteristics of the robot type. The robot's acceleration within the reaction time of the monitoring functions varies depending on the robot type in the event of an error. This affects the actual stopping distance.

This aspect must be taken into account by the system integrator during parameterization of the monitoring functions as part of the safety assessment.

13.6.8.1 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.

The monitoring spheres which move with the robot are simultaneously monitored against the activated Cartesian workspaces and must move within these workspaces.

The AMF **Cartesian workspace monitoring** is used to define a Cartesian workspace. This 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.

8 instances are available for this parameterizable AMF, e.g. a maximum of 8 Cartesian workspaces can be configured.

Parameters of the AMF

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
X, Y, Z [mm]	Offset of the origin of the workspace 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 workspace 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 workspace is determined along the coordinate axes:

Parameter	Description
Length [mm]	Length of the workspace (= distance along the positive X axis of the origin) ■ 0 mm ... 100,000 mm
Width [mm]	Width of the workspace (= distance along the positive Y axis of the origin) ■ 0 mm ... 100,000 mm
Height [mm]	Height of the workspace (= distance along the positive Z axis of the origin) ■ 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 10 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.

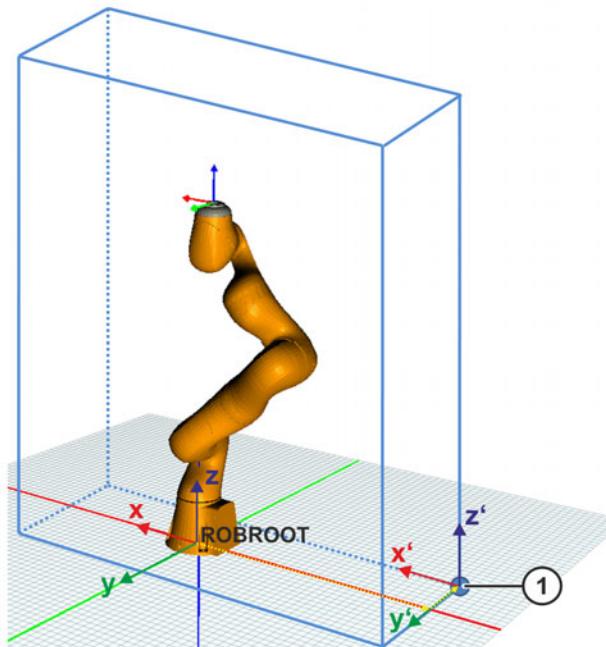


Fig. 13-4: Example of a Cartesian workspace

1 Origin of the workspace

13.6.8.2 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.

The monitoring spheres which move with the robot are simultaneously monitored against the activated Cartesian protected spaces and must move outside of these protected spaces.

The AMF **Cartesian protected space monitoring** is used to define a Cartesian protected space. This AMF is violated as soon as one of the monitored 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.

8 instances are available for this parameterizable AMF, e.g. a maximum of 8 Cartesian protected spaces can be configured.

Parameters of the AMF

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 10 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.

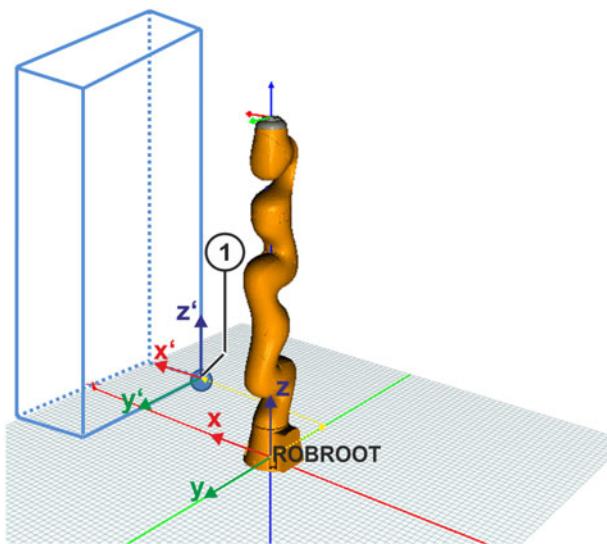


Fig. 13-5: Example of a Cartesian protected space

1 Origin of the protected space

13.6.8.3 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. This AMF is violated if an axis is not inside the defined axis range. The AMF is additionally violated in the following cases: <ul style="list-style-type: none">■ An axis is not mastered.■ The referencing of a mastered axis has failed. 16 instances are available for this parameterizable AMF, e.g. a maximum of 16 Cartesian workspaces can be configured.
--------------------	--

Parameters of the AMF

Parameter	Description
Monitored axis	Axis to be monitored <ul style="list-style-type: none">■ Axis1 ... Axis16 Note: Axis1 ... Axis7 are used for the KUKA LBR iiwa.
Lower limit [°]	Lower limit of the allowed axis range in which the monitored axis may move <ul style="list-style-type: none">■ -180° ... +180°
Upper limit [°]	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 upper to the lower 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.

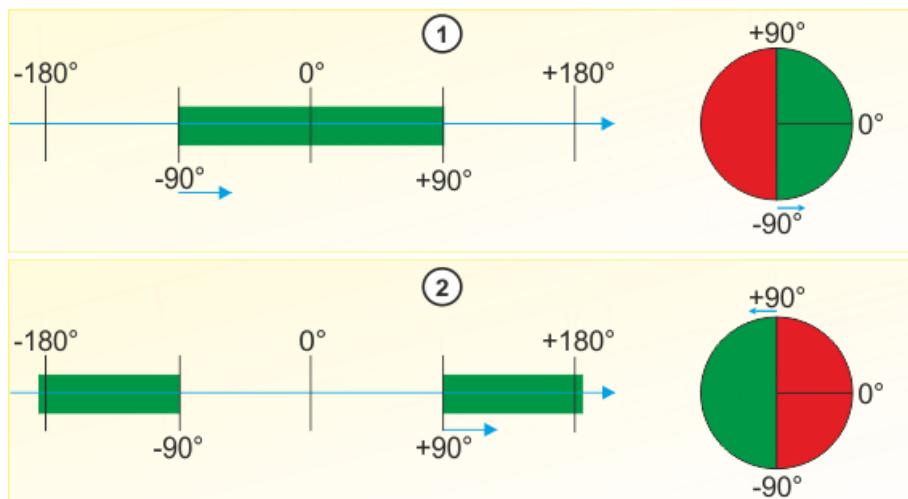


Fig. 13-6: Examples of axis-specific workspaces

- 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 0 is triggered. For this purpose, an individual PSM row must be used for each axis.

Configurable customer safety configuration						(12/64)	Add Row	Reset Row	Filter on
Row	Active	Category	AMF 1	AMF 2	AMF 3	Reaction			
10	<input checked="" type="checkbox"/>	Workspace Monitoring	Input signal (4) Input for safety signal : InputCIB_SR.4	-	Axis range monitoring (1) Monitored axis : Axis 1		Stop 0		
11	<input checked="" type="checkbox"/>	Workspace Monitoring	Input signal (4) Input for safety signal : InputCIB_SR.4	-	Axis range monitoring (2) Monitored axis : Axis 2		Stop 0		
12	<input checked="" type="checkbox"/>	Workspace Monitoring	Input signal (4) Input for safety signal : InputCIB_SR.4	-	Axis range monitoring (4) Monitored axis : Axis 4		Stop 0		

Fig. 13-7: PSM table – simultaneous monitoring of 3 axes

13.6.9 Standstill monitoring (safe operational stop)

If, under certain conditions, the robot must not move but must remain under servo-control, the standstill of all axes must be safely monitored. For this purpose, there is an AMF for the standstill monitoring of all axes. As it concerns an Extended AMF, the monitoring only begins when all other AMFs of the safety function are violated.

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, for example if the robot is jolted.

AMF	Description
Standstill monitoring all axes	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. Non-parameterizable Extended AMF with 8 available instances

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
Input signal (sensor)	Cartesian workspace monitoring	Standstill monitoring all axes	Stop 0

13.6.10 Monitoring of forces and torques

The KUKA LBR iiwa is fitted with position and joint torque sensors in every axis. These make it possible to measure and react to external forces and torques.



When using the AMFs **Collision detection**, **TCP force monitoring** and **Axis torque monitoring** it must be taken into account that, due to the stopping distances of the robot, the interaction forces may continue to increase if the AMF is violated and a safety stop is triggered. For this reason it is advisable to carry out collaborative operation (HRC) at reduced velocities only, i.e. to combine these AMFs with the AMF **Cartesian velocity** or **Axis velocity monitoring**.



When monitoring with the AMFs **Collision detection** and **TCP force monitoring**, it is not permissible to pick up additional loads (e.g. workpieces). This additional load would be misinterpreted as an external force, which would cause the safety integrity of this AMF to be lost.

13.6.10.1 Axis torque monitoring

Axis torque monitoring can limit and monitor the torques of individual axes.



If the permissible axis torque is exceeded continuously due to jamming, it is possible to move the robot free by changing to CRR mode.

AMF	Description
Axis torque monitoring	This AMF is violated if the torque of the monitored axis exceeds or falls below the configured torque limit. Parameterizable AMF with 16 available instances

Parameters of the AMF

Parameter	Description
Monitored axis	Axis to be monitored <ul style="list-style-type: none"> ■ Axis1 ... Axis16 <p>Note: Axis1 ... Axis7 are used for the KUKA LBR iiwa.</p>
Minimum torque [Nm]	Minimum permissible torque for the given axis <ul style="list-style-type: none"> ■ -500 ... 500 Nm
Maximum torque [Nm]	Maximum permissible torque for the given axis <ul style="list-style-type: none"> ■ -500 ... 500 Nm

13.6.10.2 Collision detection

Collision detection monitors the external axis torques against a definable limit value.

The external axis torque is defined as that part of the torque of an axis which is generated from the forces and torques occurring as the robot interacts with its environment. It 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 the safety-oriented tool are taken into consideration (if configured).
- If the safety-oriented tool is configured, it must also actually be mounted on the robot flange.
- Since the robot's angle of inclination influences the calculation of the external torque, only floor-mounting is permissible.

AMF	Description
Collision detection	This AMF is violated if the external torque of at least one axis exceeds the configured limit value. Parameterizable AMF with 8 available instances

Parameters of the AMF

Parameter	Description
Maximum external torque [Nm]	Maximum permissible external torque <ul style="list-style-type: none"> ■ 0 ... 500 Nm



When using the AMF **Collision detection**, it is important to note that external forces on the robot structure with short distances to the robot axes only cause slight external torques in the robot axes under certain circumstances. This can pose a safety risk particularly in potential crushing situations during collaborative operation (HRC). Possibly critical incidents of crushing can be avoided by using suitable equipment for the robot cell or by additionally using one of the following AMFs: **Cartesian workspace monitoring**, **Cartesian protected space monitoring** or **Axis range monitoring**.

13.6.10.3 TCP force monitoring

In TCP force monitoring, the external force acting on the TCP of the safety-oriented tool is monitored against a definable limit value. If no tool is used, the external force acting on the center point of the robot flange is monitored.

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 the safety-oriented tool are taken into consideration (if configured).
- If the safety-oriented tool is configured, it must also actually be mounted on the robot flange.
- Since the robot's angle of inclination influences the calculation of the external force, only floor-mounting is permissible.

AMF	Description
TCP force monitoring	<p>This AMF is violated if the external force acting on the TCP of the safety-oriented tool (or center point of the robot flange) exceeds the configured limit value.</p> <p>Parameterizable AMF with 8 available instances</p>

Parameters of the AMF	Parameter	Description
	Maximum TCP force [N]	<p>Maximum permissible external force on the TCP</p> <ul style="list-style-type: none"> ■ 50 ... 1,000 N

- | | |
|------------------------------------|---|
| Accuracy of force detection | <ul style="list-style-type: none"> ■ The accuracy of TCP force detection is also dependent on the robot pose. The safety controller recognizes non-permissible poses and sets the AMF TCP force monitoring to "violated" with a corresponding diagnostic 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. Axis angles close to 0° in A2, A4 and A6 must be avoided in particular: <ul style="list-style-type: none"> ■ Axis A2: -35° to 35° ■ Axis A4: -55° to 55° ■ Axis A6: -20° to 20° ■ External forces on the robot structure reduce the accuracy of TCP force detection. In many cases, the safety controller can automatically detect the external forces acting on the robot structure. 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 structure. 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**.

13.7 Example of a safety configuration



The sole purpose of this example is to illustrate the safety configuration with KUKA Sunrise.Workbench.

Task

A robot is used for assembling a workpiece. There is a rack next to the robot for fetching the parts needed and setting down the finished workpiece. Human workers supply the material and collect the finished workpieces from the other side of the rack. The cell is designed so that persons can only access the work-space of the robot in the material provision zone.

Requirements

The following safety requirements were determined within the scope of a risk analysis:

1. The Cartesian velocity of the robot must not exceed 800 mm/s.
2. There is a risk of collision if the robot is situated in the material provision zone. In this case, the higher-level controller must be notified via a safe output.
3. Collisions with more than 35 Nm are not permissible in the material provision zone.
4. The velocity of the robot must not exceed 500 mm/s in the material provision zone.
5. In the material provision zone, velocities of 250 mm/s may only be exceeded if it is possible to avoid collisions causing an external torque of 20 Nm in the joints of the robot.

Solution

Configurable customer safety configuration						(14/64)	Add Row	Reset Row	Filter on
Row	Active	Category	AMF 1	AMF 2	AMF 3	Reaction			
10	<input checked="" type="checkbox"/>	Velocity Monitoring	Cartesian velocity (1) Maximum velocity : 800 mm/s	-	-	Stop 0			
11	<input checked="" type="checkbox"/>	Workspace Monitoring	Cartesian protected space monitoring(1)	-	-	Output PROFlsafe Byte 0/3			
12	<input checked="" type="checkbox"/>	Collision detection	Cartesian protected space monitoring(1)	Collision detection (1) Maximum external torque : 35 Nm	-	Stop 0			
13	<input checked="" type="checkbox"/>	Velocity Monitoring	Cartesian protected space monitoring(1)	Cartesian velocity (2) Maximum velocity : 500 mm/s	-	Stop 0			
14	<input checked="" type="checkbox"/>	Collision detection	Cartesian protected space monitoring(1)	Cartesian velocity (3) Maximum velocity : 250 mm/s	Collision detection (2) Maximum external torque : 20 Nm	Stop 1			

Fig. 13-8: Example of a safety configuration

Row	Description
10	<p>Velocity Monitoring</p> <p>Implements requirement 1</p> <p>If the robot moves at a velocity of more than 800 mm/s, a safety stop 0 is triggered.</p>
11	<p>Workspace Monitoring</p> <p>The material provision zone is represented by a Cartesian protected space for which AMF instance no. 1 is used.</p> <p>Implements requirement 2</p> <p>If the robot is situated in the material provision zone (AMF Cartesian protected space monitoring violated), a safe output is set in order to signal that the robot has entered the potential collision area.</p>

Row	Description
12	<p>Collision detection Implements requirement 3</p> <p>If the robot is situated in the material provision zone (AMF Cartesian protected space monitoring violated) and a collision then occurs in which an external torque of 35 Nm is exceeded on at least one axis of the robot (AMF Collision detection violated), a safety stop 0 is triggered.</p>
13	<p>Velocity Monitoring Implements requirement 4</p> <p>If the robot is situated in the material provision zone (AMF Cartesian protected space monitoring violated) and moves at a velocity exceeding 500 mm/s (AMF Cartesian velocity violated), a safety stop 0 is triggered.</p>
14	<p>Collision detection Implements requirement 5</p> <p>If the robot is situated in the material provision zone (AMF Cartesian protected space monitoring violated) and moves there at a velocity exceeding 250 mm/s (AMF Cartesian velocity violated), a collision caused by an external torque of over 20 Nm on at least one axis of the robot (AMF Collision detection violated) triggers a safety stop 1.</p>

13.8 Position and torque referencing

13.8.1 Position referencing

Description	<p>Position referencing checks whether the saved zero position of the motor of an axis (= saved mastering position) corresponds to the actual mechanical zero position.</p> <p>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 if the mastering sensor does not detect the mechanical zero position of the axis within the range of the saved zero position of the motor, or if it is detected at an unexpected position.</p> <p>The safety integrity of the safety functions based upon this is limited until the position referencing test has been performed. This includes safely monitored Cartesian and axis-specific robot positions, Cartesian velocities, forces and collisions.</p> <p>If position referencing fails on at least one axis, all AMFs based on safe axis positions are violated. (>>> "Position-based AMFs" Page 177)</p>
Requirement	<p>The position of an axis is not referenced after the following events:</p> <ul style="list-style-type: none"> ■ The robot controller is rebooted. ■ The axis is remastered.



Following these events, the safety functions based on safe positions are not violated. The robot can be moved, but the safety integrity of the safety functions is not given.

The position-based safety functions are only violated after these events if position referencing of one 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. ([>>> 13.6.4 "Evaluating the position referencing" Page 158](#))

Precondition

The position of an axis is referenced when it is moved over the saved zero position of the motor and when the zero position of the axis is then detected in a range of $0^\circ \pm 0.5^\circ$ by the mastering sensor. Preconditions for this:

- The velocity at which the axis is moved over the zero position must be $< 30^\circ/\text{s}$.
- An axis-specific range before and after the zero position must be passed through as a minimum. 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 iiwa 7 R800	$\pm 10.5^\circ$	$\pm 14^\circ$	$\pm 14^\circ$				
LBR iiwa 14 R820	$\pm 9.5^\circ$	$\pm 9.5^\circ$	$\pm 10.5^\circ$	$\pm 10.5^\circ$	$\pm 10.5^\circ$	$\pm 14^\circ$	$\pm 14^\circ$

Performance

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 $30^\circ/\text{s}$.
- Jogging each axis individually over the zero position.
- Executing the application prepared by KUKA for referencing. The axes are moved over the zero position from the vertical stretch position.

A prepared application for position and torque referencing of the LBR iiwa is available from Sunrise.Workbench. Position and torque referencing can be carried out simultaneously with this application.

([>>> 13.8.3 "Creating an application for position and torque referencing" Page 174](#))



If it is not possible to reference from the vertical stretch position, a customized application for position referencing must be created and executed.

13.8.2 Torque referencing

Description

The LBR iiwa 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.

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 torque referencing has been performed successfully. This includes 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 177](#))

Requirement

The joint torque sensor of an axis is not referenced after the following events:

- The robot controller is rebooted.
- Position referencing of the axis fails.
- The maximum torque of the joint torque sensor has been exceeded.



Following these events, the safety functions based on safe torques are not violated. The robot can be moved, but the safety integrity of the safety functions is not given.

The torque-based safety functions 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. ([>>> 13.6.5 "Evaluating the torque referencing" Page 158](#))

Performance

A prepared application for position and torque referencing of the LBR iiwa is available from Sunrise.Workbench. Position and torque referencing can be carried out simultaneously with this application.

([>>> 13.8.3 "Creating an application for position and torque referencing" Page 174](#))

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.

Torque referencing must be executed with the safety-oriented tool mounted on the flange. This tool must still be integrated into the application.



Before torque referencing is executed, the user must ensure that the load data of the tool mounted on the robot correspond to the load data specified for the safety-oriented tool. The safety integrity of the referencing of the joint torque sensors is otherwise not given.

In particular, no supplementary loads may be mounted on the robot, e.g. loads attached to the robot structure or workpieces which are picked up.



No external forces may act on the robot and the tool mounted on the flange during torque referencing. The user must ensure this when referencing is executed. 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.

13.8.3 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 of the robot structure.
- The measurement is started with the method sendSafetyCommand().

- There may be a maximum of 15 s between 2 consecutive measurements.

Procedure

1. Select the Sunrise project in the **Package Explorer**.
2. Select the menu sequence **File > New > Other....**
3. In the **Sunrise** folder, select the **Application for Position and GMS Referencing of LBR iiwa** option and click on **Finish**.
The **PositionAndGMSReferencing.java** application is created in the source folder of the project and opened in the editor area of Sunrise.Workbench.
4. Make the following adaptations in the application:
 - Integrate the safety-oriented tool in the application.
 - If measurement poses cannot be addressed due to the system configuration, adapt them.
5. Synchronize the project in order to transfer the application to the robot controller.

13.9 Safety acceptance overview

The system 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 industrial robot
- After a change to the industrial robot
- 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 using the risk analysis as a basis. 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 checklist 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 tests of the safety configuration
(>>> 13.9.1 "Checklist for general safety functions" Page 176)
2. Checklists for checking the safety-oriented tool
(>>> 13.9.2 "Checklists for the safety-oriented tool" Page 177)
3. Checklists for checking the PSM rows used
(>>> 13.9.3 "Checklist for PSM rows" Page 178)
4. Checklists for checking the AMFs used
(>>> 13.9.4 "Checklists for AMFs used" Page 179)

13.9.1 Checklist for general 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 protection: is all operator protection equipment configured, properly connected and tested for correct function?		
2	Operator protection: a stop is triggered if AUT or T2 mode is active with the operator safety open.		
3	Operator protection: a manual reset function is present and activated.		
4	Enabling switch: is a safety stop 0 configured for PSM rows for the enabling switch?		
5	Local E-STOP: is a safety stop 0 configured for all E-STOP PSM rows?		
6	External E-STOP: is a safety stop 0 configured for all E-STOP PSM rows?		
7	Local E-STOP: are all local E-STOP devices configured, properly connected and tested for correct function?		
8	External E-STOP: are all external E-STOP devices configured, properly connected and tested for correct function?		
9	Protective stop: is all safety equipment configured, properly connected and tested for correct function?		
10	Safe operational stop: is all equipment for the safe operational stop configured, properly connected and tested for correct function?		
11	When using position-based AMFs: is the limited safety integrity of the position-based AMFs taken into consideration in the absence of position referencing? (>>> "Position-based AMFs" Page 177) Note: Initiation of the safe state in the absence of position referencing can be configured by using the AMF Position referencing .		
12	When using position-based AMFs: has position referencing been carried out successfully?		
13	Velocity Monitoring: have all necessary velocity monitoring tests been configured and tested?		
14	Workspace Monitoring: have all necessary workspace monitoring tests been configured and tested?		
15	Collision detection: have all necessary HRC functionalities been configured?		
16	Collision detection: has it been configured in such a way that velocity monitoring is also active when collision detection is active?		
17	Collision detection: has it been configured in such a way that velocity monitoring is also active when force detection is active?		
18	When using the AMF Collision detection or the AMF TCP force monitoring : is the robot floor-mounted horizontally with an accuracy of < 0.5°?		

No.	Activity	Yes	Not relevant
19	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? (>>> "Axis torque-based AMFs" Page 177) Note: Initiation of the safe state in the absence of position and/or torque referencing can be configured by using the AMF Position referencing and the AMF Torque referencing .		
20	Have torque and position referencing been carried out successfully?		

Place, date	
Signature	

By signing, the signatory confirms the correct and complete performance of the safety acceptance test.

Position-based AMFs The safety integrity of position-based AMFs is only given without limitations when position referencing has been carried out successfully.

AMF	Position referencing	Torque referencing
Standstill monitoring all axes	✓	✗
Axis range monitoring	✓	✗
Cartesian velocity	✓	✗
Cartesian workspace monitoring	✓	✗

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.

AMF	Position referencing	Torque referencing
Axis torque monitoring	✗	✓
Collision detection	✓	✓
TCP force monitoring	✓	✓

13.9.2 Checklists for the safety-oriented tool

13.9.2.1 Geometry data of the safety-oriented 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**
- **Cartesian workspace monitoring**
- **Cartesian protected space monitoring**

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.

Precondition

- Position referencing has been carried out successfully.

Checklist

No.	Activity	Yes	Not relevant
1	Have the radius and position of tool sphere 1 been correctly entered and checked?		
2	Have the radius and position of tool sphere 2 been correctly entered and checked?		
3	Have the radius and position of tool sphere 3 been correctly entered and checked?		
4	Have the radius and position of tool sphere 4 been correctly entered and checked?		
5	Have the radius and position of tool sphere 5 been correctly entered and checked?		
6	Have the radius and position of tool sphere 6 been correctly entered and checked?		

13.9.2.2 Load data of the safety-oriented tool

Description

If one 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

The load data can be checked by carrying out torque referencing at suitable poses, e.g. adjacent poses in the horizontal extended position. If the load data are correct, torque referencing must be successful.

Precondition

- Position and torque referencing have been carried out successfully.

Checklist

No.	Activity	Yes	Not relevant
1	Have the load data of the tool been correctly entered and checked?		

13.9.3 Checklist for PSM rows

Description

Each PSM row in the **User 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.

A PSM row can be tested by violating 2 of its AMFs. 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.

(>>> 13.9.4 "Checklists for AMFs used" Page 179)



For each PSM row, the points in the checklist must be executed and separately documented.

Checklist

- PSM 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: _____		
3	AMF 3 was tested successfully. Precondition: AMF 1 and AMF 2 are violated. AMF 3: _____		
4	The configuration of the row takes into account that the safe state of the AMFs is the “violated” state. In the event of an error, an AMF goes into the safe state.		
5	The configuration of the row takes into account that the safe state of the output is LOW.		

13.9.4 Checklists for AMFs used

An AMF which is used in more than one PSM row must be separately tested in each PSM row. ([>>> 13.9.3 "Checklist for PSM rows" Page 178](#))

13.9.4.1 AMF “Emergency stop smartPAD”

Checklist

No.	Activity	Yes	Not relevant
1	This AMF is the only AMF configured in the PSM row.		
2	The configured reaction is triggered by pressing the E-STOP on the smartPAD.		

13.9.4.2 AMF “Control panel enable smartPAD released”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by releasing the enabling switch on the smartPAD.		

13.9.4.3 AMF “Control panel panic smartPAD”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by pressing the enabling switch on the smartPAD fully down.		

13.9.4.4 AMF “Operating mode Test”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in T1.		
2	The configured reaction is triggered in T2.		
3	The configured reaction is triggered in CRR.		

13.9.4.5 AMF “Operating mode Automatic”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in AUT.		

13.9.4.6 AMF “Operating mode with reduced speed”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in T1.		
2	The configured reaction is triggered in CRR.		

13.9.4.7 AMF “Operating mode with high velocity”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in T2.		
2	The configured reaction is triggered in AUT.		

13.9.4.8 AMF “Input”

Checklist

- Input used: _____
- Instance of the input used: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the input is LOW (state “0”).		

13.9.4.9 AMF “Standstill monitoring all axes”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if one axis is moved.		

13.9.4.10 AMF “Motion enable”

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the E-STOP is pressed on the smartPAD.		

13.9.4.11 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

- Monitored axis: _____
- Instance of the monitored axis: _____
- Maximum permissible axis torque: _____
- Minimum permissible axis torque: _____

No.	Activity	Yes	Not relevant
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.		

13.9.4.12 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

- Monitored axis: _____
- Instance of the monitored axis: _____
- Maximum permissible axis velocity: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the velocity of the monitored axis exceeds the maximum permissible velocity.		

13.9.4.13 AMF “Position referencing”



This AMF is violated after the robot controller is rebooted.

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if one axis is not referenced.		

13.9.4.14 AMF “Torque referencing”



This AMF is violated after the robot controller is rebooted.

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the torque sensor of one axis is not referenced.		

13.9.4.15 AMF “Axis range monitoring”

Checklist

- Monitored axis: _____
- Instance of the monitored axis: _____
- Lower limit of the permissible axis range: _____
- Upper limit of the permissible axis range: _____

No.	Activity	Yes	Not relevant
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.		

13.9.4.16 AMF “Cartesian velocity”

Description The AMF can be tested by moving a monitored point at a Cartesian velocity of approx. 10% over the configured velocity limit.

Checklist

- Instance of the monitored velocity: _____
- Maximum permissible Cartesian velocity: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the Cartesian velocity of a monitored point exceeds the maximum permissible velocity.		

13.9.4.17 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.

Checklist

- Type of monitoring space: _____
- Instance of the monitoring space: _____
- 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 monitoring space has been tested as above and the configured reaction is triggered if the monitoring space is violated.		

13.9.4.18 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.

Checklist

- Instance of the collision detection: _____
- Maximum permissible external torque: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the external torque of one axis exceeds the maximum permissible external torque.		

13.9.4.19 AMF “TCP force monitoring”

Description In order to test the AMF, suitable measuring equipment is required, e.g. a spring balance.

Checklist

- Instance of the TCP force monitoring: _____
- Maximum permissible external TCP force: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the external force acting on the TCP exceeds the maximum permissible force.		

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 205](#))

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 motions (PTP)
([>>> 14.2 "PTP motion type" Page 185](#))
- Linear motions (LIN)
([>>> 14.3 "LIN motion type" Page 186](#))
- Circular motions (CIRC)
([>>> 14.4 "CIRC motion type" Page 186](#))

The following types of motion can be programmed as segments of a CP spline block:

- Linear motions (LIN)
- Circular motions (CIRC)
- Polynomial motions (SPL)

The following types of motion can be programmed as segments of a JP spline block:

- Point-to-point motions (PTP)
([>>> 14.6 "Spline motion type" Page 187](#))

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.

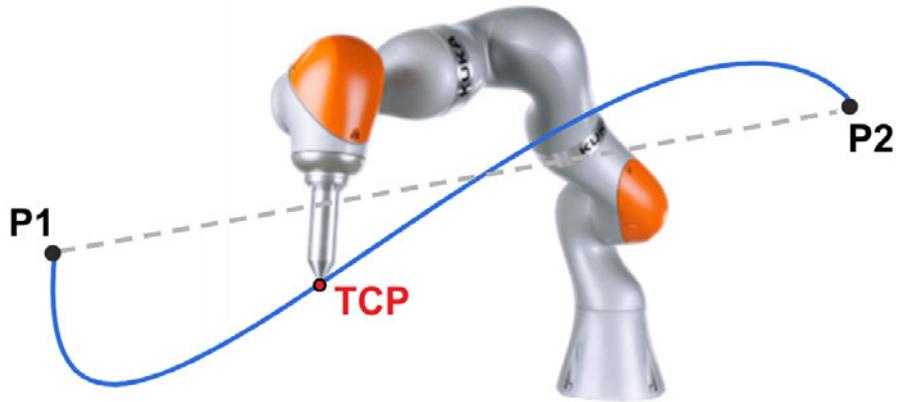


Fig. 14-1: PTP motion

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.

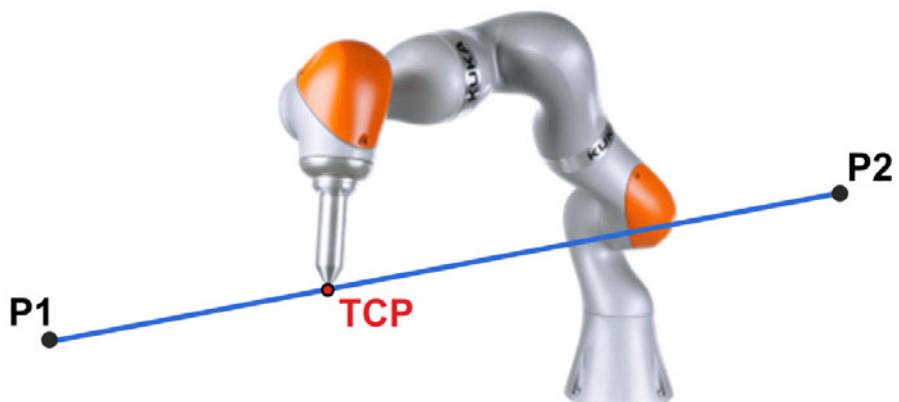


Fig. 14-2: LIN motion

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.

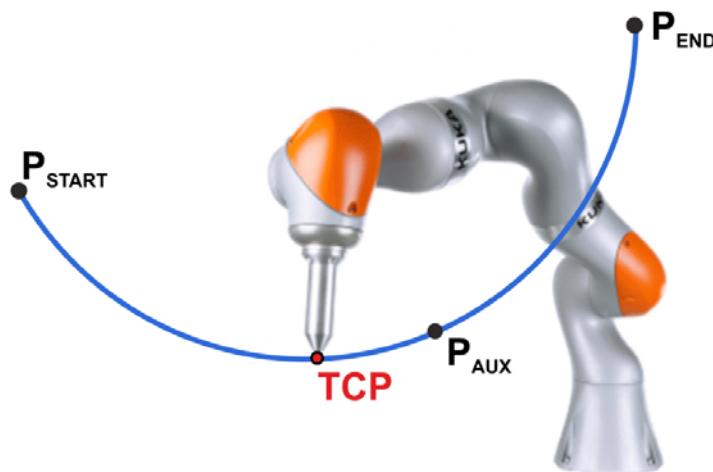


Fig. 14-3: CIRC motion

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

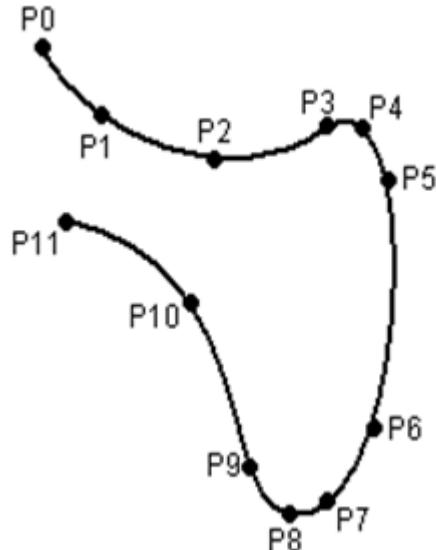
Spline is a motion type that is particularly suitable for complex, curved paths. With a spline motion, the robot can execute these complex paths in a continuous motion. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, however.

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.

In a spline motion, the robot configuration of the end pose is not taken into account.

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.

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 188)
 - 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 188)
- 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.

(>>> 14.8 "Orientation control with LIN, CIRC, SPL" Page 195)

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.

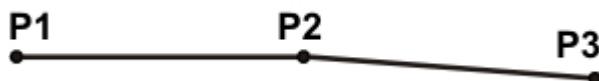


Fig. 14-5: Exact positioning at P2

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.

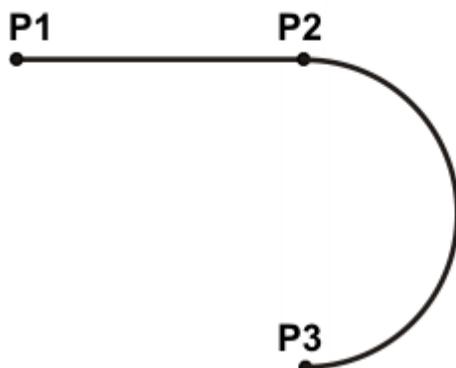


Fig. 14-6: Exact positioning at P2

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(  
    spl(getApplicationData().getFrame("/P1")),  
    spl(getApplicationData().getFrame("/P2")),  
    spl(getApplicationData().getFrame("/P3")),  
    spl(getApplicationData().getFrame("/P4")),  
    circ(getApplicationData().getFrame("/P5"),  
        getApplicationData().getFrame("/P6")),  
    spl(getApplicationData().getFrame("/P7")),  
    lin(getApplicationData().getFrame("/P8"))  
);  
...  
robot.move(ptp(getApplicationData().getFrame("/P0")));  
robot.move(mySpline);
```

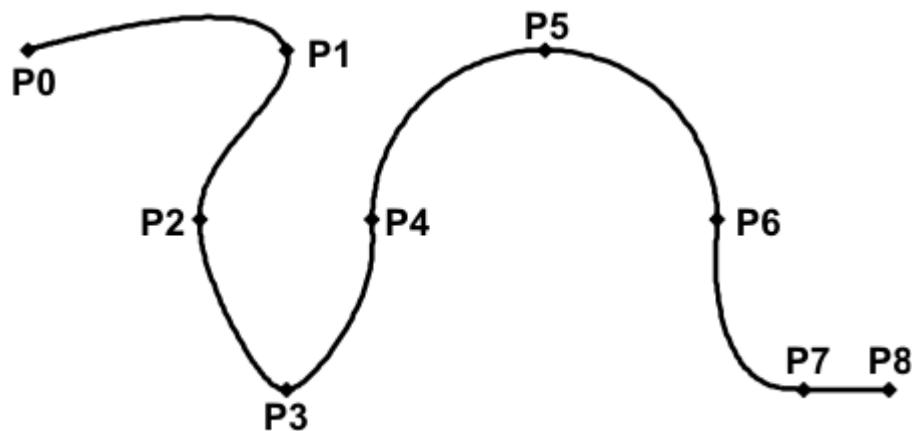


Fig. 14-8: Original path

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.

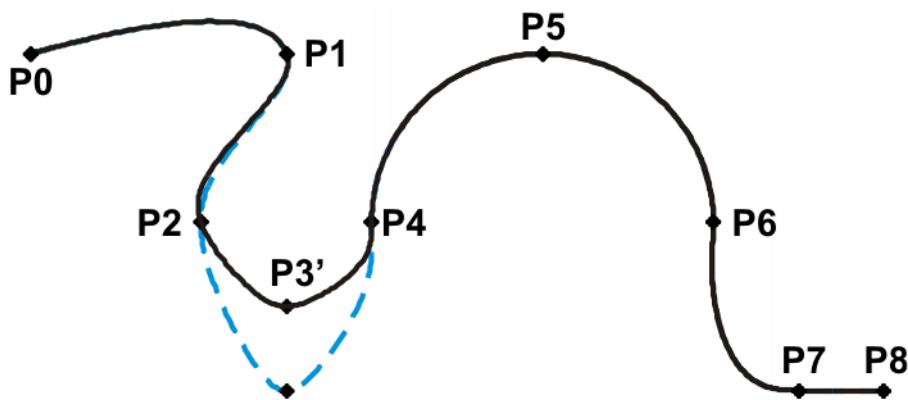


Fig. 14-9: Point has been offset

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(
    spl(getApplicationData().getFrame("/P1")),
    spl(getApplicationData().getFrame("/P2")),
    lin(getApplicationData().getFrame("/P3")),
    spl(getApplicationData().getFrame("/P4")),
    circ(getApplicationData().getFrame("/P5")),
    getApplicationData().getFrame("/P6")),
    spl(getApplicationData().getFrame("/P7")),
    lin(getApplicationData().getFrame("/P8"))
);
...
robot.move(ptpgetApplicationData().getFrame("/P0")));
robot.move(mySpline);
```

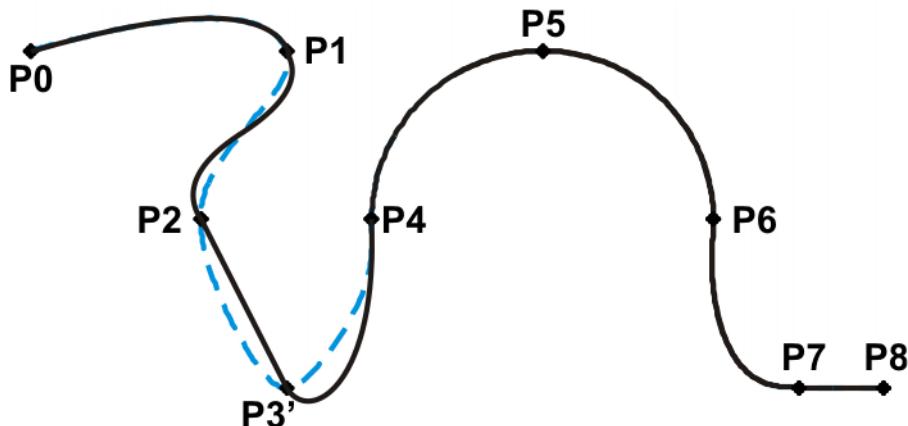


Fig. 14-10: Segment type has been changed

Example 2

Original path:

```
Spline mySpline = new Spline(
    spl(getApplicationData().getFrame("/P2")),
    spl(getApplicationData().getFrame("/P3")),
    spl(getApplicationData().getFrame("/P4")),
    spl(getApplicationData().getFrame("/P5")),
);
...
robot.move(mySpline);
```

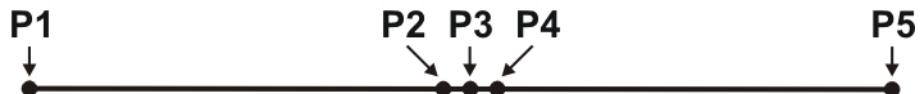


Fig. 14-11: Original path

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.

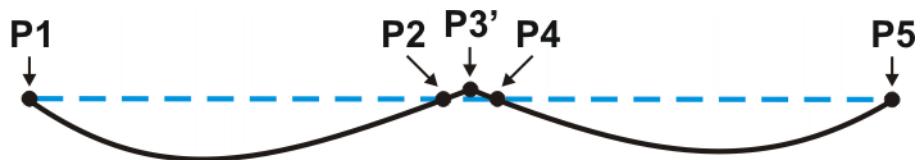


Fig. 14-12: Point has been offset

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.

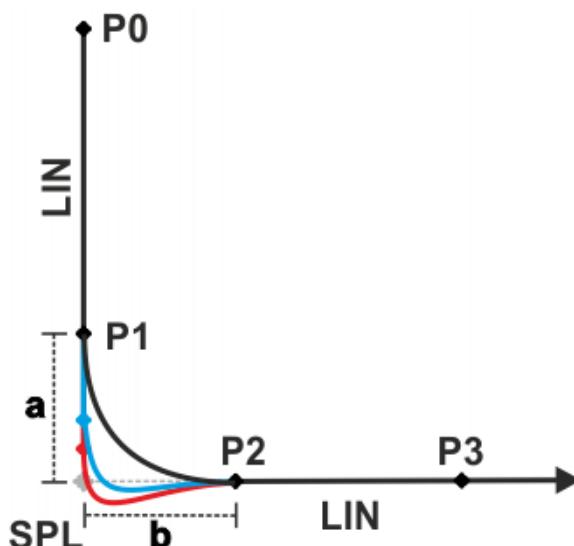


Fig. 14-13: LIN-SPL-LIN

The path remains inside the smaller angle if the following conditions are met:

- The extensions of the two LIN segments intersect.
 - $\frac{2}{3} \leq a/b \leq \frac{3}{2}$
- 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 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.



To approximate motions without exact positioning, they must be executed asynchronously or grouped in a MotionBatch.

(>> 15.6.1 "Structure of a motion command (move/moveAsync)"

Page 211)

(>> 15.6.6 "MotionBatch" Page 215)



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.

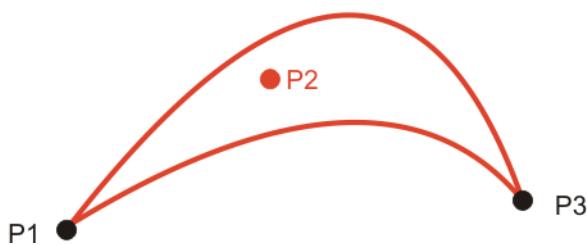


Fig. 14-14: PTP motion, P2 is approximated

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.

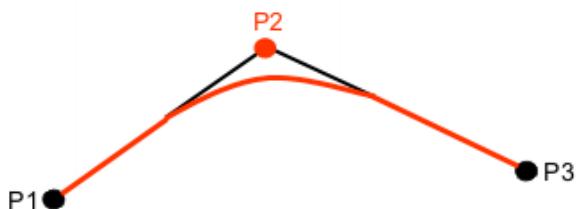


Fig. 14-15: LIN motion, P2 is approximated

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.

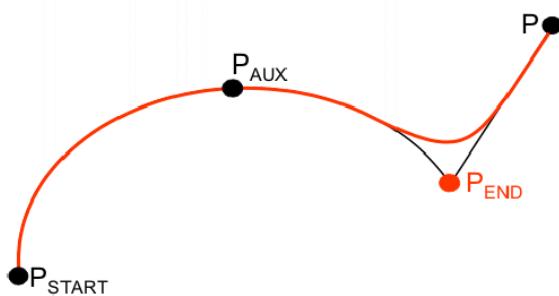


Fig. 14-16: CIRC motion, P_{END} is approximated

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.

14.8 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. (>>> "Ignore" Page 196)</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>
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 applies as the default.</p>

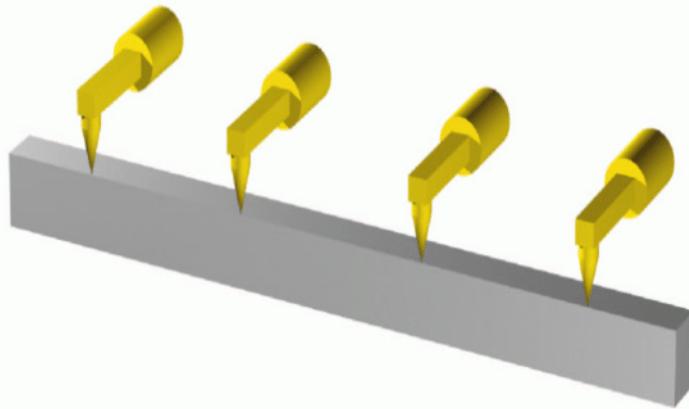


Fig. 14-17: Constant orientation (constant)

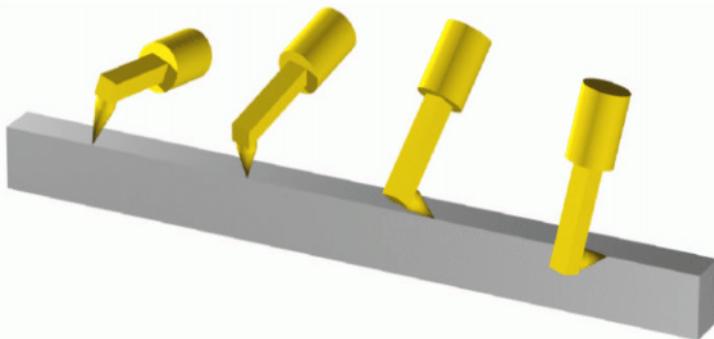


Fig. 14-18: Variable orientation (VariableOrientation or OriJoint)

CIRC motion

It is possible to define for CIRC motions whether the orientation control is to be space-related or path-related.

(>>> 14.8.1 "CIRC – reference system for the orientation control" Page 197)

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

14.8.1 CIRC – reference system for the orientation control

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));
```

Restriction OrientationReferenceSystem.Path is not allowed for the following spline segments:

- CIRC segments with SplineOrientationType.Ignore
- CIRC segments preceded by a segment with SplineOrientationType.Ignore

14.8.2 CIRC – combinations of reference system and type for the orientation control



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

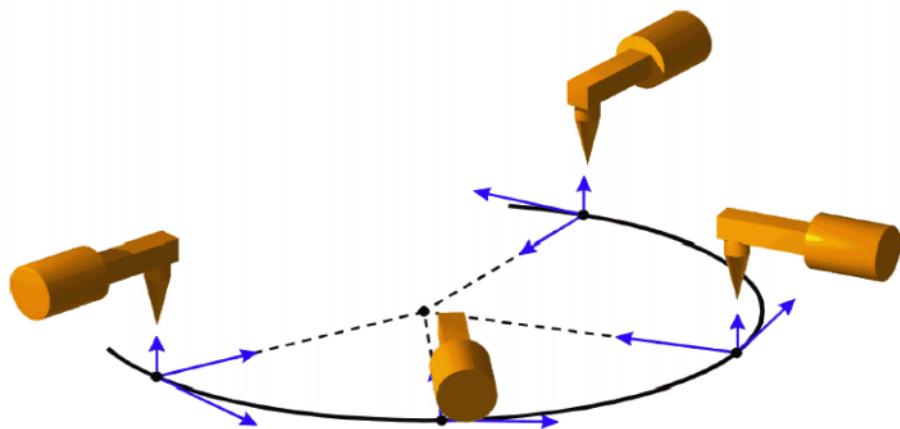


Fig. 14-19: Constant orientation, path-related

Path-related circular motion with variable orientation:

- OrientationReferenceSystem.Path
- SplineOrientationType.VariableOrientation

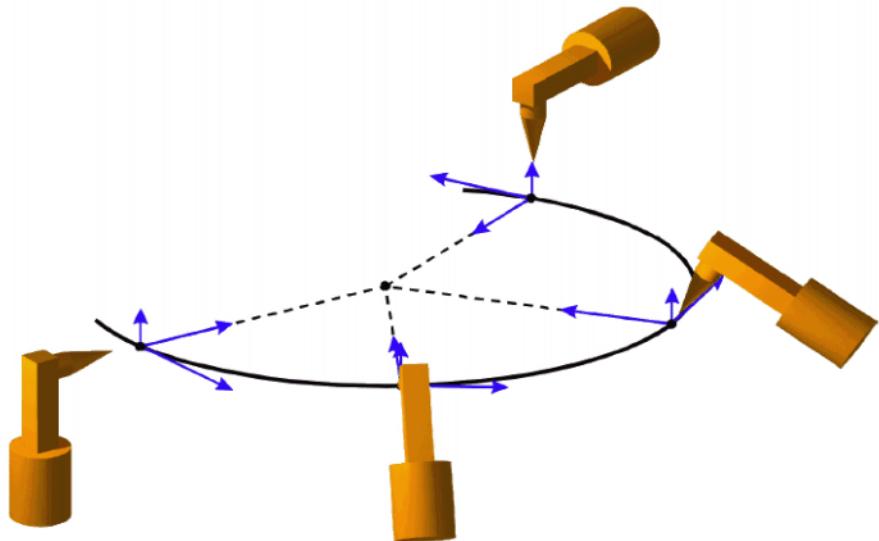


Fig. 14-20: Variable orientation, path-related

Base-related circular motion with constant orientation:

- OrientationReferenceSystem.Base
- SplineOrientationType.Constant

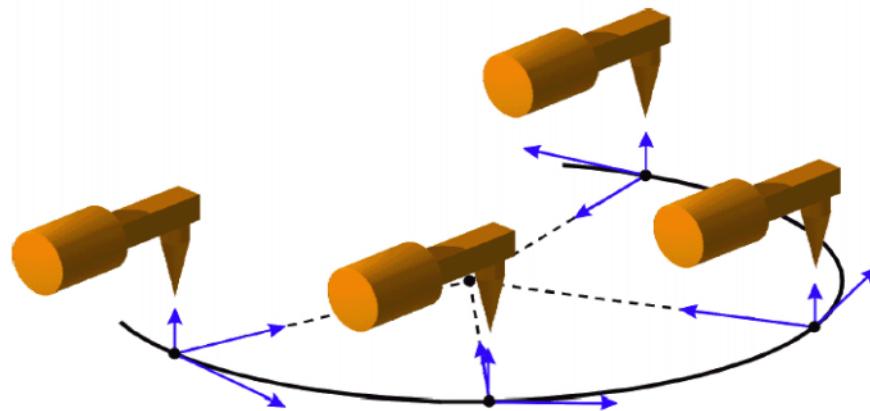


Fig. 14-21: Constant orientation, base-related

Base-related circular motion with variable orientation:

- OrientationReferenceSystem.Base
- SplineOrientationType.VariableOrientation

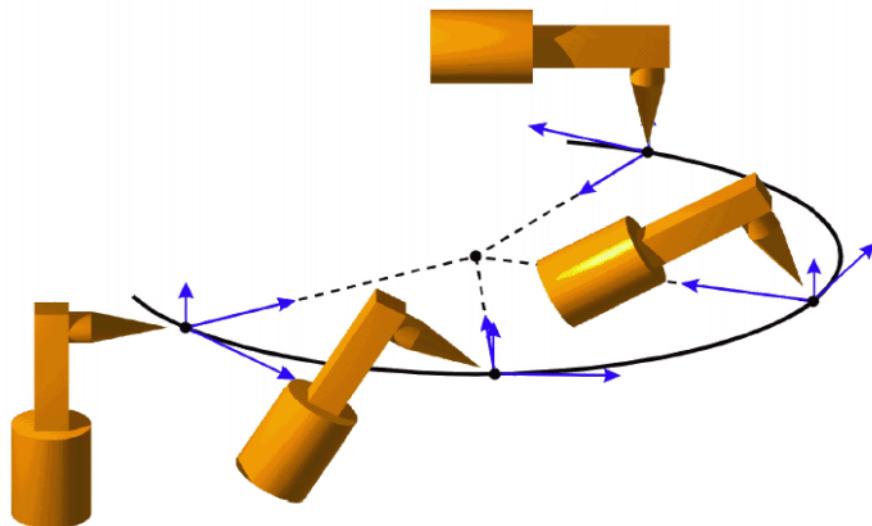


Fig. 14-22: Variable orientation, base-related

14.9 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, the KUKA LBR iiwa can theoretically reach a given position and orientation with any number of axis poses. To unambiguously determine the axis pose for an LBR iiwa, the redundancy angle, in addition to the Status, must be specified.

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(getApplicationData().getFrame(String frameName));
```

14.9.1 Redundancy angle

With its 7th axis, the KUKA LBR iiwa 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 iiwa, 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.

14.9.2 Status

The Status specification prevents ambiguous axis positions. The Status is described by a binary number with 3 bits.

Bit 0

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

In an LBR iiwa, bit 1 specifies the position of axis A4.

Position	Value
A4 < 0°	Bit 1 = 1
A4 ≥ 0°	Bit 1 = 0

Bit 2

In an LBR iiwa, 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 at the start of motion by the axis configuration 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.OriJoint 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.

14.9.3 Turn

The Turn specification makes it possible to move axes through angles greater than +180° or less than -180° without the need for special motion strategies (e.g. auxiliary points). 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 ≥ 0°

Bit = 1: Angle < 0°

Value	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A7 ≥ 0°	A6 ≥ 0°	A5 ≥ 0°	A4 ≥ 0°	A3 ≥ 0°	A2 ≥ 0°	A1 ≥ 0°
1	A7 < 0°	A6 < 0°	A5 < 0°	A4 < 0°	A3 < 0°	A2 < 0°	A1 < 0°

The Turn is not taken into account in an LBR iiwa because none of its axes can rotate over ±180°.

14.10 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.



Recommendation: Move the robot as slowly as possible near singularities.

14.10.1 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 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.

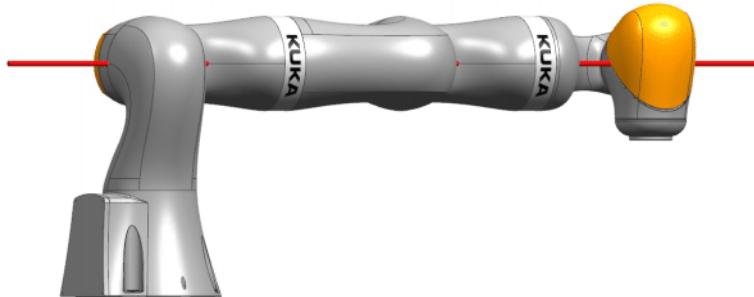


Fig. 14-23: Extended position $A4 = 0^\circ$

Motion is blocked in the direction of the robot base or parallel to axis A3 or A5. An additional kinematic condition for this singularity is reaching the workspace limit. It is automatically met through $A4 = 0^\circ$.

An extended robot arm causes a degree of freedom for the motion of the wrist root point to be lost (it can no longer be moved along the axis of the robot arm). The position of axes A3 and A5 can no longer be resolved.

A4/A6 singularity

This kinematic singularity is given when $A4 = 90^\circ$ and $A6 = 0^\circ$.

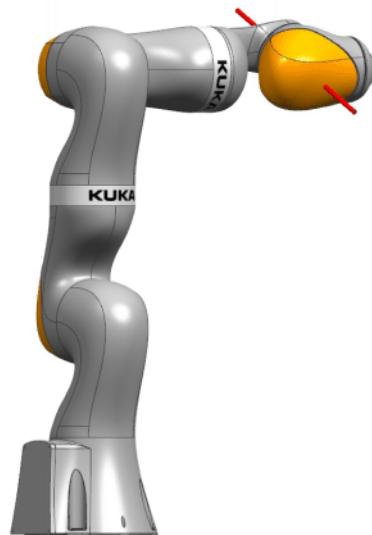


Fig. 14-24: $A_4 = 90^\circ$ and $A_6 = 0^\circ$

Motion parallel to axis A6 or A2 is blocked.

A2/A3 singularity

This kinematic singularity is given when $A_2 = 0^\circ$ and $A_3 = \pm 90^\circ$ ($\pi/2$).

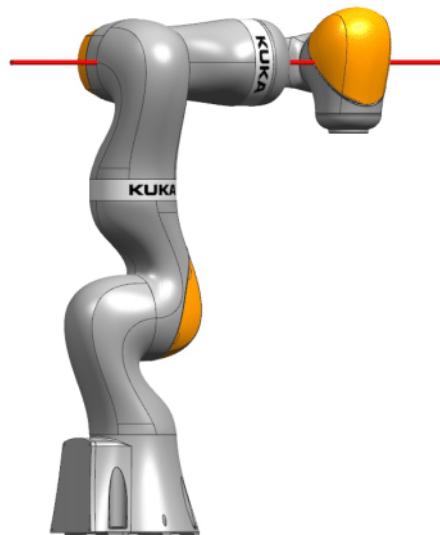


Fig. 14-25: $A_2 = 0^\circ$ and $A_3 = \pm 90^\circ$ ($\pi/2$).

Motion is blocked in the direction of the robot or parallel to axis A2 or 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 $A_6 = 0^\circ$

Motion parallel to axis A6 is blocked.

14.10.2 System-dependent singularities

The redundant configuration of the LBR with its 7th axis allows the robot arm to move without the flange moving. In this so-called “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.

For all of the robot settings in this category, slight Cartesian changes result in very large changes to the axis angles. They are very similar to the singularities in 6-axis robots, since a division is also made in the position and orientation part of the wrist root point in the LBR.

Wrist axis singularity

Wrist axis singularity means the axis position $A_6 = 0^\circ$. Thus the position of axes A5 and A7 cannot be restored and 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 A1, no reference value can be given for the redundancy circle according to the definition above, because any A1 value is permissible here for $A_3 = 0^\circ$.

Every axis position of A1 can be compensated for with a combination of A5, A6 and A7 so that the flange position remains unchanged.

A2 singularity

With an extended “shoulder”, the position of axes A1 and A3 can no longer be resolved according to the pattern above.

A2/A4 singularity

If A1 and A7 coincide, the position of axes A1 and A7 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

- The robot application has been created.
(>>> 5.4 "Creating a new robot application" Page 49)

Procedure

1. Double-click on a Java file in the **Package Explorer**.
Or: Select the file and the menu sequence **Navigate > Open**.
Or: Right-click on the file and select **Open** or **Open With > Java Editor** from the context menu.
2. To close the file: Click on the “X” at the top right of the corresponding tab.

15.1.2 Structure of a robot application

```

1 package application;
2 import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
3 public class RobotApplication extends RoboticsAPIApplication {
4     private Controller kuka_Sunrise_Cabinet_1;
5     private LBR lbr_iwa_7_R800_1;
6     public void initialize() {
7         kuka_Sunrise_Cabinet_1 = getController("KUKA_Sunrise_Cabinet_1");
8         lbr_iwa_7_R800_1 = (LBR) getRobot(kuka_Sunrise_Cabinet_1,
9             "LBR_iwa_7_R800_1");
10    }
11    public void run() {
12        lbr_iwa_7_R800_1.move(ptpHome());
13    }
14    /**
15     * Auto-generated method stub. Do not modify the contents of this method.
16     */
17    public static void main(String[] args) {
18        RobotApplication app = new RobotApplication();
19        app.runApplication();
20    }
21}

```

Fig. 15-1: Structure of a robot application

Item	Description
1	This line contains the name of the package in which the application is located.
2	The import section contains the imported classes which are required for programming the application. Note: Clicking on the “+” icon opens the section, displaying the imported classes.
3	Header of the application (contains the class names of the application) (>>> "Header" Page 206)

Item	Description
4	<p>Declaration section</p> <p>The data arrays of the classes required in the application are declared here. When the application is created, one instance is automatically created for each of the following classes.</p> <ul style="list-style-type: none"> ■ Controller: robot controller used ■ LBR: robot used
5	<p>initialize() method</p> <p>In this method, the data arrays created in the declaration section are assigned initial values.</p>
6	<p>run() method</p> <p>The robot is programmed in this method. When the application is created, a motion instruction which moves the robot to the HOME position is automatically inserted.</p> <p>(>>> 15.14 "HOME position" Page 237)</p>
7	<p>main() method</p> <p>This method is necessary for the application to be executed. It must not be changed.</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 publically 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 a variable

- 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.
 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

An auto-complete function is available in the Java Editor.

When entering the dot operator for a data array or enum, a list containing the following elements is automatically displayed:

- Available methods of the corresponding class (only for data arrays)
- Available constants of the corresponding class

If required, an element can be selected from the list and inserted in the program text using the Enter key. This makes it unnecessary to type the complex

syntax of methods, for example. All that is then required is to enter the variable elements in the syntax manually.

The available template suggestions can also be displayed for a selected element (Press the CTRL and space bar keys simultaneously).

If an element is selected, the Javadoc information on this element is displayed automatically. ([>>> 15.1.4 "Displaying information in Javadoc" Page 208](#))



Navigation in the “Auto-complete” list:

■ Up or down with the arrow keys

■ Or: Type the first letter of the desired element. The marker jumps to the relevant position.

15.1.3.3 Templates – Fast entry of Java statements

Description	Templates for fast entry are available in the Java Editor for common Java statements, e.g. a FOR loop.
Procedure	<ol style="list-style-type: none"> 1. Begin typing the code. 2. Press the CTRL key and space bar simultaneously. 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	<p>Selecting templates in the Templates view:</p> <ol style="list-style-type: none"> 1. Select the menu sequence Window > Show View > Other.... The Show View window opens. 2. In the General folder, select Templates. Click on OK to confirm. The Templates view opens. 3. Position the cursor in the line in which the code template is to be inserted. 4. In the Templates view, double-click on the desired template under Java statements. The code is inserted in the editor. 5. Complete the syntax.

15.1.3.4 Creating user-specific templates

Description	Users can create their own templates, e.g. templates for motion blocks with specific motion parameters which are used frequently during programming.
Procedure	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 1. Select the desired program code. 2. Right-click in the editor area. 3. Select Refactor > Extract Method... from the context menu. The Extract Method window opens. 4. In Method name, enter a unique method name and select the desired Access modifier. Click on OK to confirm. <p>The selected program code is removed and the new method is created. The new method is called in the position in which the code was just removed.</p>
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 information in Javadoc

Description	During programming, the Javadoc information relating to a certain class can be displayed in different ways, either completely or partially. The information is only available in English.
Procedure	<p>Display in a window in the editor area:</p> <ul style="list-style-type: none"> ■ If, during auto-complete, an element is selected from the displayed list, the Javadoc information relating to this element is automatically displayed. ■ Moving the mouse pointer over the name of a method, class (data type, not a user-defined data array name), interface or Enum in the program text automatically displays the corresponding Javadoc information.



Left-clicking on these elements in the program text additionally displays the Javadoc information in the **Javadoc** view.

Procedure	Pinning the window with the Javadoc information in the editor area:
------------------	---

- If the window was opened during auto-complete, press the tab key or click inside the window.
- If the window was opened with the mouse pointer, press the F2 key or click inside the window.

It may be necessary to pin the window if the complete information is too extensive to be displayed. Pinning the window allows the user to scroll through it.

Example

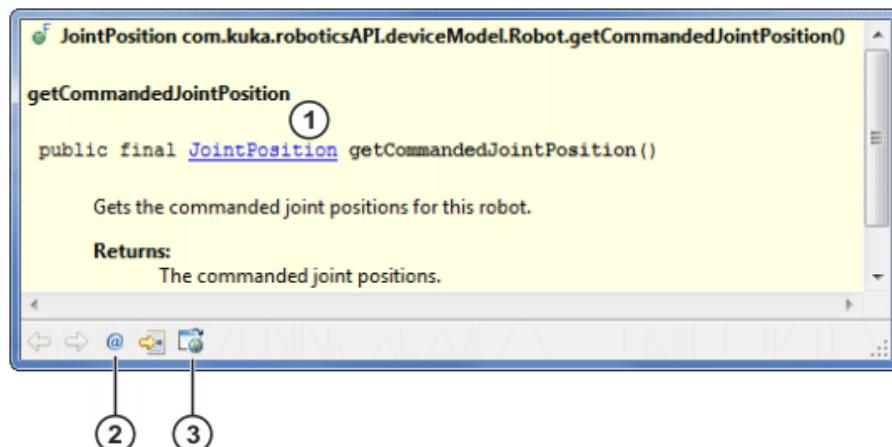


Fig. 15-2: Example of Javadoc description in a pinned window

Item	Description
1	Linked class Left-clicking on the linked class displays the complete Javadoc information relating to this class in the Javadoc browser. Note: If the corresponding link in the Javadoc view is selected, the complete Javadoc information is displayed in the view itself.
2	Show in Javadoc View button The window in the editor section closes and the Javadoc information is displayed in the Javadoc view.
3	Open Attached Javadoc Browser button The window in the editor section closes and the complete Javadoc information relating to the corresponding class is displayed in the Javadoc browser.

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 Examples: <code>private; new; linRel; Tool</code>
Elements that must be replaced by program-specific entries	<ul style="list-style-type: none"> ■ Italics ■ Upper/lower-case letters Examples: <code>endpoint; name; mode</code>
Optional elements	<ul style="list-style-type: none"> ■ In angle brackets Example: <code><.setVelocity (value) ></code>
Elements that are mutually exclusive	<ul style="list-style-type: none"> ■ Separated by the “ ” symbol Example: <code>++ --</code>

15.3 Data types

Overview

There are 2 kinds of data type in Java:

- Primitive data types
- Complex data types

Complex data types are defined in Java by classes.

Overview of important data types:

Data type	Description
int	Integer ■ $-2^{31}-1 \dots +2^{31}-1$ Examples: -1; 32; 8000
double	Double-precision floating-point number ■ $-1.7E+308 \dots +1.7E+308$ 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 characters Examples: "KUKA"; "tool"



The names of the primitive data types are displayed in violet in the Java Editor.

15.4 Variables

Description

Before a variable can be used in the program, it must be declared, i.e. the data type and identifier must be defined. A variable can be declared in the run() method of an application, for example.

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;
boolean isObjectPlaced;
```

15.5 RoboticsAPI version information

RoboticsAPI is the programming interface for all robot-specific commands. It has a 4-figure version number. When comparing several versions, this provides information about changes to the interface.

15.5.1 Displaying the RoboticsAPI version

The RoboticsAPI version number of a project can be found in the file Station-Setup.cat (station configuration).

Procedure	<ol style="list-style-type: none"> 1. Open the project in the Package Explorer. 2. Open the station configuration 3. Select the Software tab. 4. Activate the Show Libraries check box (set the check mark).
Description	<p>The RoboticsAPI version number is displayed in the BasicRoboticsJavaLib row.</p> <ul style="list-style-type: none"> ■ Currently installed version column: version last installed on the controller ■ Selected version column: version currently being used in the Sunrise project

15.5.2 Structure of the RoboticsAPI version number:

The RoboticsAPI version number has the format XX.YY.ZZ.Build number.

- XX: Release version
- YY: Incompatible changes between versions
In applications created while using the older API version, errors may occur when changing over to a more current version.
- ZZ: Compatible changes between versions
Applications created while using the older API version can be also be used with the current version.
- Build number: Assigned automatically.

15.6 Motion programming: PTP, LIN, CIRC

15.6.1 Structure of a motion command (move/moveAsync)

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 methods move(...) and moveAsync(...) are available to this end:

- 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.

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:

(>>> 15.9.4 "Moving tools and workpieces" Page 225)



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 This specifies the object variable name which was declared and initialized in the application.
<i>Movement</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■ End 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 (getApplicationData () .get-
Frame ("End_point") <.Motion_parameter>)
```

PTP motion with specified axis angles:

```
ptp (A1, A2, ... A7) <.Motion_parameter>
```

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 parameter</i>	Further motion parameters, e.g. velocity and acceleration

Examples

PTP motion to the "StartPos" frame:

```
robot.move(ptpgetApplicationData().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 parameter>)
```

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 parameter</i>	Further motion parameters, e.g. velocity and acceleration

Examples

LIN motion to the “/Table/P1” frame:

```
robot.move(lingetApplicationData().getFrame("/Table/P1"));
```

LIN motion with the Cartesian velocity specified:

```
robot.move(lingetApplicationData().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 parameter>)
```

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 parameter</i>	Further motion parameters, e.g. velocity and acceleration

Examples

CIRC motion to the end frame “/Table/P4” via the auxiliary frame “/Table/P3”:

```
robot.move(circgetApplicationData().getFrame("/Table/P3"),
getApplicationData().getFrame("/Table/P4"));
```

CIRC motion with the absolute acceleration specified:

```
robot.move(circgetApplicationData().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 by default offset 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)

Element	Description
a, b, c	<p>Rotation about the Z, Y or X axis (type: double)</p> <p>The unit depends on the method used:</p> <ul style="list-style-type: none"> ■ Offset values and Transformation.ofRad: rad ■ Transformation.ofDeg: degrees
<i>Reference system</i>	<p>Type: AbstractFrame</p> <p>Reference coordinate system in which the motion is executed</p>

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 from the current position in the tool coordinate system. 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 – setJoint...Rel(...) – can be specified for the entire batch. Cartesian motion parameters – 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 individual block parameter overwrites the batch parameter. 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

Element	Description
<i>Motion</i>	Motion with or without motion parameters ■ ptp, lin, circ or spline
<i>Motion parameter</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

- A spline block should cover only 1 process (e.g. 1 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:
 - a. First teach or calculate a few characteristic points. Example: points at which the curve changes direction.
 - b. Test the path. At points where the accuracy is still insufficient, add more SPL points.
- Avoid successive 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.

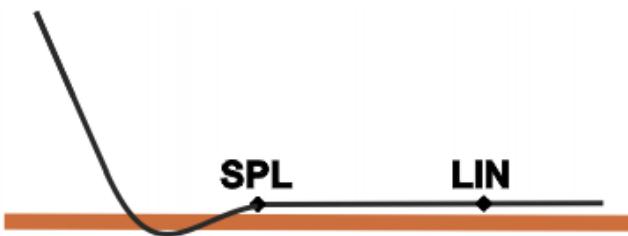


Fig. 15-3: Collision with work surface

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 192)

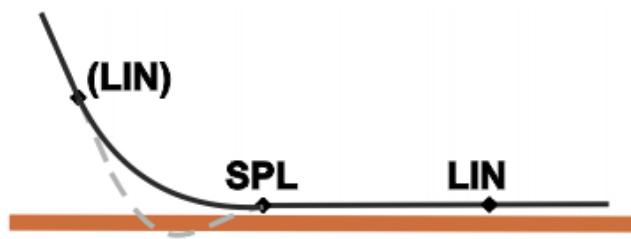


Fig. 15-4: Avoiding a collision with the work surface

- 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. The maximum number of spline segments in a spline block is at present limited to 20.

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 individual block parameter overwrites the block parameter. This also applies if a lower parameter value is specified for the block than for the individual block.

Syntax

```
Spline Name = new Spline(
    Segment,
    Segment,
    ...
    Segment,
    Segment
) < .Motion parameter>;
```

Explanation of the syntax

Element	Description
<i>Name</i>	Name of the spline block
<i>Segment</i>	Motion with or without motion parameters <ul style="list-style-type: none"> ■ spl, lin or circ
<i>Motion parameter</i>	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"),
    getgetApplicationData().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. The maximum number of spline segments in a spline block is at present limited to 20.

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 individual block parameter overwrites the block parameter. This also applies if a lower parameter value is specified for the block than for the individual block.

Syntax

```
SplineJP Name = new SplineJP(  
    Segment,  
    Segment,  
    ...  
    Segment,  
    Segment  
) < .Motion parameter>;
```

Explanation of the syntax

Element	Description
<i>Name</i>	Name of the spline block
<i>Segment</i>	PTP motion with or without motion parameters
<i>Motion parameter</i>	Motion parameters which are programmed at the end of the spline block apply to the entire spline block.

Example

```
SplineJP mySpline = new SplineJP(  
    ptp(getApplicationContext().getFrame("/P1")),  
    ptp(getApplicationContext().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 name) ;
```

Explanation of the syntax

Element	Description
<i>Object</i>	Object of the station which is being moved
<i>Spline name</i>	Name of the spline block

Example

```
robot.move (mySpline) ;
```

15.8 Motion parameters

The required motion parameters can be added in any order to the motion instruction. Dot separators and “set” methods are used for this purpose.

Overview

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>(>>> 15.8.1 "Programming axis-specific motion parameters" Page 220)</p>
setCartAcceleration(...)	<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 velocity (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>(>>> 15.8.1 "Programming axis-specific motion parameters" Page 220)</p>
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 220)</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) always corresponds to the block length. For motions which are not commanded within a spline, however, 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>

Method	Description
setBlendingCart(...)	Absolute approximation distance (type: double, unit: mm) <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(...)	Orientation parameter for approximate positioning (type: double, unit: rad) <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>
setOrientation-Type(...)	Orientation control (type: Enum) <ul style="list-style-type: none"> ■ Constant ■ Ignore ■ OriJoint ■ VariableOrientation (default) <p>(>>> 14.8 "Orientation control with LIN, CIRC, SPL" Page 195)</p>
setOrientationReferenceSystem(...)	Only relevant for CIRC motions: Reference system of orientation control (type: Enum) <ul style="list-style-type: none"> ■ Base ■ Path <p>(>>> 14.8.1 "CIRC – reference system for the orientation control" Page 197)</p>

15.8.1 Programming axis-specific motion parameters

Description	<p>The following axis-specific motion parameters can be programmed:</p> <ul style="list-style-type: none"> ■ Relative velocity <code>setJointVelocityRel(...)</code> ■ Relative acceleration <code>setJointAccelerationRel(...)</code> ■ Relative jerk <code>setJointJerkRel(...)</code> <p>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.</p> <p>By way of example, these possibilities are described using the relative velocity:</p> <ul style="list-style-type: none"> ■ <code>setJointVelocityRel (Value)</code> If a value of type double is transferred, the relative velocity applies to all axes. ■ <code>setJointVelocityRel (Array_variable)</code> 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. ■ <code>setJointVelocityRel (Axis, Value)</code> To specify the relative velocity of an individual axis, this axis is transferred as an Enum of type <code>JointEnum</code>. This Enum has 12 axes (<code>JointEnum.J1 ... JointEnum.J12</code>).
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 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.

The robot controller and robot are automatically declared and initialized when the application is created. Tools and workpieces used in the application must be declared and initialized by the user.

Tools and workpieces with load data and geometric data are created and managed in the **Template data** view.

(>>> 9.3 "Object management" Page 116)

Data types

The data types for the objects in a station are predefined in the RoboticsAPI:

Data type	Object
Controller	Robot controller
LBR	Lightweight robot
Tool	Tool
Workpiece	Workpiece
GeometricObject	Fixed geometric objects

15.9.1 Declaring tools and workpieces

Syntax

```
private Data type Object name;
```

Explanation of the syntax

Element	Description
private	The keyword designates locally valid variables. Locally valid means that the data array can only be used by the corresponding class.
Data type	Object type
Object name	Name of the object variable

Example

In the application, 2 tools (gripper, guiding tool) and 1 workpiece (pen) are used.

```
private Tool gripper;
private Tool guidingTool;
private Workpiece pen;
```

15.9.2 Initializing tools and workpieces

Description	This section describes how tools and workpieces which were created in the object templates of the corresponding project are initialized.						
Syntax	<i>Object name</i> = getApplicationData().createFromTemplate("Object template");						
Explanation of the syntax	<table border="1"> <thead> <tr> <th>Element</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>Object name</i></td><td>Name of the object variable</td></tr> <tr> <td><i>Object template</i></td><td>Name of the object template as specified in the Template data view</td></tr> </tbody> </table>	Element	Description	<i>Object name</i>	Name of the object variable	<i>Object template</i>	Name of the object template as specified in the Template data view
Element	Description						
<i>Object name</i>	Name of the object variable						
<i>Object template</i>	Name of the object template as specified in the Template data view						

Example The following tools and workpieces were created in the object templates:

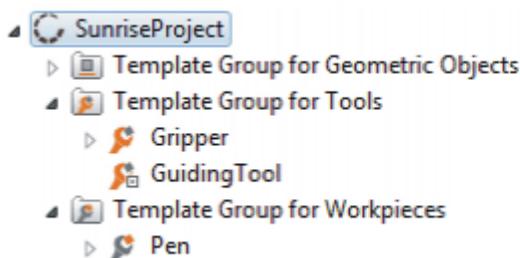


Fig. 15-5: Object templates

```
private Tool gripper;
private Tool guidingTool;
private Workpiece pen;

public void initialize() {
    ...
    gripper = getApplicationData().createFromTemplate("Gripper");
    guidingTool =
        getApplicationData().createFromTemplate("GuidingTool");
    pen = getApplicationData().createFromTemplate("Pen");
    ...
}
```

15.9.3 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.

(>>> 9.3.6 "Load data" Page 120)

15.9.3.1 Attaching a tool to the robot flange

Description	Via the method <code>attachTo(...)</code> , 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 <code>getFlange()</code> .
Syntax	<code>Tool.attachTo(Robot.getFlange());</code>

Explanation of the syntax

Element	Description
Tool	Name of the tool variable
Robot	Name of the robot

Example

A guiding tool is attached to the robot flange.

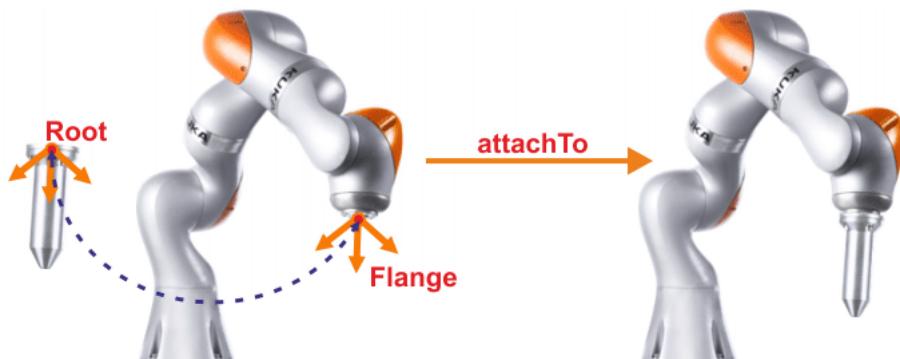


Fig. 15-6: Attaching the guiding tool to the flange.

```
private LBR robot;
private Tool guidingTool;
...
public void run() {
    ...
    guidingTool.attachTo(robot.getFlange());
    ...
}
```

15.9.3.2 Attaching a workpiece to other objects

Description

By default 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 are created in the **Template data** view. In order to use a frame in the program, the tool or workpiece object is polled with the method `getFrame(...)`. As an input parameter, this contains the path of the frame as a string.

(>>> 9.3.4 "Creating a frame for a tool or workpiece" Page 118)

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
Workpiece	Name of the workpiece variable
Reference frame	Reference frame of the workpiece which is used for the attachment to the other object
End frame	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 connected to it match.

Example 1

A pen is attached to the gripper frame via its origin frame.

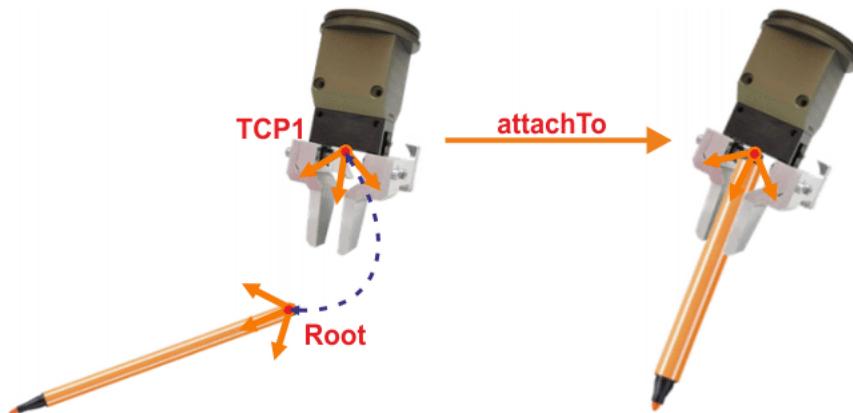


Fig. 15-7: Pen in gripper (attachment via origin frame)

```
private LBR robot;
private Tool gripper;
private Workpiece pen;
...
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, a connection 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.

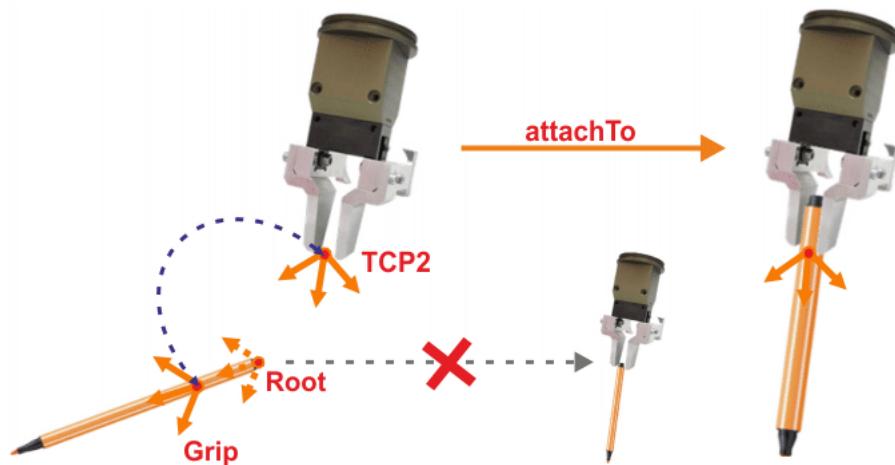


Fig. 15-8: Pen in gripper (connection via grip frame)

```
private LBR robot;
private Tool gripper;
private Workpiece pen;
...
public void run() {
    ...
    pen.getFrame("/Grip").attachTo(gripper.getFrame("/TCP2"));
    ...
}
```

15.9.3.3 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();
```

15.9.4 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 reference point is by default the default motion frame which was defined for this object in the **Template data** view.
([>>> 9.3.5 "Defining a default motion frame" Page 119](#))
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
<code>Object</code>	Object of the station which is being moved This specifies the object variable name which was declared and initialized in the application.
<code>Moved frame</code>	Path to the frame of the object which is used for the motion
<code>Motion</code>	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 Inputs/outputs

Exporting an I/O configuration from WorkVisual automatically creates and inserts the Java package **com.kuka.generated.ioAccess** in the source folder for the project. This package contains one Java class per defined I/O group. 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 these Java classes 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 (aggregating).

To use the inputs/outputs of an I/O group in the application, the user must create and install a data array of the relevant type for the I/O group.

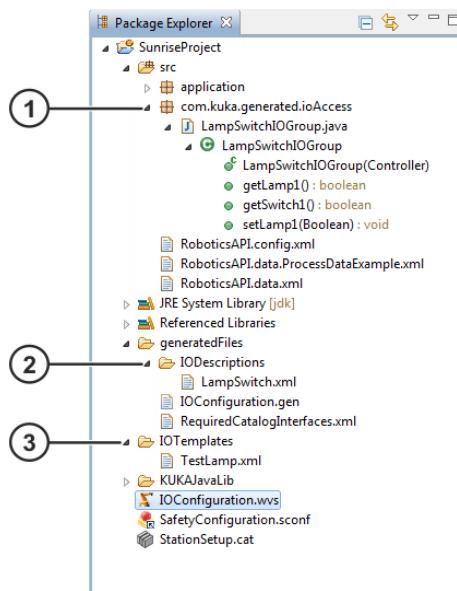


Fig. 15-9: Project structure after exporting the I/O configuration

Item	Description
1	<p>com.kuka.generated.ioAccess Java package</p> <p>The Java class <i>NameIOGroup.java</i> (here: <i>LampSwitchIOGroup.java</i>) 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>

Item	Description
2	generatedFiles > IODescriptions folder The data in an I/O group are saved in an XML file. The XML file can be displayed but not edited.
3	IOTemplates folder 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. A template can be copied into another Sunrise project in order to be used there. The template can then be imported into WorkVisual, edited there and re-exported. (>>> 11.4.8 "Importing an I/O group from a template" Page 134) (>>> 11.4.7 "Exporting an I/O group as a template" Page 133)

15.10.1 Creating a data array for an I/O group

Description Declaring the data array of the type of the I/O group automatically imports the Java package com.kuka.generated.ioAccess with the classes and methods of the I/O group.

Syntax `private Class name Array name;`

Explanation of the syntax

Element	Description
<code>private</code>	The keyword <code>private</code> designates data arrays which can only be used by the surrounding class.
<code>Class name</code>	Type of the data array Class name of the I/O group: ■ <code>NameIOPgroup</code> <code>Name</code> = Name of the I/O group, as defined in WorkVisual
<code>Array name</code>	Name of the data array used in programming

Example

For the I/O group “SwitchLamp”, the data array “switchLamp” is created.

```
public class RobotApplication extends RoboticsAPIApplication {
    ...
    private Controller controller;
    private SwitchLampIOPgroup switchLamp;
    ...
}
```

15.10.2 Initializing a data array for an I/O group

Description When the data array is initialized, the robot controller to which the inputs/outputs of the group are connected via the field bus, is specified via the constructor of the class.

Syntax `Array name = new Class name (Controller) ;`

Explanation of the syntax

Element	Description
<code>Array name</code>	Name of the data array
<code>new</code>	Operator with which a new instance of the class <code>Class name</code> is created

Element	Description
<i>Class name</i>	Class name of the I/O group: ■ <i>NameIOGroup</i> <i>Name</i> = Name of the I/O group, as defined in WorkVisual
<i>Controller</i>	Name of the object assigned to the robot controller The assignment generally takes place in the initialize() method of a robot application. (>>> 15.1.2 "Structure of a robot application" Page 205)

Example

The data array “switchLamp” is initialized.

```
public void initialize() {
    ...
    switchLamp = new SwitchLampIOGroup(controller);
    ...
}
```

15.10.3 Reading inputs/outputs**Description**

The “get” method of an input/output is used to poll the state of the input/output.

Syntax

Array name.*get Input/output*() ;

Explanation of the syntax

Element	Description
<i>Array name</i>	Name of the data array
<i>Input/output</i>	Name of the input/output (as defined in WorkVisual)

Example

The state of the switch at input “Switch1” and of the lamp at output “Lamp1” is polled.

```
public void run() {
    ...
    switchLamp.getLamp1();
    switchLamp.getSwitch1();
    ...
}
```

15.10.4 Setting outputs**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

Array name.*set Output*(*Value*) ;

Explanation of the syntax

Element	Description
<i>Array name</i>	Name of the data array
<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() {
    ...
    switchLamp.setLamp1(true);
    ThreadUtil.milliSleep(2000);
    switchLamp.setLamp1(false);
    ...
}

```

15.11 Polling axis torques

Description

A joint torque sensor which measures the torque acting on the axis is located in each axis of the KUKA LBR iiwa. The measured torque values can be polled and evaluated in the application via the method `getMeasuredTorque()` of the LBR class.

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 structure and mass inertias during motion. These data can be accessed via the LBR method `getExternalTorque()`.

Both commands return an object of the type `TorqueSensorData`, which contains the torque sensor data for all axes. From this object, it is then possible to poll either all values as an array with `getTorqueValues(...)` or a single axis value with `getSingleTorqueValue(...)`.

Syntax

To poll the measured sensor data:

```
TorqueSensorData measuredData = lbr.getMeasuredTorque();
```

To poll externally acting torque data:

```
TorqueSensorData externalData = lbr.getExternalTorque();
```

To poll torque values of all axes from the sensor data:

```
double[] allValues = measuredData|externalData.getTorqueValues();
```

To poll torque values of a specific axis from the sensor data:

```
double singleValue =
measuredData|externalData.getSingleTorqueValues(joint);
```

Explanation of the syntax

Element	Description
<code>measuredData</code>	Type: <code>TorqueSensorData</code> Variable for the return value of <code>getMeasuredTorque()</code> . The return value contains the measured sensor data.
<code>externalData</code>	Type: <code>TorqueSensorData</code> Variable for the return value of <code>getExternalTorque()</code> . The return value contains the externally acting torques.
<code>lbr</code>	Type: <code>LBR</code> Name of the robot from which the sensor data are polled
<code>allValues</code>	Type: <code>double[]</code> ; unit: Nm Array with all torque values which are polled from the sensor data
<code>singleValue</code>	Type: <code>double</code> ; unit: Nm Torque value of the axis which is polled from the sensor data
<code>joint</code>	Type: <code>Enum of type JointEnum</code> Axis whose torque value is to be polled ■ JointEnum.J1 ... JointEnum.J12: Axes A1 ... A12

Example

For a specific process step, the measured and externally acting torques are polled 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.

```
TorqueSensorData measuredData = lbr.getMeasuredTorque();
TorqueSensorData externalData = lbr.getExternalTorque();

double[] measuredTorques = measuredData.getTorqueValues();
double[] externalTorques = externalData.getTorqueValues();

double torqueA2 = measuredData.getSingleTorqueValue(JointEnum.J2);
getLogger().info("Currently measured torque for joint 2 [Nm]:" +
torqueA2);
```

15.12 Reading Cartesian forces and torques

It is possible 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 following points must be taken into consideration:

- The torques of the axes are measured by the torque sensors.
- The Cartesian forces and torques are calculated from the measured torques.
- The reliability of the calculated values can decrease considerably in extreme poses, e.g. extended positions or singularities.
- There are methods available in the RoboticsAPI for checking the quality and validity of the calculated values.



When changing the load data, e.g. with the attachTo command, the poll can only be executed after the motion command has been sent to the controller. For this purpose, a null space motion or the motion command positionHold(...) is sufficient.

15.12.1 Polling calculated force/torque data

Description

The method getExternalForceTorque(...) of the LBR class is used to poll the external Cartesian forces and torques currently acting on a specific point.

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 = lbr.getExternalForceTorque(
measureFrame<, orientationFrame>);
```

Explanation of the syntax

Element	Description
<i>data</i>	Type: ForceSensorData Variable for the return value of getExternalForceTorque(...). The return value contains the polled force/torque data.
<i>lbr</i>	Type: LBR Name of the robot
<i>measure Frame</i>	Type: AbstractFrame Reference frame for which the currently acting forces and torques are calculated
<i>orientation Frame</i>	Type: AbstractFrame Optional: Orientation of the frame in which the forces and torques are represented.

Examples

Poll of the force/torque data on the robot flange:

```
ForceSensorData data =
    robot.getExternalForceTorque(robot.getFlange());
```

Poll of the force/torque data on the robot flange, with reference to the orientation of the world coordinate system:

```
ForceSensorData data =
    robot.getExternalForceTorque(robot.getFlange(),
        World.Current.getRootFrame());
```

15.12.2 Polling individual force/torque values

Description

The force/torque data read with getExternalForceTorque() can be polled separately from each other via the methods getForce() and getTorque(...) of the ForceSensorData class.

The result of these pollings is a vector in each case. The values for each degree of freedom can be polled individually with the "get" methods of the "Vector" class.

(>>> 15.12.4 "Polling individual values of a vector" Page 233)

Syntax

To poll a force vector:

```
Vector force = data.getForce();
```

To poll a torque vector:

```
Vector torque = data.getTorque();
```

Explanation of the syntax

Element	Description
<i>force</i>	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the Cartesian forces which act in the X, Y and Z directions (unit: N)
<i>torque</i>	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the Cartesian torques which act about the X, Y and Z axes (unit: Nm)
<i>data</i>	Type: ForceSensorData Variable for the return value of getExternalForceTorque(...). The return value contains the polled force/torque data.

Example

Poll of 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();
```

15.12.3 Checking the reliability of the calculated force/torque values

Description

In unfavorable positions, the calculated force/torque values 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 polled in the program. The following methods of the ForceSensorData class are available:

- `getForceInaccuracy()`, `getTorqueInaccuracy()`

The inaccuracies of the calculated force/torque values can be polled separately from one another.

The result of these pollings is a vector in each case. The values for each degree of freedom can be polled individually with the "get" methods of the Vector class.

(>>> 15.12.4 "Polling individual values of a vector" Page 233)

- `isForceValid(...)`, `isTorqueValid(...)`

It is possible to poll whether the calculated force/torque values are valid. Each method is transferred a limit value for the maximum permissible inaccuracy, up to which point the calculated values are still valid as parameters.

Syntax

Polling the inaccuracy of the calculated force/torque values:

```
Vector force = data.getForceInaccuracy();
Vector torque = data.getTorqueInaccuracy();
```

Polling the validity of the force/torque values:

```
boolean valid = data.isForceValid(tolerance);
boolean valid = data.isTorqueValid(tolerance);
```

Explanation of the syntax

Element	Description
<code>force</code>	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)
<code>torque</code>	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: N)
<code>data</code>	Type: ForceSensorData Variable for the return value of the method <code>getExternalForceTorque(...)</code> . The return value contains the polled force/torque data.

Element	Description
<i>tolerance</i>	Type: double; unit: N or Nm Limit value for the maximum permissible inaccuracy up to which the calculated force/torque values are still valid
<i>valid</i>	Type: Boolean Variable for the return value of <code>isForceValid(...)</code> or <code>isTorqueValid(...)</code> : <ul style="list-style-type: none"> ■ TRUE, if the inaccuracy value in all Cartesian directions is less than or equal to the limit value defined with <i>tolerance</i> ■ FALSE, if 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 currently acting along the axes of the flange coordinate system have been calculated with an accuracy of 20 N or better.

```
ForceSensoData data =
robot.getExternalForceTorque(robot.getFlange());  
  
if (data.isForceValid(20)){
    //do something
}
```

15.12.4 Polling individual values of a vector

Methods which poll 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 polled individually.

Overview

The following methods of the Vector class are available:

Method	Description
<code>getX()</code>	Return value type: double Polls for the X component of the vector
<code>getY()</code>	Return value type: double Polls for the Y component of the vector
<code>getZ()</code>	Return value type: double Polls for the Z component of the vector
<code>get(index)</code>	Return value type: double Polls for 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

15.13 Polling the robot position

The axis-specific and Cartesian robot position can be polled in the application. It is possible to poll the actual and the setpoint position for each.

Overview

The following methods of the Robot class are available:

Method	Description
getCommandedCartesianPosition(...)	Return value type: Frame Polls for the Cartesian setpoint position
getCommandedJointPosition()	Return value type: JointPosition Polls for the axis-specific setpoint position
getCurrentCartesianPosition(...)	Return value type: Frame Polls for the Cartesian actual position
getCurrentJointPosition()	Return value type: JointPosition Polls for the axis-specific actual position
getPositionInformation(...)	Return value type: PositionInformation Polls for the Cartesian position information The return value contains the following information: <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)

15.13.1 Polling the axis-specific actual or setpoint position

Description For polling 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 axis values can then be polled individually. The desired axis is specified either via its index or the corresponding JointEnum value.

Syntax To poll the axis-specific actual position:

```
JointPosition position = robot.getCurrentJointPosition();
```

To poll the axis-specific setpoint position:

```
JointPosition position = robot.getCommandedJointPosition();
```

To poll the axis values individually:

```
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 polled axis positions.
<i>robot</i>	Type: Robot Name of the robot from which the axis positions are polled
<i>value</i>	Type: double; unit: rad Axis angle of the polled axis

Element	Description
<i>axis</i>	Type: int Option 1: specify the index of the axis whose axis value is polled ■ 0 ... 11: axes A1 ... A12
	Type: Enum of type JointEnum Option 2: specify the JointEnum value of the axis whose axis value is polled ■ JointEnum.J1 ... JointEnum.J12: Axes A1 ... A12

Example

First the axis-specific actual position of the robot and then the axis value of axis A3 are polled via the index. The angle for axis A4 is displayed in degrees on the smartHMI.

```
JointPosition actPos = lbr.getCurrentJointPosition();
double a3 = actPos.get(2);
getLogger().info(Math.toDegrees(a3));
```

15.13.2 Polling the Cartesian actual or setpoint position**Description**

It is possible to poll the Cartesian actual or setpoint position of the robot flange as well as every 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.

The result of the polling, i.e. the Cartesian position, refers by default to the world coordinate system. Optionally, it is possible to specify another reference coordinate system relative to which the Cartesian position is polled. This can for example be a frame created in the application data or a calibrated base.

The result of the polling 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 polled via the type-specific get methods.

Syntax

To poll the Cartesian actual position:

```
Frame position = robot.getCurrentCartesianPosition(
frameOnFlange<, referenceFrame>);
```

To poll 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 polled Cartesian position.
<i>robot</i>	Type: Robot Name of the robot from which the Cartesian position is polled

Element	Description
<i>frameOnFlange</i>	Type: ObjectFrame Robot flange or a frame subordinated to the flange whose Cartesian position is polled
<i>referenceFrame</i>	Type: AbstractFrame Reference coordinate system relative to which the Cartesian position is polled. 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 = lbr.getCurrentCartesianPosition(lbr.getFlange());
```

Cartesian actual position of the TCP of a tool with reference to a base:

```
tool.attachTo(lbr.getFlange());
...
Frame cmdPos = lbr.getCurrentCartesianPosition(tool.getFrame()/
TCP), getApplicationData().getFrame("/Base"));
```

15.13.3 Polling the Cartesian setpoint/actual value difference**Description**

The Cartesian setpoint/actual value difference (= difference between the programmed and measured position) can be polled with the `getPositionInformation(...)` method.

The result of the polling is saved in a variable of type `PositionInformation`. From this variable, the translational and rotational setpoint/actual value differences can be polled separately from each other.

Syntax

To poll position information:

```
PositionInformation info = robot.getPositionInformation(
frameOnFlange<, referenceFrame>);
```

To poll the translational setpoint/actual value difference:

```
Vector translatoryDiff = info.getTranslationOffset();
```

To poll 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 polled position information.
<i>robot</i>	Type: <code>Robot</code> Name of the robot from which the position information is polled
<i>frameOnFlange</i>	Type: <code>ObjectFrame</code> Robot flange or a frame subordinated to the flange whose position information is being polled

Element	Description
<i>referenceFrame</i>	Type: AbstractFrame Reference coordinate system relative to which the position information is polled. If no reference coordinate system is specified, the position information refers to the world coordinate system.
<i>translatoryDiff</i>	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Translational setpoint/actual value difference in the X, Y, Z directions (type: double, unit: mm) The offset values for each degree of freedom can be polled individually with the "get" methods of the Vector class. (>>> 15.12.4 "Polling individual values of a vector" Page 233)
<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 polled 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(lbr.getFlange());
...
PositionInformation posInf =
lbr.getPositionInformation(tool.getFrame("/TCP"),
getApplicationData().getFrame("/Base"));

Vector transDiff = posInf.getTranslationOffset();
Rotation rotDiff = posInf.getRotationOffset();

double transOffsetInX = transDiff.getX();
double rotOffsetofC = rotDiff.getGammaRad();
```

15.14 HOME position

The HOME position is an application-specific position of the robot. It can be reset for an application during initialization.

The HOME position has the following values by default:

Axis	A1	A2	A3	A4	A5	A6	A7
Pos.	0°	0°	0°	0°	0°	0°	0°

15.14.1 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 Option 1: transfer the axis position of the robot in the new HOME position. Type: AbstractFrame Option 2: 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 for example the case with a taught frame.

Examples

To transfer an axis-specific position as the HOME position:

```
private LBR lbr;  
...  
JointPosition newHome = new JointPosition(0.0, 0.0, 0.0,  
Math.toRadians(90), 0.0, 0.0, 0.0);  
lbr.setHomePosition(newHome);
```

To transfer the taught frame as the HOME position and move to it with `ptpHome()`:

```
private LBR lbr;  
...  
ObjectFrame newHome = getApplicationData().getFrame("/Homepos");  
lbr.setHomePosition(newHome);  
lbr.moveAsync(ptpHome());
```

15.15 Polling system states

Different system states can be polled from the robot and processed in the application. The polling of system states is primarily required when using a higher-level controller so that the controller can react to changes in state.

15.15.1 Polling the HOME position

Description

The following methods of the Robot class are available for polling the HOME position:

- `getHomePosition()`
Polls for the HOME position currently defined for the robot
- `isInHome()`
Polls whether the robot is currently in the HOME position

Syntax

To poll 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 <code>getHomePosition()</code> . The return value contains axis angles of the polled HOME position.
<i>robot</i>	Type: Robot Name of the robot from which the HOME position is polled
<i>result</i>	Type: Boolean Variable for the return value of <code>isInHome()</code> . 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.

```
private LBR lbr;
...
while(! lbr.isInHome ()) {
    //do something
}
```

15.15.2 Polling the mastering state

Description

The method `isMastered()` is available for polling 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 polled
<i>result</i>	Type: Boolean Variable for the return value. The return value is TRUE if all axes are mastered. If one or more of the axes are unmastered, the return value is false.

15.15.3 Polling “ready for motion”

Description

The method `isReadyToMove()` is available for polling whether the robot is ready for motion. The method belongs to the `Robot` class.

The “ready for motion” signal is withdrawn if a protective stop is active or if the robot drives are in the error state.

Syntax

```
boolean result = robot.isReadyToMove();
```

Explanation of the syntax

Element	Description
<i>robot</i>	Type: Robot Name of the robot which is polled whether it is ready for motion
<i>result</i>	Type: Boolean Variable for the return value. The return value is TRUE if the robot is ready for action.

15.15.3.1 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
<code>kuka_Sunrise_Cabinet</code>	Type: Controller Controller attribute of the robot application (= name of the robot controller in the application)

15.15.4 Polling activity

Description

A robot is active if a motion command is active. This affects both motion commands from the application and jogging commands.

The method hasActiveMotionCommand() is available for polling whether the robot is active. The method belongs to the Robot class.

Syntax

```
boolean result = robot.hasActiveMotionCommand();
```

Explanation of the syntax

Element	Description
<code>robot</code>	Type: Robot Name of the robot whose activity is polled
<code>result</code>	Type: Boolean Variable for the return value. The return value is TRUE if a motion command is active.



If hasActiveMotionCommand() returns the value FALSE (robot is not active), this does not necessarily mean that the robot must be stationary. This is the case while the robot is braking after the E-STOP has been pressed, for example.

An active robot can also be stationary (hasActiveMotionCommand() returns true), for example if a positionHold(...) command is executed in position control.

15.15.5 Polling and evaluating safety signals

Overview In a robot application, the state of the following safety signals can be polled and evaluated.

- Operating mode
- Local E-STOP
- External E-STOP
- Operator safety
- Stop request (safety stop)

Precondition To evaluate an E-STOP or operator safety:

- The appropriate category is selected in the PSM row in the safety configuration.
 - **Local E-STOP** category for a local E-STOP
 - **External E-STOP** category for an external E-STOP
 - **Operator safety** category for operator safety
- The configured reaction is a safety stop (no output).

15.15.5.1 Polling the state of the safety signals

Description The current state of the different safety signals is first polled via the method `getSafetyState()` and grouped in an object of type `ISafetyState`.

From this object, the state of individual safety signals can then be polled via specific methods. The possible safety states are Enums of the respective return value type.

Syntax

```
ISafetyState currentState = robot.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 polling.
<code>robot</code>	Type: <code>LBR</code> Name of the robot from which the state of the safety signals is polled

Overview The following methods are available for polling the safety signals of `ISafetyState` individually:

Method	Description
getOperationMode()	<p>Return value type: OperationMode (com.kuka.roboticsAPI.deviceModel package)</p> <p>Poll of the currently set operating mode</p> <p>Enum values of the return value type:</p> <ul style="list-style-type: none"> ■ T1, T2, AUT ■ CRR: CRR mode
getEmergencyStopInt()	<p>Return value type: EmergencyStop</p> <p>Poll of whether a local E-STOP is activated</p> <p>Enum values of the return value type:</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: EmergencyStop</p> <p>Poll of whether an external E-STOP is activated</p> <p>Enum values of the return value type:</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.
getOperatorSafetyState()	<p>Return value type: OperatorSafety</p> <p>Poll of the operator safety signal</p> <p>Enum values of the return value type:</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: SafetyStopType</p> <p>Poll of whether a safety stop is activated</p> <p>Enum values of the return value type:</p> <ul style="list-style-type: none"> ■ NOSTOP: No safety stop is activated. ■ STOP0: A safety stop 0 is activated. ■ STOP1: A safety stop 1 is activated.

Example

The system polls 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.

```

ISafetyState safetyState = robot.getSafetyState();

SafetyStopType safetyStop = safetyState.getSafetyStopSignal();

if(safetyStop != SafetyStopType.NOSTOP) {
    OperatorSafety operatorSafety =
    safetyState.getOperatorSafetyState();
    if (operatorSafety == OperatorSafety.OPERATOR_SAFETY_OPEN)
        getLogger().warn("The safety gate is open!");
}

```

15.15.5.2 Reacting to a change in state of safety signals

- Description** There is a notification service of the Controller class in the RoboticsAPI which reports changes in the state of safety signals. This service allows the user to directly react 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.

```
kuka_Sunrise_Cabinet.addControllerListener(new
ISunriseControllerStateListener() {
...
@Override
public void onSafetyStateChanged(Device device,
SunriseSafetyState safetyState) {
    OperatorSafety operatorSafety =
    safetyState.getOperatorSafetyState();
    if (operatorSafety == OperatorSafety.OPERATOR_SAFETY_OPEN)
        getLogger().warn("The safety gate is open!");
}
});
```

15.16 Changing and polling the program run mode

- Description** The program run mode can be changed and polled via the methods setExecutionMode(...) and getExecutionMode() of the SunriseExecutionService. The SunriseExecutionService itself is polled by the Controller.

- Preparation**
1. Variable of type SunriseExecutionService.
 2. Poll the SunriseExecutionService via the method getExecutionService() and save in the variable.

Syntax To change the program run mode:

```
service.setExecutionMode(ExecutionMode.newMode);
```

To poll the current program run mode:

```
currentMode = service.getExecutionMode();
```

Explanation of the syntax

Element	Description
service:	Type: SunriseExecutionService Variable for the return value (contains the SunriseExecutionService polled by the Controller)
newMode	Type: Enum of type ExecutionMode 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)
currentMode	Type: ExecutionMode Variable for the return value (contains the program run mode polled by the SunriseExecutionService)

Example

The SunriseExecutionService is polled by the Controller and saved in the variable "serv".

```
private SunriseExecutionService serv;
...
public void initialize() {
    controller = getController("Controller_LBR5");
    robot = (LBR) getRobot(controller, "LBR5_7kg");
    serv = (SunriseExecutionService) controller.getExecutionService();
    ...
}
```

The system first switches to Step mode and then back to standard mode.

```
public void run() {
    ...
    serv.setExecutionMode(ExecutionMode.Step);
    ...
    serv.setExecutionMode(ExecutionMode.Continuous);
    ...
}
```

The current program run mode is polled.

```
public void run() {
    ...
    ExecutionMode currentMode;
    currentMode = serv.getExecutionMode();
    ...
}
```

15.17 Changing and polling the program override

Description

The program override currently set can be polled in the application with `getProgramOverride()` and reduced with `clipProgramOverride(...)`.

It is not possible to increase the program override via the application. Even a program override reduced once with `clipProgramOverride(...)` cannot be increased again. If the current set value for the override is lower than the value programmed with `clipProgramOverride(...)`, the statement `clipProgramOverride(...)` is ignored.

The methods belong to the Controller class.

Syntax

To change the program override:

```
kuka_Sunrise_Cabinet.clipProgramOverride(newPOV);  
To poll the current program override:  
currentPOV = kuka_Sunrise_Cabinet.getProgramOverride();
```

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)
<i>newPOV</i>	Type: double New program override ■ 0.0 ... 1.0: 0% ... 100%
<i>currentPOV</i>	Type: double Variable for the return value (contains the polled program override)

Example

A program override of more than 80% is to be reduced to 50%.

```
private Controller kuka_Sunrise_Cabinet;  
...  
double currentPOV = kuka_Sunrise_Cabinet.getProgramOverride();  
  
if(currentPOV > 0.8){  
    kuka_Sunrise_Cabinet.clipProgramOverride(0.5);  
}
```

15.18 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 are, for example, 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.

15.18.1 Conditions in the RoboticsAPI

Description

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.condition-Model package and implement the ICondition interface.

Overview

The following condition types are available:

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. (>>> 15.18.3 "Axis torque condition" Page 247)
ForceCondition	The force condition is met if the force measured by the robot system exceeds a defined quantity. (>>> 15.18.4 "Force condition" Page 248)

Data type	Description
ForceComponentCondition	The force component condition is met if the force measured by the robot system lies outside of a defined range. (>>> 15.18.5 "Force component condition" Page 251)
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. (>>> 15.18.6 "Path-related condition" Page 253)
BooleanIOPCondition	The condition for Boolean signals is met if a Boolean digital input has a specific state. (>>> 15.18.7 "Condition for Boolean signals" Page 255)
IORangeCondition	The condition for the value range of a signal is met if the value of an analog or digital input lies within a defined range. (>>> 15.18.8 "Condition for the range of values of a signal" Page 255)

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.
(>>> 15.19 "Break conditions for motion commands" Page 256)
- 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.
(>>> 15.20 "Path-related switching actions (Trigger)" Page 258)
- Monitoring of processes (Monitoring)
The state of a condition is checked cyclically using a listener. If the state of the condition changes, it is possible to react.
(>>> 15.21 "Monitoring processes (Monitoring)" Page 262)
- Blocking wait for condition
An application is stopped until a certain condition is met or a certain wait time has expired.
(>>> 15.22 "Blocking wait for condition" Page 266)

15.18.2 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> <i>other:</i> further condition
AND	AND operation linking the calling ICondition object with one or more additional conditions <code>ICondition and(ICondition other1, ICondition other2, ...);</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 other1, ICondition other2, ...);</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));

```

15.18.3 Axis torque condition**Description**

The axis torque condition is used to check whether the torque measured in an axis lies outside of a defined range of values.

Constructor syntax

```
JointTorqueCondition(JointEnum joint, double minTorque,
double maxTorque)
```

Element	Description
<i>joint</i>	Axis whose torque value is to be checked ■ JointEnum.J1 ... JointEnum.J12: Axes A1 ... A12
<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

```
JointTorqueCondition torqueCondJ3 =
new JointTorqueCondition(JointEnum.J3, -2.5, 4.0);
```

The condition is met if a torque value of ≤ -2.5 Nm or ≥ 4.0 Nm is measured in axis A3.

15.18.4 Force condition

Description

The force condition is used to check whether a force measured by the robot system exceeds a defined limit value. The location of the force measurement is defined by a frame.

For example, the force exerted by a tool mounted on the robot flange can be checked. The force vector F in the tool direction is not taken into account here.

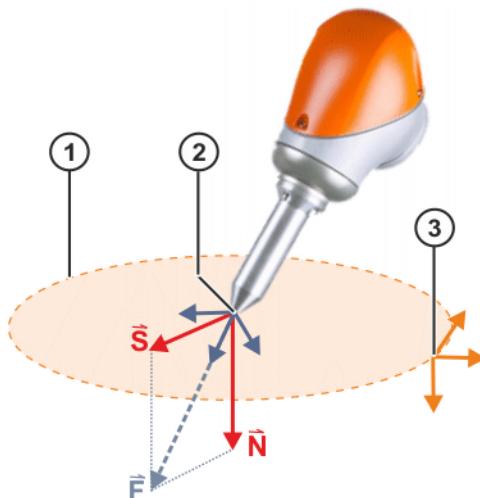


Fig. 15-10: Force vectors

- 1 Surface on which the force is exerted
- 2 Measurement point, here the tool frame
- 3 Frame for determining the orientation of the surface. The position of the frame on the surface is irrelevant.

The following force vectors are relevant:

- Normal force N :
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.
- Shearing force S :
Projection of the force exerted on the surface. This results in the part of the force exerted parallel to the surface. The shearing force is generated by friction.

Methods

Force conditions are of the data type ForceCondition. ForceCondition contains the following static methods for programming force conditions:

- `createSpatialForceCondition(...)`: Force condition for spatial force from all directions
- `createNormalForceCondition(...)`: Force condition for normal force
- `createShearForceCondition(...)`: Force condition for shearing force

To formulate the condition, a frame is defined as a measurement point relative to the flange coordinate system. The force is measured at this frame, e.g. the tip of a tool. The orientation of this measurement point 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 force threshold is defined to determine the minimum amount of force which meets the condition.

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 condition is also met.

15.18.4.1 Force condition for spatial force from all directions

Description Via the static method `createSpatialForceCondition(...)`, a force condition can be defined for the spatial force from all directions acting on the measurement point.

Syntax

```
ForceCondition.createSpatialForceCondition(AbstractFrame measureFrame, AbstractFrame orientationFrame, double threshold, double tolerance)
```

Element	Description
<i>measureFrame</i>	Frame relative to the robot flange, on which the force is measured, e.g. the TCP of a tool. The position of the measurement point is defined using this parameter.
<i>orientationFrame</i>	Optional. The orientation of the measurement point is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, the <i>measureFrame</i> defines the orientation of the measurement point.
<i>threshold</i>	Maximum magnitude of force which may act on the measurement point (unit: N). The condition is met if the magnitude of force specified here is exceeded.
<i>tolerance</i>	Optional. Threshold for the maximum permissible inaccuracy of the calculated values. The condition is met if the value specified here is exceeded.

Example A force condition is to be formulated. 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.

```
Tool gripper = ...;
gripper.attachTo(lbr.getFlange());

ForceCondition spatialForce_tcp =
ForceCondition.createSpatialForceCondition(
    gripper.getFrame("/TCP"),
    30.0,
);
```

15.18.4.2 Force condition for normal force

Description A force condition for the normal force can be formulated via the static method `createNormalForceCondition(...)`. The component of the force exerted on the

measurement point and acting along a definable axis 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)
```

Element	Description
<i>measure Frame</i>	Frame relative to the robot flange, on which the force is measured, e.g. the TCP of a tool. The position of the measurement point is defined using this parameter.
<i>orientation Frame</i>	Optional. The orientation of the measurement point is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, the <i>measureFrame</i> defines the orientation of the measurement point.
<i>direction</i>	The force component acting on the measurement point along the axis specified here is checked. ■ X, Y, Z
<i>threshold</i>	Maximum magnitude of force which may act on the measurement point (unit: N). The condition is met if the magnitude of force specified here is exceeded.
<i>tolerance</i>	Optional. Threshold for the maximum permissible inaccuracy of the calculated values. The condition is met if the value specified here is exceeded.

Example

A force condition is to be formulated for the following situation. A gripper mounted on the flange presses on a table plate. The force at the TCP of the gripper is measured using the orientation of the table plate. In the process, the force which acts along the Z axis of this defined measurement point is examined.

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.

```
Tool gripper = ...;
gripper.attachTo(lbr.getFlange());

SpatialObject table = ...;

ForceCondition normalForce_z =
ForceCondition.createNormalForceCondition(
    gripper.getFrame("/TCP"),
    table.getFrame("/Tabletop"),
    CoordinateAxis.Z,
    45.0,
    8.0

);
```

15.18.4.3 Force condition for shearing force

Description

A force condition for the shearing force can be formulated via the static method `createShearForceCondition(...)`. The component of the force exerted on the measurement point and acting parallel to a surface, is considered here. This surface is generally defined by specifying the axis which is perpendicular to the surface on which the force is exerted (surface normal).

Syntax

```
ForceCondition.createShearForceCondition(AbstractFrame
    measureFrame, AbstractFrame orientationFrame, CoordinateAxis normalDirection, double threshold, double tolerance)
```

Element	Description
<i>measureFrame</i>	Frame relative to the robot flange on which the force is measured, e.g. the TCP of a tool. The position of the measurement point is defined using this parameter.
<i>orientationFrame</i>	Optional. The orientation of the measurement point is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, the <i>measureFrame</i> defines the orientation of the measurement point.
<i>normalDirection</i>	Axis of the measurement point which is perpendicular to the surface on which the force is exerted. ■ X, Y, Z
<i>threshold</i>	Maximum magnitude of force which may act on the measurement point (unit: N). The condition is met if the magnitude of force specified here is exceeded.
<i>tolerance</i>	Optional. Threshold for the maximum permissible inaccuracy of the calculated values. The condition is met if the value specified here is exceeded.

Example

A force condition is to be formulated for the following process. A gripper mounted on the flange presses on a table plate. The force at the TCP of the gripper is measured using the orientation of the table plate. This process considers the shearing 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 shearing 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.

```
Tool gripper = ...;
gripper.attachTo(lbr.getFlange());

SpatialObject table = ...;

ForceCondition shearForce_xyPlane =
ForceCondition.createShearForceCondition(
    gripper.getFrame("/TCP"),
    table.getFrame("/Tabletop"),
    CoordinateAxis.Z,
    25.0,
    5.0

);
```

15.18.5 Force component condition**Description**

The force component condition is used to check whether a force measured by the robot system lies outside of a defined range in the X, Y or Z direction. The location of the force measurement is defined by a frame.

Force component conditions are of data type ForceComponentCondition. To formulate the condition, a frame is defined as a measurement point relative to the flange coordinate system. The force is measured on this frame, e.g. the tip

of a tool. The orientation of this measurement point 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 measurement point.

The Cartesian force is calculated from the values of the joint torque sensor. 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 the following constructors:

```
ForceComponentCondition(AbstractFrame measureFrame,  
CoordinateAxis coordinateAxis, double min, double max)
```

```
ForceComponentCondition(AbstractFrame measureFrame, Abs-  
tractFrame orientationFrame, CoordinateAxis coordinateAxis, double  
min, double max)
```

```
ForceComponentCondition(AbstractFrame measureFrame,  
CoordinateAxis coordinateAxis, double min, double max, double  
tolerance)
```

```
ForceComponentCondition(AbstractFrame measureFrame, Abs-  
tractFrame orientationFrame, CoordinateAxis coordinateAxis, double  
min, double max, double tolerance)
```

Element	Description
<i>measureFrame</i>	Frame below the robot flange, on which the force is measured, e.g. the TCP of a tool. The position of the measurement point is defined using this parameter.
<i>orientationFrame</i>	Optional. The orientation of the measurement point is defined using this parameter. If the <i>orientationFrame</i> parameter is not specified, the <i>measureFrame</i> defines the orientation of the measurement point.
<i>coordinateAxis</i>	Coordinate axis of the measurement point. Defines the direction from which the acting force is checked.
<i>min</i>	Lower limit of the range of values for the force exerted on the measurement point (unit: N). The condition is met if the acting force falls below this value.
<i>max</i>	Upper limit of the range of values for the force exerted on the measurement point (unit: N). The condition is met if the acting force exceeds this value. Note: The upper limit value must be greater than the lower limit value: <i>max > min</i> .
<i>tolerance</i>	Optional. Threshold for the maximum permissible inaccuracy of the calculated values. The condition is met if the value specified here is exceeded.

Example

A joining process is ideally executed with a force of between 20 N and 25 N. A force component condition is to be formulated, and is met if the force, which acts on 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.

```

Workpiece peg = ...;
peg.attachTo(...);

ForceComponentCondition assemblyForce_inverted = new
ForceComponentCondition(
    peg.getFrame("/Assembly"),
    CoordinateAxis.Z,
    20.0,
    25.0
);

ForceComponentCondition assemblyForce = (ForceComponentCondition)
assemblyForce_inverted.invert();

```

15.18.6 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 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.

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)
```

Element	Description
<i>reference</i>	Data type: com.kuka.roboticsAPI.conditionModel.ReferenceType Reference point of the condition <ul style="list-style-type: none"> ■ ReferenceType.START: Start point ■ ReferenceType.DEST: End point

Element	Description
<i>distance</i>	Offset in space relative to the reference point of the condition. Unit for CP motions: mm (the path parameter is specified for PTP motions). <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 254)</p>
<i>delay</i>	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) <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 254)</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 <i>distance</i> < 0	The switching point is before the start of the motion.
reference = ReferenceType.START <i>distance</i> = 0 <i>delay</i> < 0	
reference = ReferenceType.DEST <i>distance</i> > 0	The switching point is after the end of the motion.
reference = ReferenceType.DEST <i>distance</i> = 0 <i>delay</i> > 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);
```

15.18.7 Condition for Boolean signals

Description	The Boolean signal condition can be used to check digital 1-bit inputs. The condition is met if a Boolean input has a specific state. Boolean signal conditions are of data type BooleanIOPCondition.						
Constructor syntax	BooleanIOPCondition(AbstractIO <i>booleanSignal</i> , boolean <i>booleanIO-Value</i>) <table border="1"> <thead> <tr> <th>Element</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>boolean Signal</i></td><td>Boolean input signal that is checked</td></tr> <tr> <td><i>boolean IOValue</i></td><td>State of the input signal with which the condition is met ■ true, false</td></tr> </tbody> </table>	Element	Description	<i>boolean Signal</i>	Boolean input signal that is checked	<i>boolean IOValue</i>	State of the input signal with which the condition is met ■ true, false
Element	Description						
<i>boolean Signal</i>	Boolean input signal that is checked						
<i>boolean IOValue</i>	State of the input signal with which the condition is met ■ 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.
----------------	---

```
SwitchesIOPGroup switches = new SwitchesIOPGroup(...);
AbstractIO switch_1 = switches.getInput("Switch1");

BooleanIOPCondition switch1_active = new BooleanIOPCondition(switch_1,
true);
```

15.18.8 Condition for the range of values of a signal

Description	The value of a digital or analog input 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 <i>signal</i> , Number <i>minValue</i> , Number <i>maxValue</i>) <table border="1"> <thead> <tr> <th>Element</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>signal</i></td><td>Analog or digital signal that is checked</td></tr> <tr> <td><i>minValue</i></td><td>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>.</td></tr> <tr> <td><i>maxValue</i></td><td>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>.</td></tr> </tbody> </table>	Element	Description	<i>signal</i>	Analog or digital 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> .
Element	Description								
<i>signal</i>	Analog or digital 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.
----------------	---

```
SensorIOPGroup sensors = new SensorIOPGroup(...);
AbstractIO temperatureSensor =
sensors.getInput("TemperatureSensor2");

IORangeCondition tempHigher35 = new
IORangeCondition(temperatureSensor, 35.0, 2000.0);
```

15.19 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.

15.19.1 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:

(>>> 15.18.1 "Conditions in the RoboticsAPI" Page 245)

To define a break condition for a motion, an object of the desired condition type is transferred to the motion command via the motion 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.

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: ■ ptp(getApplicationContext().getFrame("/P1"))
<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(lin(getApplicationContext().getFrame("/P10"))
.breakWhen(cond_1));
```

15.19.2 Evaluating the break conditions

Description

If break conditions have been defined for a motion command, these can be evaluated:

- Evaluation of the condition which caused the motion to be terminated.
- Evaluation of the position of the robot at that time.

For this purpose, the motion command is temporarily stored in an IMotionContainer variable. Via the method getFiredBreakConditionInfo(), this variable can be polled 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();
```



It is currently not possible to evaluate a break condition which has been triggered by an individual motion within a MotionBatch.

Explanation of the syntax

Element	Description
<i>motion</i>	Motion instruction Example: <ul style="list-style-type: none"> ■ lbr.move(ptpgetApplicationData().getFrame("/P1"))
<i>motionCmd</i>	Type: IMotionContainer Temporary memory for the motion command
<i>firedCondInfo</i>	Type: IFiredConditionInfo Information about termination of the motion

15.19.2.1 Polling for the triggered break conditions

Description

The triggered break condition is polled via the method getFiredCondition(). The returned value is of type ICondition and can be compared to the transferred break conditions via the equals(...) method.

The poll 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 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 =  
lbr.move(lin(getApplicationData().getFrame("P10")) .breakWhen(cond1) .  
breakWhen(cond2));
```

The IFiredConditionInfo is polled via "motionCmd". If the polled information is not equal to zero, the motion has been terminated. The system only polls for the triggered break condition in this case.

```
IFiredConditionInfo firedInfo = motionCmd.getFiredConditionInfo();
if(firedInfo != null){

    ICondition firedCond = firedInfo.getFiredCondition();
    if(firedCond.equals(cond1)){
        ...
    }
    ...
}
```

15.19.2.2 Polling for the robot position at the time of termination

Description The robot position at the time when the break condition was triggered is polled via the IFiredConditionInfo 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
firedPosInfo	Type: PositionInformation Variable for the return value. The return value contains the position information at the time when the break condition was triggered.
firedCondInfo	Type: IFiredConditionInfo Information about termination of the motion

Example

The Cartesian actual position of the robot when the break condition is triggered is polled via the method getCurrentCartesianPosition().

```
PositionInformation firedPosInfo = firedInfo.getPositionInfo();
Frame firedCurrPos = firedPosInfo.getCurrentCartesianPosition();
```

15.20 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 a setdown position for the workpiece it is transporting is free.

15.20.1 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:
 (>>> 15.18.1 "Conditions in the RoboticsAPI" Page 245)

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 motion 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(...).

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: ■ ptpgetApplicationData().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 (>>> 15.20.2 "Programming a path-related switching action" Page 259)

15.20.2 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 package com.kuka.roboticsAPI.conditionModel. This interface currently does not offer 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.

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(...)
onTrigger Fired(...)	Method whose execution is fired by the trigger
triggerInformation	Type: IFiredTriggerInfo Contains information about the firing trigger (>>> 15.20.3 "Evaluating trigger information" Page 260)

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
BooleanIOCCondition buttonPressed = new
BooleanIOCCondition(IOs.getInput("DI1"), true);

//motion with trigger
robot.move(P1).triggerWhen(buttonPressed, toggleOut_1));
robot.move(P2));
```

15.20.3 Evaluating trigger information

The method `onTriggerFired(...)` 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 method `onTriggerFired(...)`. This trigger information can be polled.

Overview

The following methods of the `IFiredTriggerInfo` class are available:

Method	Description
<code>getFiredCondition()</code>	Return value type: <code>ICondition</code> Polls for the condition which fired the trigger
<code>getMissedEvents()</code>	Return value type: <code>int</code> Polls for how many times the event which fired the trigger still occurred while the triggered action was being executed Note: The event which triggered the action cannot be fired again while the action is being executed.

Method	Description
getMotionContainer()	Return value type: IMotionContainer Polls for the motion command, during the execution of which the trigger was fired
getPositionInformation()	Return value type: PositionInformation Polls for 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 Polls for the time at which the trigger was fired

To poll for the position information obtained with getPositionInformation(), the following methods of the PositionInformation class are available:

Method	Description
getCommandedCartesianPosition()	Return value type: Frame Polls for the Cartesian setpoint position at triggering time
getCommandedJointPosition()	Return value type: JointPosition Polls for the axis-specific setpoint position at triggering time
getCurrentCartesianPosition()	Return value type: Frame Polls for the Cartesian actual position at triggering time
getCurrentJointPosition()	Return value type: JointPosition Polls for 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.

```

BooleanIOCondition in1 = new BooleanIOCondition(_input_1, true);

ICallbackAction ica = new ICallbackAction() {

    @Override
    public void onTriggerFired(IFiredTriggerInfo triggerInformation) {
        getLogger().info("TriggerTime: " +
triggerInformation.getTriggerTime().toString());
        getLogger().info("TriggerCondition: " +
triggerInformation.getFiredCondition().toString());
    }
};

robot.move(P1).triggerWhen(in1, ica));
  
```

Example 2 The axis-specific and Cartesian robot position at triggering time are polled.

```

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(P1).triggerWhen(in1, ica));

```

15.21 Monitoring processes (Monitoring)

Monitoring means keeping a process under surveillance 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.

15.21.1 Listener for monitoring conditions

Various listener interfaces are available from the package com.kuka.roboticsAPI.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, change in state FALSE > TRUE).</p> <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, change in state TRUE > FALSE).</p> <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, change in state FALSE > TRUE or TRUE > FALSE).</p> <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	Create a listener object to monitor the condition. (>>> 15.21.2 "Creating a listener object to monitor the condition" Page 263)
2	Program the the desired handling routine in the listener method.
3	Register the listener for notification in case of a change in state of the condition. (>>> 15.21.3 "Registering a listener for notification of change in state" Page 264)
4	If this has not already been done by the method selected for registration, activate the notification service for the listener. (>>> 15.21.4 "Activating or deactivating the notification service for listeners" Page 265)

15.21.2 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(...), which is automatically declared when the object is created, has input parameters. These input parameters contain information about the event triggered by the execution of the method, and can be polled 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
Input parameters of the listener method:	
condition Observer	Type: ConditionObserver Object notified by the listener
time	Type: Date Date and time the listener was notified

Element	Description
missed Events	<p>Type: int</p> <p>Number of missed changes in state since the method was last executed</p> <p>Possible causes of missed events:</p> <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.
condition Value	<p>Type: Boolean</p> <p>Only present with the listener method onAnyEdge(...). Specifies the edge via which the method was called.</p> <ul style="list-style-type: none"> ■ true: rising edge (change in state FALSE > TRUE) ■ false: falling edge (change in state TRUE > FALSE)

15.21.3 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 polled via the method getObserverManager(). The ObserverManager class provides various methods for creating the required object. <ul style="list-style-type: none"> ■ 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. (>>> 15.21.4 "Activating or deactivating the notification service for listeners" Page 265)
Syntax	<pre>ConditionObserver myObserver = getObserverManager().createAndEnableConditionObserver (condition, notificationType, listener)</pre>

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. (>>> "NotificationType" Page 265)
<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	<p>The listener is notified in the event of an edge change (according to the listener type used).</p> <p>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:</p> <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
MissedEvents	<p>The listener is notified in the event of an edge change (according to the listener type used).</p> <p>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.</p>
All	<p>Combination of OnEnable and MissedEvents</p> <p>The listener is notified in the case of all events described under OnEnable and MissedEvents.</p>

15.21.4 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

15.21.5 Programming example for monitoring

A 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.

The notification type NotificationType.MissedEvents is selected in order to notify the listener of missed exceeded forces as quickly as possible.

```

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();

```

15.22 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.

`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:
[\(>>> 15.18.1 "Conditions in the RoboticsAPI" Page 245\)](#)

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 given wait time. The <code>Enum</code> is contained by default in the Java libraries.
<i>result</i>	Type: <code>Boolean</code> 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.

```
SwitchIOGroup inputs = new SwitchIOGroup(kuka_Sunrise_Cabinet);
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
}
```

15.23 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.

15.23.1 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: DataRecorder*ID*.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
<i>fileName</i>	File name (with extension) under which the recorded data are saved Example: "Recording_1.log"
<i>timeout</i>	Recording duration <ul style="list-style-type: none"> ■ -1: No recording duration is defined. ■ ≥ 1 Default: -1 The time unit is defined with <i>timeUnit</i> .
<i>timeUnit</i>	Time unit for the recording duration Example: TimeUnit.SECONDS The Enum of type TimeUnit is contained by default in the Java libraries.
<i>sampleRate</i>	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");
```

15.23.2 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
<code>addInternalJointTorque(...)</code>	Return value type: DataRecorder Recording of the measured axis torques of the robot which is transferred as a parameter (type: robot)
<code>addExternalJointTorque(...)</code>	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)

Method	Description
addCartesianForce(...)	<p>Return value type: DataRecorder</p> <p>Recording of the Cartesian forces along the X, Y and Z axes of the frame which is transferred as a parameter (unit: N). The variance of the Cartesian forces is also recorded.</p> <p>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, <code>null</code> must be transferred.</p>
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). The variance of the Cartesian forces is also recorded.</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 connected 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: robot). As a second parameter, the unit in which the axis angles are recorded must be transferred (Enum of type: AngleUnit).</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: robot) As a second parameter, the unit in which the axis angles are recorded must be transferred (Enum of type: AngleUnit).</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"> ■ <code>AbstractFrame measureFrame</code> Frame connected to the robot flange, e.g. the TCP of a tool. Defines the position of the measurement point. ■ <code>AbstractFrame referenceFrame</code> Defines the reference coordinate system. <p>Note: Both parameters must always be transferred together. None of the parameters may be <code>null</code>.</p>

Example

For an LBR iiwa, 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

```
private LBR lbr_iiwa;
private Tool gripper;
...
gripper.attachTo(lbr_iiwa.getFlange());

DataRecorder rec = new DataRecorder();
rec.addInternalJointTorque(lbr_iiwa);
rec.addCartesianForce(gripper.getFrame("TCP"),
getApplicationData().getFrame("/Base"));
```

15.23.3 Starting data recording

Data recording can be started independent of robot motion (possible at any point in the application), or synchronous 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.

(>>> 15.20.1 "Programming triggers" Page 258)

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.

```
private LBR lbr_iawa;
...
DataRecorder rec = new DataRecorder();
...
rec.enable();
...
lbr_iawa.move(lingetApplicationData().getFrame("/Pre-position")));
rec.startRecording();
```

Example 2

Data recording is to begin 2 s after the start of a motion. A MotionPathCondition object is parameterized for this.

```
private LBR lbr_iawa;
...
DataRecorder rec = new DataRecorder();
...
StartRecordingAction startAct = new StartRecordingAction(rec);
MotionPathCondition startCond = new
MotionPathCondition(ReferenceType.START, 0.0, 2000);
lbr_iawa.move(lingetApplicationData().getFrame("/
Destination")).triggerWhen(startCond, startAct));
```

15.23.4 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.

(>>> 15.20.1 "Programming triggers" Page 258)

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(...).

15.23.5 Polling states from the DataRecorder object**Overview**

The following methods of the DataRecorder class are available:

Method	Description
isEnabled()	Return value type: Boolean The system polls whether the DataRecorder object is activated (= true).
isRecording()	Return value type: Boolean The system polls whether data recording is running (= true).
isFileAvailable()	Return value type: Boolean The system polls whether the file with the recorded data is already saved on the robot controller and whether it is available for evaluation (= true).
awaitFileAvailable(...)	Return value type: Boolean Blocks the calling application or background task 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). The blocking statement returns the value “false” if the file is not available within the maximum blocking duration. Syntax: <ul style="list-style-type: none"> ■ <code>awaitFileAvailable(long timeout, java.util.concurrent.TimeUnit timeUnit)</code> Parameters: <ul style="list-style-type: none"> ■ <i>timeout</i>: maximum blocking duration ■ <i>timeUnit</i>: time unit for the maximum blocking time

15.23.6 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 iiwa 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.

```

private LBR lbr_iwa;
private Tool gripper;
...
gripper.attachTo(lbr_iwa.getFlange());
...
DataRecorder rec = new DataRecorder();
rec.setFileName("Recording.log");
rec.setSampleRate(10);

rec.addExternalJointTorque(lbr_iwa);
rec.addCartesianForce(gripper.getFrame("/TCP"), null);

StartRecordingAction startAction = new StartRecordingAction(rec);
ForceCondition startCondition =
ForceCondition.createSpatialForceCondition(gripper.getFrame("/TCP"),
20.0);

lbr_iwa.move(ptp(getApplicationContext().getFrame("/StartPosition")));
lbr_iwa.move(lin(getApplicationContext().getFrame("/
MountingPosition")).triggerWhen(startCondition, startAction));
lbr_iwa.move(lin(getApplicationContext().getFrame("/DonePosition")));

rec.stopRecording();

if (rec.awaitFileEnable(5, TimeUnit.SECONDS)) {
// Evaluation of the file if available
}

```

15.24 Message programming

15.24.1 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.



It is advisable to only display messages on the smartHMI which are absolutely essential. A too intensive use of the message display 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 the standard Java commands. If the standard commands are used, it is not possible to guarantee that the message will be displayed on the smartHMI.

Syntax

Notification message:

```
getLogger().info ("Meldungstext");
```

Warning message:

```
getLogger().warn ("Message text");
```

Error message:

```
getLogger().error ("Message text");
```

Message that is only written to the log file:

```
getLogger().fine ("Message text");
```

Explanation of the syntax

Element	Description
Message text	Text 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.

```
IMotionContainer motion = lbr.move(lin(getFrame("/P20"))
    .breakWhen(collision));

if(motion.getFiredBreakConditionInfo() == null) {
    getLogger().info("End point reached.");
}
else{
    getLogger().warn("Motion canceled after collision!");
}
```

15.24.2 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.

Different 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 user must take note
	QUESTION Dialog with a question which the user must answer
	WARNING Dialog with a warning of which the user must take note
	ERROR Dialog with an error message of which the user must take note

The user answers by selecting a button that can be labeled by the programmer. Up to 12 buttons can be defined.

The application or the background task from which the dialog was called is stopped until the user reacts. How program execution continues can be made dependent on which button the user selects. The method `displayModalDialog(...)` returns the index of the button which the user 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
<code>Dialog_type</code>	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.

Element	Description
<i>Dialog_text</i>	Type: String Text which is displayed in the dialog window on the smartHMI
<i>Button_1 ... 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:

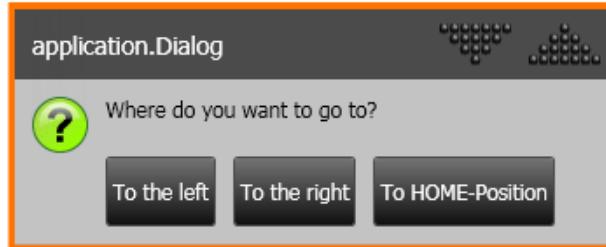


Fig. 15-11: Example of a user dialog

```
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:
        lbr.move(ptpgetApplicationData().getFrame("/Left"));
        break;
    case 1:
        lbr.move(ptpgetApplicationData().getFrame("/Right"));
        break;
    case 2:
        lbr.move(ptpHome());
        break;
}
```

15.25 Program execution control

15.25.1 Stopping an application

Description

An application can be stopped via the `halt()` method.

An application stopped in this way cannot be resumed in the program but only with the Start/pause key on the smartPAD. The next statement after `halt()` is then executed.

Syntax

```
getApplicationControl().halt();
```

15.25.2 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. ([>>> 15.25.6 "Examples of nested loops"](#)
Page 279)

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++) {
    getLogger().info(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.

15.25.3 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. ([>>> 15.25.6 "Examples of nested loops"](#)
Page 279)

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){
    getLogger().info("input 1 is TRUE.");
}
getLogger().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.

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 x1).

15.25.4 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 a condition is met, the loop is ended 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 e.g. because the break condition is being calculated or is a specific signal.

DO WHILE loops can be nested. ([>>> 15.25.6 "Examples of nested loops"](#)
Page 279)

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.

15.25.5 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
<i>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

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

```
for(int a = 1; a <= 10; a++) {
    if(a == 3){
        a = a + 5;
    }
    getLogger().info(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.

15.25.6 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++) {
    getLogger().info(i + ". Cycle begins");

    for (int k = 10; k > 0; k--) {
        getLogger().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 in WHILE
loop**

```
int sum = 0;
int round = 1;
int throw = 0;
Random num = new Random();

while (sum < 21) {
    round++;

    for (int i = 1; i <= 3; i++) {
        throw = (num.nextInt(6) + 1);
        if (throw % 2 == 0)
            sum += throw;
    }
}
```

The following rules apply in a dice game:

- The total sum of all rolls must be at least 21 (poll with WHILE loop).
- The dice are rolled 3 times in each round (FOR loop).
- Only even numbers (2, 4 and 6) are counted (IF poll with modulo).

16 Background tasks

16.1 Using background tasks

Tasks Background tasks are used in order to be able to perform actions in the background, parallel to a running robot application. Multiple background tasks can run in parallel and independently of one another.

Background tasks can be used for the following tasks:

- Controlling and monitoring peripheral devices. Examples: Monitoring of safety equipment; monitoring of a cooling circuit.
This means that no PLC is required for smaller applications, as the robot controller can perform such tasks by itself.
- Monitoring information about the robot and station



WARNING Background tasks must not be used for moving the robot. This is the task of the robot application. Calling motion commands from a background task can result in unspecified behavior of the robot and thus cause personal injury and damage to property.

Properties 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.

(>>> 5.5 "Creating a new background task" Page 50)

There are 2 types of background task that differ in terms of their duration:

- Cyclic background tasks
Executed cyclically. The cyclical behavior can be adapted by the programmer depending on the task to be performed.
- Non-cyclic background tasks
Executed once.

Background tasks of start type **Automatic** are started automatically after synchronization of a project and after the robot controller has booted. Background tasks of start type **Manual** must be started manually via the smartPAD.

16.2 Cyclic background task

Structure

```

(1) package backgroundTask;
(2) import java.util.concurrent.TimeUnit;
(3) public class BackgroundTaskCyclic extends RoboticsAPICyclicBackgroundTask {
    private Controller kuka_Sunrise_Cabinet_1;
(4)
(5)     public void initialize() {
        kuka_Sunrise_Cabinet_1 = getController("KUKA_Sunrise_Cabinet_1");
        initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
                        CycleBehavior.BestEffort);
    }
(6)     public void runCyclic() {
    }
}

```

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 When the task is created, one instance of the Controller class is automatically created. The data array can be used, for example, to access the available robots in order to poll the position and torque data.
5	initialize() method Here, the data arrays created in the declaration section are assigned initial values. When the task is created, the method initializeCyclic(...) is automatically called; this is used to define the cyclical behavior of the task. (>>> "Initialization" Page 282) 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 initialize the cyclical behavior of the background task. The parameters transferred to the method are preset with default values and can be modified by the programmer.

```
initializeCyclic(long initialDelay, long period, TimeUnit timeUnit,  
CycleBehavior behavior) ;
```

Element	Description
<i>initialDelay</i>	Delay after which the cyclic background task is started. Default: 0 ms The time unit is defined with <i>timeUnit</i> .
<i>period</i>	Period (= time between 2 calls of runCyclic()) Default: 500 ms The time unit is defined with <i>timeUnit</i> .

Element	Description
<i>timeUnit</i>	Time unit of <i>initialDelay</i> and <i>period</i> Default: TimeUnit.MILLISECONDS The Enum of type TimeUnit is contained by default in the Java libraries.
<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 runCyclic() is defined here. <ul style="list-style-type: none">■ CycleBehavior.BestEffort runCyclic() is executed completely and then called again.■ CycleBehavior.Strict Execution of the background task is canceled and a CycleExceededException is launched. Default: CycleBehavior.BestEffort

16.3 Non-cyclic background task

Structure

```

① package backgroundTask;

② import com.kuka.roboticsAPI.applicationModel.tasks.RoboticsAPIBackgroundTask;

③ public class BackgroundTask extends RoboticsAPIBackgroundTask {
    private Controller kuka_Sunrise_Cabinet_1;
}

⑤ public void initialize() {
    kuka_Sunrise_Cabinet_1 = getController("KUKA_Sunrise_Cabinet_1");
}

⑥ public void run() {
}

```

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 When the task is created, one instance of the Controller class is automatically created. The data array can be used, for example, to access the available robots in order to poll the position and torque data.

Item	Description
5	<p>initialize() method</p> <p>Here, the data arrays created in the declaration section are assigned initial values.</p> <p>Note: The method must not be deleted or renamed.</p>
6	<p>run() method</p> <p>The code that is to be executed once is programmed here. The runtime is not limited.</p> <p>Note: The method must not be deleted or renamed.</p>

17 Programming with a compliant robot

17.1 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 KUKA LBR iiwa 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 KUKA LBR iiwa 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.

17.2 Available controllers – overview

The KUKA LBR iiwa can be operated with a number of 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.

Slider	Description
Position controller	<p>Data type: PositionControlMode</p> <p>The aim of position control is to execute the specified path with the maximum possible positional accuracy and without path deviation. By default, external influences such as obstacles or process forces are not taken into account.</p>
Cartesian impedance controller	<p>Data type: CartesianImpedanceControlMode</p> <p>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.</p>
Cartesian impedance controller with overlaid force oscillation	<p>Data type: CartesianSinelImpedanceControlMode</p> <p>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 to implement force-dependent search runs and vibration motions for joining processes, for example.</p>

17.3 Using controllers in robot applications

- | | |
|--------------------|--|
| Description | In robot applications, the controller to be used is set separately for every motion command. The following steps are required by default for this: |
| Procedure | <ol style="list-style-type: none"> 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. |

17.3.1 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. By default, the controller object is generated using the standard constructor.

Syntax

```
ControllerType controlMode;
controlMode = new ControllerType();
```

Explanation of the syntax	Element	Description
	Controller-Type	Data type of the controller. Subclass of AbstractMotionControlMode.
	controlMode	Name of the controller object

Example Creating a Cartesian impedance controller:

```
CartesianImpedanceControlMode cartImpCtrlMode;
cartImpCtrlMode = new CartesianImpedanceControlMode();
```

17.3.2 Defining controller parameters

The parameters that can be set depend on the type of the controller used. The individual controller classes in the RoboticsAPI provide specific "set" and "get" methods for each parameter.

(>>> 17.5.2 "Parameterization of the impedance controller" Page 289)

(>>> 17.6.3 "Parameterization of the impedance controller with overlaid force oscillation" Page 297)

17.3.3 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.



If the robot is in a singularity position, the motion must not start under impedance control.

Syntax

```
movableObject.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 the controller object

17.4 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.

17.5 Cartesian impedance controller

The Cartesian impedance controller is represented by the class `Cartesian-ImpedanceControlMode`.

The impedance controller refers by default 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 tool coordinate system currently used for the gripper or 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.

17.5.1 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 :

$$F = C * \Delta x$$

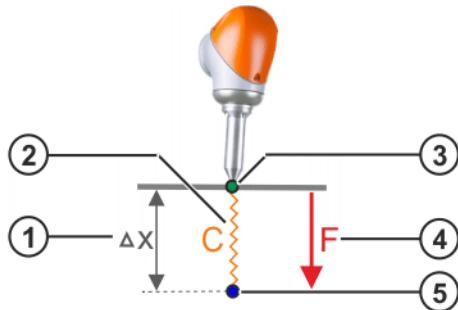


Fig. 17-1: Virtual spring with spring stiffness C

- | | |
|-------------------------|---------------------|
| 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 here too, 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. 17-2: Force exerted on contact

As shown in the figure ([>>> Fig. 17-2](#)), 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. 17-3: 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.

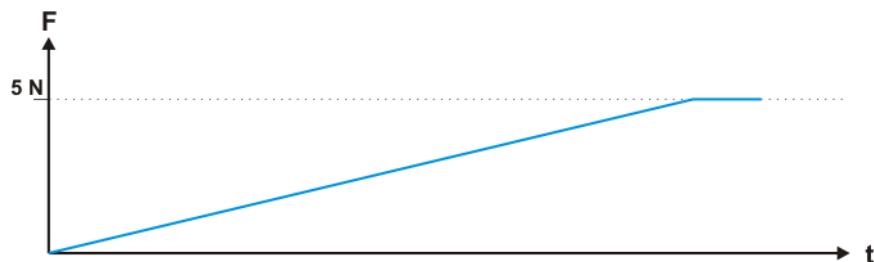


Fig. 17-4: Force over time (low stiffness, large position difference)

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 $F_x = C_x \Delta x$ in the x direction and $F_y = C_y \Delta y$ in the y direction. The vector addition results in the overall force F_{res} .

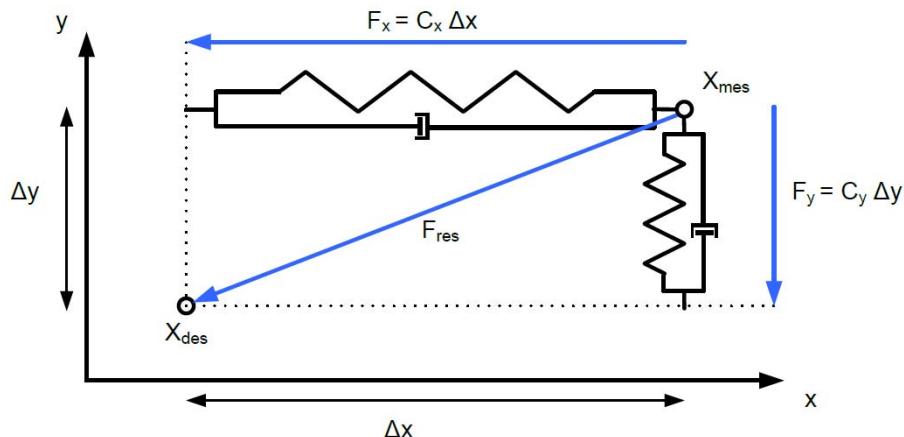


Fig. 17-5: Overall force in the case of deflection in 2 directions

17.5.2 Parameterization of the 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 rota-

tional 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.



WARNING In impedance control, 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, resulting in unpredictable motions of the robot.

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 at the TCP
- Maximum Cartesian velocity
- Maximum Cartesian path deviation

17.5.2.1 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.TRANSL	Combination of the translational degrees of freedom in the X, Y and Z directions
CartDOF.A	Rotational degree of freedom about the Z axis
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

17.5.2.2 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 degree of freedom for which the controller parameter is to apply is specified first. The parametrize(...) method of the controller data types is used for this purpose. To define the degree 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 point operator. This controller parameter is set to the value specified as the in-

put 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
<i>controlMode</i>	Type: CartesianImpedanceControlMode Name of the controller object
<i>degreeOfFreedom_1, degreeOfFreedom_2, ...</i>	Type: CartDOF List of degrees of freedom to be described
<i>setParameter(value)</i>	Method for setting a controller parameter A separate method is available for each settable <i>parameter</i> (<i>value</i> = 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);

lbr.move(lingetApplicationData().getFrame("/")
P1").setCartVelocity(800).setMode(cartImpCtrlMode));
```

17.5.2.3 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>
setAdditionalControl-Force(...)	<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"> ■ ≥ 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 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 currently overlaid without a delay. If the force to be overlaid is too great, this can result in overloading of the robot and cancellation of the program. The class <code>CartesianSinelmpedanceControlMode</code> has the option of overlaying forces after a delay.</p>

17.5.2.4 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>
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>maxForceXYZ</i>: Maximum force at the TCP in the corresponding Cartesian direction (type: double, unit: N) <ul style="list-style-type: none"> ■ ≥ 0.0 ■ <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 ■ <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.

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
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

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 1 cm, 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);

```

17.6 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 `CartesianSinelImpedanceControlMode`.

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.

17.6.1 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.

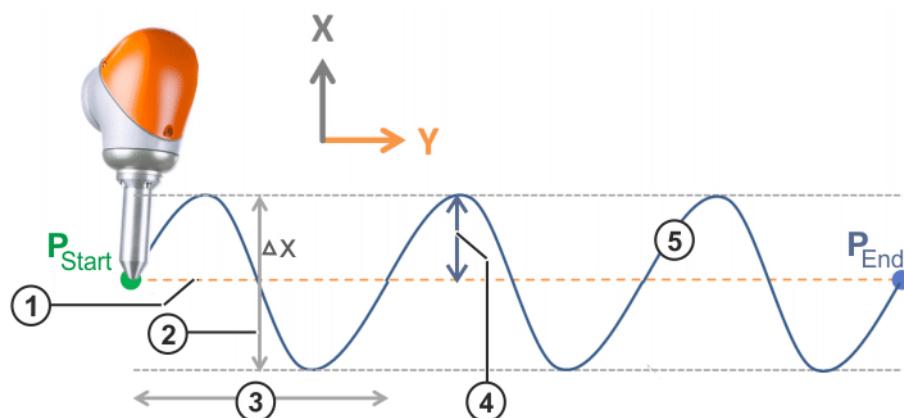


Fig. 17-6: Overlaying a simple force oscillation

- | | |
|-------------------------|-------------|
| 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:

$$\lambda = c / f = \text{robot velocity} / \text{frequency}$$

17.6.2 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.

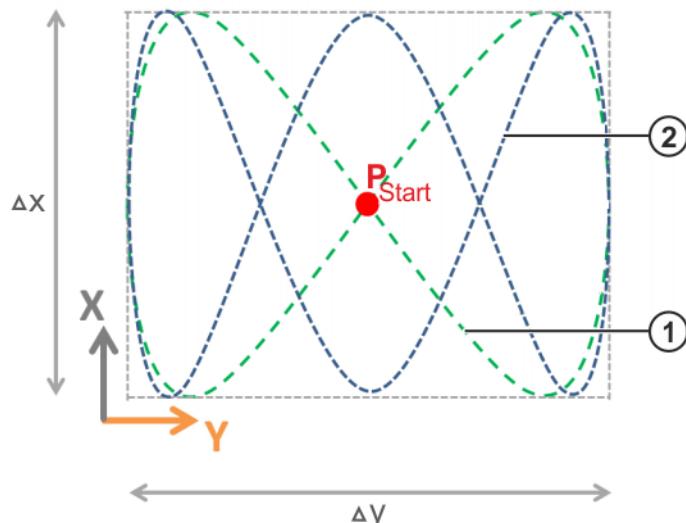


Fig. 17-7: Path of a Lissajous curve

- 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.

17.6.3 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.



In impedance control, 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, resulting in unpredictable motions of the robot.

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.

(>>> 17.5.2 "Parameterization of the impedance controller" Page 289)



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

17.6.3.1 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.</p> <p>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>
setBias(...)	<p>Constant force overlaid in addition to the force oscillation (type: double) The additionally overlaid constant force shifts the force oscillation in the parameterized direction. Using setBias(...), it is also possible to generate constant forces without oscillation for the impedance controller with overlaid force oscillation.</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: If no additional constant force is overlaid for a degree of freedom, the default value is used for this degree of freedom.</p>

Method	Description
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 Default: Not limited. <p>Rotational degrees of freedom (unit: Nm):</p> <ul style="list-style-type: none"> ■ ≥ 0.0 Default: Not limited. <p>Note: If no force limit is specified for a degree of freedom, the default value is used for this degree of freedom.</p>
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));
```

17.6.3.2 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 300)</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 = $(1/\text{frequency}) * 0.5$
- 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(...)`.

17.7 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.robotsAPI.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

17.7.1 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
	<i>controlMode</i>	Type: CartesianSineImpedanceControlMode Name of the controller object
	<i>degreeOfFreedom</i>	Type: CartDOF Degree of freedom for which the constant force is to be overlaid.
	<i>force</i>	Type: double Value of the overlaid constant force. Corresponds to the call of setBias(...) 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
	<i>stiffness</i>	Type: double 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

17.7.2 Overlaying a simple force oscillation

Description The createSinePattern(...) method overlays a simple force oscillation in one Cartesian direction.

Syntax

```
controlMode = CartesianSineImpedanceControlMode.create-
SinePattern(CartDOF.degreeOfFreedom, frequency, amplitude,
stiffness);
```

Explanation of the syntax	Element	Description
	<i>controlMode</i>	Type: CartesianSineImpedanceControlMode Name of the controller object
	<i>degreeOfFreedom</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]) <ul style="list-style-type: none">■ 0.0 ... 15.0

Element	Description
<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);
lbr.move(linRel(0.0, 150.0,
0.0).setCartVelocity(100).setMode(sineMode));
```

17.7.3 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:

- Ratio amplitude 1st degree of freedom : 2nd degree of freedom: 1 : 1
- Ratio frequency 1st degree of freedom : 2nd degree of freedom: 1 : 0.4
- Phase offset between 1st and 2nd degree of freedom: $\frac{1}{2} * \pi$

Syntax

```
controlMode = CartesianSineImpedanceControlMode.createLissajousPattern(CartPlane.plane, frequency, amplitude, stiffness);
```

Explanation of the syntax

Element	Description
<i>controlMode</i>	Type: <code>CartesianSineImpedanceControlMode</code> Name of the controller object
<i>plane</i>	Type: <code>CartPlane</code> Plane in which the Lissajous oscillation is to be overlaid

Element	Description
<i>frequency</i>	<p>Type: double</p> <p>Frequency of the oscillation for the first degree of freedom of the specified plane (unit: Hz)</p> <ul style="list-style-type: none"> ■ 0.0 ... 15.0 <p>The frequency for the second degree of freedom is calculated as follows:</p> <ul style="list-style-type: none"> ■ $frequency * 0.4$
<i>amplitude</i>	<p>Type: double</p> <p>Amplitude of the oscillation for both degrees of freedom on the specified plane (unit: N)</p> <ul style="list-style-type: none"> ■ ≥ 0.0
<i>stiffness</i>	<p>Type: double</p> <p>Stiffness values for both degrees of freedom of the specified plane (unit: N/m)</p> <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. 17-7](#))).

```
CartesianSineImpedanceControlMode lissajousMode;
lissajousMode =
CartesianSineImpedanceControlMode.createLissajousPattern(CartPlane.X
Y, 10.0, 50.0, 500.0);
lbr.move(linRel(0.0, 150.0,
0.0).setCartVelocity(100).setMode(lissajousMode));
```

17.8 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 set-point 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
<i>controlMode</i>	<p>Type: Subclass of <code>AbstractMotionControlMode</code></p> <p>Name of the controller object</p>
<i>time</i>	<p>Type: long</p> <p>Indicates how long the specified <i>controlMode</i> is to be held. The value must be ≥ 0. A value of < 0 indicates infinite.</p>
<i>unit</i>	<p>Type: Enum of type <code>TimeUnit</code></p> <p>Defines the unit of the specified time. The Enum is contained by default in the Java libraries.</p>

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);  
  
lbr.move(positionHold(controlMode, 10, TimeUnit.SECONDS));
```


18 Diagnosis

18.1 Displaying field bus errors



WorkVisual can be used for precise error analysis. More information on field bus diagnosis with WorkVisual is contained in the **WorkVisual** documentation.

18.1.1 General field bus errors

Description The general error state of the connected field buses can be displayed in Station view using the **KUKA_Sunrise_Cabinet** tile. Additional details can be displayed by opening the **Field buses** level.

Procedure

1. Open Station view.
2. Select the **KUKA_Sunrise_Cabinet** tile.
The status indicator of the **Field buses** tile indicates the collective state of all field buses connected to the controller.
3. Select the **Field buses** tile.
The detail view opens with error information about the currently connected field buses.

18.1.2 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, while the upper indicator shows the state of the selected I/O group.

Procedure

- In the navigation bar, select the I/O group from **I/O Groups**. The detail view of the I/O group opens. Any faulty inputs/outputs of the selected group are indicated.

18.2 Displaying the protocol

A protocol of the events and changes in state of the system can be displayed on the smartHMI.

Procedure

1. Open the Station view or the Robots view.
2. Select the **Protocol** tile. The **Protocol** view opens.
If the view is opened via the Robots view, only those protocol entries are displayed by default which affect the robot selected in the navigation bar.

18.2.1 “Protocol” view

Overview

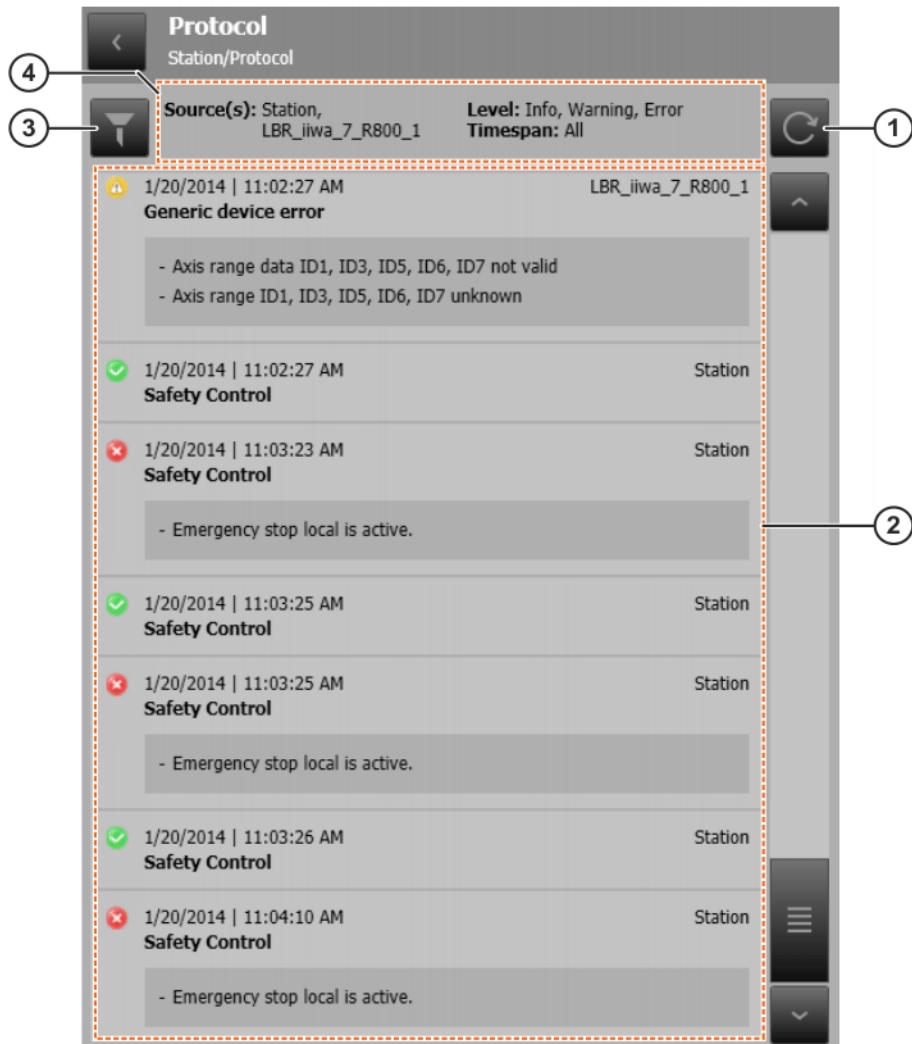


Fig. 18-1: “Protocol” view

Item	Description
1	Refresh button Refreshes the displayed protocol entries. After refreshing, the most recent entry is shown by default at the top of the list. If a time filter is active, the oldest entry is shown at the top of the list.
2	List of protocol entries (>>> "Log event" Page 308)
3	Filter settings button Opens the Filter settings window in which the protocol entries can be filtered according to various criteria.
4	Filter settings display The currently active filters are displayed here.

Log event

The protocol entries contain various information pertaining to each log event.

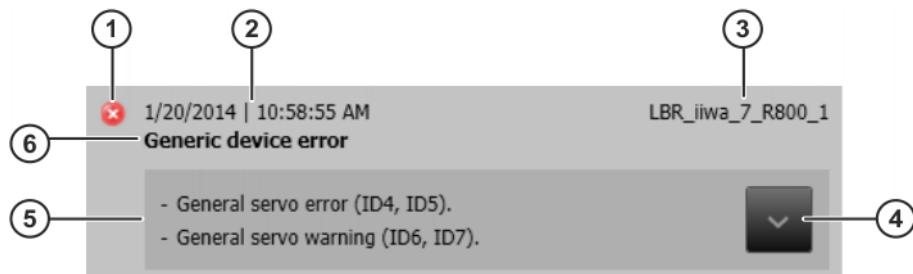


Fig. 18-2: Information about the log event

Item	Description
1	Log level of the event (>>> "Log level" Page 309)
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) By default, 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

18.2.2 Filtering protocol entries

Precondition

- The **Protocol** window is open.

Procedure

- Touch the **Filter settings** button. The **Filter settings** window is opened.
- 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 **Protocol** view is closed. When the view is re-opened, the default settings are reactivated.

Description

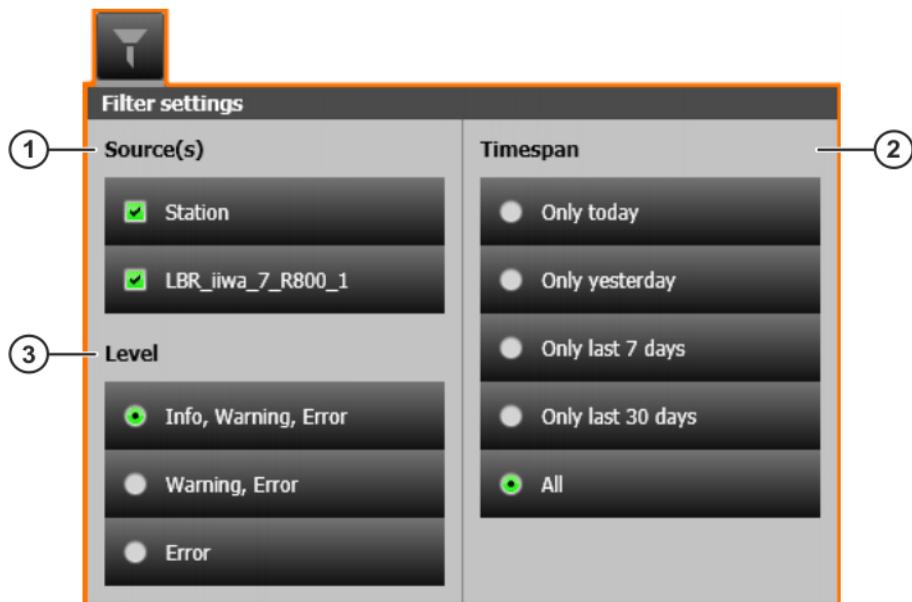


Fig. 18-3: “Filter settings” window

Item	Description
1	<p>Source(s) filter</p> <p>The protocol entries can be filtered according to the sources that caused the log event.</p> <ul style="list-style-type: none"> ■ Station: All protocol entries are displayed which affect the station and the inputs/outputs of field buses. ■ Robot: Only those protocol entries are displayed which affect the robot selected in the navigation bar, here an LBR iiwa 7 R800. <p>Default for Protocol in Station view: Both sources are selected.</p> <p>Default for Protocol in Robots view: The source is the robot selected in the navigation bar.</p>
2	<p>Timespan filter:</p> <p>A time filter can be activated to display only the protocol entries of a specific timespan.</p> <p>Default: All (no time filter active)</p>
3	<p>Level filter</p> <p>The protocol entries can be filtered according to their log level.</p> <p>Default: Info, Warning, Error (no filter active for log level)</p>

18.3 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 diagnostics 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.

18.3.1 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

- 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.
- In the main menu, select **Diagnosis > Create diagnosis package** and select the desired file location.
 - **Hard disk**
 - **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.

18.3.2 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** diagnostic 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

- Press the “Main menu” key and hold it down.
- Press the keypad key twice.
- Release the “Main menu” key.

The diagnostic information is grouped. 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.

18.3.3 Creating a diagnostic package with Sunrise.Workbench

Precondition

- Network connection to the robot controller

Procedure

- Right-click on the project in the **Package Explorer** and select **Sunrise > Create diagnostics package** from the context menu. The wizard for creating the diagnosis package opens.
- 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**. Click on **OK** to confirm.

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 folder in Windows Explorer**.
5. Click on **Finish**. The wizard is closed.



Projects and applications are not included in the diagnostics package. It is advisable to transfer these data separately, as they can contain important information for troubleshooting.

18.3.4 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 diagnostics 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 copied. If necessary, create a folder for the diagnosis package by clicking on **Create new folder**. Click on **OK** to confirm.
3. Activate the radio button **Copy existing diagnosis packages to local file system** and select the desired diagnosis packages.
4. Click on **Next >**. The diagnosis package is 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 package was copied, e.g. to send it directly by e-mail, click on **Open folder in Windows Explorer**.
6. Click on **Finish**. The wizard is closed.

19 KUKA Service

19.1 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:

- Model and serial number of the manipulator
- Model and serial number of the controller
- Model and serial number of the linear unit (if present)
- Model and serial number of the energy supply system (if present)
- Version of the system software
- Optional software or modifications
- Diagnostic package **KrcDiag**:
Additionally for KUKA Sunrise: Existing projects including applications
For versions of KUKA System Software older than V8: Archive of the software (**KrcDiag** is not yet available here.)
- Application used
- External axes used
- Description of the problem, duration and frequency of the fault

19.2 KUKA Customer Support

Availability KUKA Customer Support is available in many countries. Please do not hesitate to contact us if you have any questions.

Argentina Ruben Costantini S.A. (Agency)
Luis Angel Huergo 13 20
Parque Industrial
2400 San Francisco (CBA)
Argentina
Tel. +54 3564 421033
Fax +54 3564 428877
ventas@costantini-sa.com

Australia Headland Machinery Pty. Ltd.
Victoria (Head Office & Showroom)
95 Highbury Road
Burwood
Victoria 31 25
Australia
Tel. +61 3 9244-3500
Fax +61 3 9244-3501
vic@headland.com.au
www.headland.com.au

Belgium	KUKA Automatisering + Robots N.V. Centrum Zuid 1031 3530 Houthalen Belgium Tel. +32 11 516160 Fax +32 11 526794 info@kuka.be www.kuka.be
Brazil	KUKA Roboter do Brasil Ltda. Travessa Claudio Armando, nº 171 Bloco 5 - Galpões 51/52 Bairro Assunção CEP 09861-7630 São Bernardo do Campo - SP Brazil Tel. +55 11 4942-8299 Fax +55 11 2201-7883 info@kuka-roboter.com.br www.kuka-roboter.com.br
Chile	Robotec S.A. (Agency) Santiago de Chile Chile Tel. +56 2 331-5951 Fax +56 2 331-5952 robotec@robotec.cl www.robotec.cl
China	KUKA Robotics China Co.,Ltd. Songjiang Industrial Zone No. 388 Minshen Road 201612 Shanghai China Tel. +86 21 6787-1888 Fax +86 21 6787-1803 www.kuka-robotics.cn
Germany	KUKA Roboter GmbH Zugspitzstr. 140 86165 Augsburg Germany Tel. +49 821 797-4000 Fax +49 821 797-1616 info@kuka-roboter.de www.kuka-roboter.de

France	KUKA Automatisme + Robotique SAS Techvallée 6, Avenue du Parc 91140 Villebon S/Yvette France Tel. +33 1 6931660-0 Fax +33 1 6931660-1 commercial@kuka.fr www.kuka.fr
India	KUKA Robotics India Pvt. Ltd. Office Number-7, German Centre, Level 12, Building No. - 9B DLF Cyber City Phase III 122 002 Gurgaon Haryana India Tel. +91 124 4635774 Fax +91 124 4635773 info@kuka.in www.kuka.in
Italy	KUKA Roboter Italia S.p.A. Via Pavia 9/a - int.6 10098 Rivoli (TO) Italy Tel. +39 011 959-5013 Fax +39 011 959-5141 kuka@kuka.it www.kuka.it
Japan	KUKA Robotics Japan K.K. YBP Technical Center 134 Godo-cho, Hodogaya-ku Yokohama, Kanagawa 240 0005 Japan Tel. +81 45 744 7691 Fax +81 45 744 7696 info@kuka.co.jp
Canada	KUKA Robotics Canada Ltd. 6710 Maritz Drive - Unit 4 Mississauga L5W 0A1 Ontario Canada Tel. +1 905 670-8600 Fax +1 905 670-8604 info@kukarobotics.com www.kuka-robotics.com/canada

Korea	KUKA Robotics Korea Co. Ltd. RIT Center 306, Gyeonggi Technopark 1271-11 Sa 3-dong, Sangnok-gu Ansan City, Gyeonggi Do 426-901 Korea Tel. +82 31 501-1451 Fax +82 31 501-1461 info@kukakorea.com
Malaysia	KUKA Robot Automation Sdn Bhd South East Asia Regional Office No. 24, Jalan TPP 1/10 Taman Industri Puchong 47100 Puchong Selangor Malaysia Tel. +60 3 8061-0613 or -0614 Fax +60 3 8061-7386 info@kuka.com.my
Mexico	KUKA de México S. de R.L. de C.V. Progreso #8 Col. Centro Industrial Puente de Vigas Tlalnepantla de Baz 54020 Estado de México Mexico Tel. +52 55 5203-8407 Fax +52 55 5203-8148 info@kuka.com.mx www.kuka-robotics.com/mexico
Norway	KUKA Sveiseanlegg + Roboter Sentrumsvegen 5 2867 Hov Norway Tel. +47 61 18 91 30 Fax +47 61 18 62 00 info@kuka.no
Austria	KUKA Roboter CEE GmbH Gruberstraße 2-4 4020 Linz Austria Tel. +43 7 32 78 47 52 Fax +43 7 32 79 38 80 office@kuka-roboter.at www.kuka.at

Poland	KUKA Roboter Austria GmbH Spółka z ograniczoną odpowiedzialnością Oddział w Polsce Ul. Porcelanowa 10 40-246 Katowice Poland Tel. +48 327 30 32 13 or -14 Fax +48 327 30 32 26 ServicePL@kuka-roboter.de
Portugal	KUKA Sistemas de Automatización S.A. Rua do Alto da Guerra nº 50 Armazém 04 2910 011 Setúbal Portugal Tel. +351 265 729780 Fax +351 265 729782 kuka@mail.telepac.pt
Russia	KUKA Robotics RUS Werbnaia ul. 8A 107143 Moskau Russia Tel. +7 495 781-31-20 Fax +7 495 781-31-19 info@kuka-robotics.ru www.kuka-robotics.ru
Sweden	KUKA Svetsanläggningar + Robotar AB A. Odhners gata 15 421 30 Västra Frölunda Sweden Tel. +46 31 7266-200 Fax +46 31 7266-201 info@kuka.se
Switzerland	KUKA Roboter Schweiz AG Industriestr. 9 5432 Neuenhof Switzerland Tel. +41 44 74490-90 Fax +41 44 74490-91 info@kuka-roboter.ch www.kuka-roboter.ch

Spain	KUKA Robots IBÉRICA, S.A. Pol. Industrial Torrent de la Pastera Carrer del Bages s/n 08800 Vilanova i la Geltrú (Barcelona) Spain Tel. +34 93 8142-353 Fax +34 93 8142-950 Comercial@kuka-e.com www.kuka-e.com
South Africa	Jendamark Automation LTD (Agency) 76a York Road North End 6000 Port Elizabeth South Africa Tel. +27 41 391 4700 Fax +27 41 373 3869 www.jendamark.co.za
Taiwan	KUKA Robot Automation Taiwan Co., Ltd. No. 249 Pujong Road Jungli City, Taoyuan County 320 Taiwan, R. O. C. Tel. +886 3 4331988 Fax +886 3 4331948 info@kuka.com.tw www.kuka.com.tw
Thailand	KUKA Robot Automation (M)SdnBhd Thailand Office c/o Maccall System Co. Ltd. 49/9-10 Soi Kingkaew 30 Kingkaew Road Tt. Rachatheva, A. Bangpli Samutprakarn 10540 Thailand Tel. +66 2 7502737 Fax +66 2 6612355 atika@ji-net.com www.kuka-roboter.de
Czech Republic	KUKA Roboter Austria GmbH Organisation Tschechien und Slowakei Sezemická 2757/2 193 00 Praha Horní Počernice Czech Republic Tel. +420 22 62 12 27 2 Fax +420 22 62 12 27 0 support@kuka.cz

Hungary KUKA Robotics Hungaria Kft.
Fö út 140
2335 Taksony
Hungary
Tel. +36 24 501609
Fax +36 24 477031
info@kuka-robotics.hu

USA KUKA Robotics Corporation
51870 Shelby Parkway
Shelby Township
48315-1787
Michigan
USA
Tel. +1 866 873-5852
Fax +1 866 329-5852
info@kukarobotics.com
www.kukarobotics.com

UK KUKA Automation + Robotics
Hereward Rise
Halesowen
B62 8AN
UK
Tel. +44 121 585-0800
Fax +44 121 585-0900
sales@kuka.co.uk

Index

Symbols

“Ready for motion”, polling 239

Numbers

2004/108/EC 38
2006/42/EC 38
89/336/EEC 38
95/16/EC 38

A

Accessories 17, 21
Activity, polling 240
Actual position, axis-specific 85
Actual position, Cartesian 86
addCartesianForce(...) 269
addCartesianTorque(...) 269
addCommandedCartesianPositionXYZ(...) 269
addCommandedJointPosition(...) 269
addControllerListener(...) 240, 243
addCurrentCartesianPositionXYZ(...) 270
addCurrentJointPosition(...) 269
addExternalJointTorque(...) 268
addInternalJointTorque(...) 268
AMF 14
ANSI/RIA R.15.06-2012 38
API 14
App_Enable 140
App_Start 140
Application data (view) 44
Application mode 79
Application server 14
Application tool 71
Application, stopping 275
Applied norms and regulations 38
Approximate positioning 193
areDataValid() 102
attachTo(...) 222
AUT 23
AutExt_Active 140
AutExt_AppReadyToStart 140
Auto-complete 206
Automatic 23
Automatic mode 36
Auxiliary point 186, 213
awaitFileAvailable(...) 272
Axis limit 166
Axis range 23, 166
Axis torque condition 247
Axis torque monitoring 168
Axis torques, polling 229
Axis-specific monitoring spaces, defining 166
Axis-specific position, polling 234

B

Background task, new 50
Background tasks 281
Base coordinate systems 69
Base for jogging 112

Blocking wait 266
BooleanIOCondition 246
Brake defect 33
Brake test 95
Brake test application, template 97
Brake test, evaluation 105
Brake test, performing 107
Brake test, polling results 106
Brake test, programming interface 101
Brake test, results (display) 108
Brake test, start of execution 104
Brake test, starting position 101
Brake, defective 96, 107
BrakeTest (class) 99, 104
BrakeTestResult (class) 105
BrakeTestResult.State (Enum) 106
Braking distance 23
Break conditions for motions 256
Break conditions, evaluating 256
breakWhen(...) 256, 257
Bus I/Os, mapping 134

C

Cartesian impedance controller 285, 287, 295
Cartesian position, polling 235
Cartesian protected spaces, defining 164
Cartesian setpoint/actual value difference, polling 236
Cartesian workspaces 163
CE mark 22
Checksum, safety configuration 155
CIRC 213
CIRC, motion type 186
Circular motion 213
Cleaning work 37
clipProgramOverride(...) 244
Collision detection 169
Complex conditions 246
Condition for Boolean signals 255
Condition for the range of values of a signal 255
Conditional branch 278
Conditions 245
Connecting cables 17, 21
Constant force, overlaying 301
Continuous Path 185
Controller object, creating 286
Controller parameters, defining 286
Controllers, overview 285
Coordinate system, for jog keys 61
Coordinate systems 69
Counting loop 275
CP motions 185
CP spline block 185
CP Spline block, creating 217
createAndEnableConditionObserver(...) 264
createConditionObserver(...) 264
createDesiredForce(...) 301
createLissajousPattern(...) 301

createNormalForceCondition(...) 248, 249
createShearForceCondition(...) 248, 250
createSinePattern(...) 301
createSpatialForceCondition(...) 248, 249
createSpiralPattern(...) 301
CRR 14, 23, 74
Cyclic background task 281

D

Danger zone 23
Data types 210
Data, recording and evaluating 267
DataRecorder 267
Debug (perspective) 45
Declaration of conformity 22
Declaration of incorporation 21, 22
Decommissioning 37
Default motion frame 119
DefaultApp_Error 140
detach() 225
Diagnosis 307
Diagnosis package, creating 311
Diagnosis package, loading from the robot controller 312
Diagnostic information, collecting 310
Diagnostic package, creating 311
displayModalDialog(...) 274
Disposal 37
DO WHILE loop 277
Documentation, industrial robot 13

E

EC declaration of conformity 22
Electromagnetic compatibility (EMC) 38
EMC Directive 22, 38
EMERGENCY STOP 58
EMERGENCY STOP device 26, 27, 28
EMERGENCY STOP, external 26, 28, 160
EN 60204-1 + A1 39
EN 61000-6-2 38
EN 61000-6-4 + A1 38
EN 614-1 38
EN ISO 10218-1 38
EN ISO 12100 38
EN ISO 13849-1 38
EN ISO 13849-2 38
EN ISO 13850 38
enable(), DataRecorder 270
Enabling device 26, 27
Enabling device, external 26, 29
Enabling switch 59, 60
Enabling switches 27
equals(...) 257
execute(...) 104
External control 139

F

Fast entry, Java 207
Faults 33
Field bus errors, displaying 307
Filter settings 308

Flange coordinate system 69
Fonts 209
FOR loop 275
Force component condition 251
Force condition 248
ForceComponentCondition 246
ForceCondition 245
Frame 14
Frame management 111
Frames, addressing 80
Frames, creating 112
Frames, deleting 113
Frames, displaying 75
Frames, moving 113
Frames, teaching 77
Function test 34

G

General safety measures 32
getAlphaRad() 237
getApplicationData().createFromTemplate() 222
getApplicationData().getFrame() 116
getAxis() 105
getBetaRad() 237
getCommandedCartesianPosition(...) 234
getCommandedCartesianPosition() 261
getCommandedJointPosition() 234, 261
getCurrentCartesianPosition() 234, 261
getCurrentJointPosition() 234, 261
getEmergencyStopEx() 242
getEmergencyStopInt() 242
getExecutionMode() 243
getExternalForceTorque(...) 230, 231
getExternalTorque() 229
getFiredBreakConditionInfo() 257
getFiredCondition() 257, 258, 260
getFlange() 222
getForce() 231
getForceInaccuracy() 232
getFrame(...) 223, 225
getGammaRad() 237
getHomePosition() 238
getLogLevel() 106
getMaxAbsMsrTorqueValues() 102
getMeasuredTorque() 105, 229
getMissedEvents() 260
getMotionContainer() 261
getObserverManager() 264, 266
getOperationMode() 242
getOperatorSafetyState() 242
getPositionInformation(...) 234
getPositionInformation() 236, 261
getProgramOverride() 244
getRotationOffset() 236
getSafetyState() 241
getSafetyStopSignal() 242
getSingleMaxAbsMsrTorqueValue(...) 102
getSingleTorqueValue(...) 229
getStartTimeStamp() 102
getState() 106
getStopTimeStamp() 102

getTestedTorque() 106
 getTimestamp() 106
 getTorque() 231
 getTorqueInaccuracy() 232
 getTorqueValues(...) 229
 getTranslationOffset() 236
 getTriggerTime() 261
 Graphics card 41
 Guard interlock 28

H

halt() 275
 Hand-held control panel 17, 21
 Hard disk space 41
 Hardware 41
 hasActiveMotionCommand() 240
 HOME position 237
 HOME position, changing 237
 HOME position, polling 238
 Hooke's law 287
 HOV 62, 72
 HRC 14

I

I/O configuration, exporting 136
 I/O configuration, new 129
 I/O configuration, opening 130
 I/O group, creating 132
 I/O group, deleting 133
 I/O group, editing 133
 I/O group, exporting as a template 133
 I/O group, importing from a template 134
 I/O Mapping (window) 134, 135
 IAnyEdgeListener 262
 ICallbackAction, interface 259
 ICondition, interface 245
 IControllerStateListener 240
 Identification plate 59
 IF ELSE branch 278
 IF FallingEdgeListener 262
 Industrial robot 21
 Information in Javadoc, displaying 208
 initialize() 206, 282, 284
 initializeCyclic(...) 282
 Inputs/outputs, display 87
 Installation 127
 Installation, KUKA Sunrise.Workbench 41
 Intended use 19, 21
 Introduction 13
 IORangeCondition 246
 IP address, displaying 89
 IP addresses 47
 IRisingEdgeListener 262
 isEnabled() 272
 isFileAvailable() 272
 isForceValid(...) 232
 isInHome() 238
 isMastered() 239
 isReadyToMove() 239
 isRecording() 272
 isTorqueValid(...) 232

ISunriseControllerStateListener 243
 ITriggerAction, interface 259

J

Java Editor 205
 Java Editor, opening 205
 Java package, new 49
 Java project, new 53
 Java projects, referencing 54
 Javadoc 14
 Javadoc (view) 45
 Jog keys 58, 72, 73
 Jog mode 31
 Jog override 62, 72
 Jogging type 62
 Jogging, axis-specific 70, 72
 Jogging, Cartesian 70, 73
 Jogging, robot 70
 Joint Path 185
 JointTorqueCondition 245
 JP motions 185
 JP spline block 185
 JP Spline block, creating 218
 JRE 14

K

Keyboard key 58
 Keypad 63
 Knowledge, required 13
 KRCDiag 310
 KUKA Customer Support 313
 KUKA smartHMI 14, 61
 KUKA smartPAD 14, 23, 33, 57
 KUKA Sunrise Cabinet 17
 KUKA Sunrise.Workbench, installation 41
 KUKA Sunrise.Workbench, user interface 43
 KUKA PSM, PSM table 153
 KUKA Sunrise Cabinet 14
 KUKA Sunrise.OS 14
 KUKA Sunrise.Workbench, starting 43
 KUKA Sunrise.Workbench, uninstalling 41

L

Labeling 31
 Liability 21
 LIN 213
 LIN REL 214
 LIN, motion type 186
 Linear motion 213, 214
 Lissajous oscillation, overlaying 303
 Load data 120
 Load data, entering 120
 Loops, nesting 279
 Low Voltage Directive 22

M

Machine data 35
 Machinery Directive 22, 38
 Main menu key 58
 Main menu, calling 67
 main() 206

- Maintenance 36
Manipulator 17, 21, 23, 26
Manual mode 35
Mapping, inputs/outputs 136
Mastering 91
Mastering state, polling 239
Mastering, deleting 91
MeasuredTorqueEvaluator (class) 98, 99, 101
MeasuredTorqueStatistic (class) 99, 101, 102
Menu bar 44
Message programming 273
Message window 81
Methods, extracting 208
Mode selection 30
Monitoring 246, 262
Monitoring of processes 246
Monitoring processes 262
Monitoring spaces 161
Motion parameters 218
Motion programming, basic principles 185
Motion types 185
MotionBatch 215
MotionPathCondition 246
`move(...)` 211, 225
`moveAsync(...)` 211, 225
- N**
Navigation bar 62
Non-cyclic background task 283
Non-safety-oriented functions 30
Normal force 248
NotificationType, Enum 265
Null space motion 74
- O**
Object management 116
ObserverManager 264, 266
`onIsReadyToMoveChanged(...)` 240
`onSafetyStateChanged(...)` 243
`onTriggerFired(...)` 259
Operating mode, changing 68
Operation, KUKA smartPAD 57
Operation, KUKA Sunrise.Workbench 43
Operator 23, 25
Operator safety 26, 28
Operators 247
Options 17, 21
Orientation control 220
Orientation control, LIN, CIRC, SPL 195
Output, change 87
Overload 33
Override 62, 72, 81, 83
Overview of the robot system 17
Overview, safety acceptance 175
- P**
Package Explorer (view) 44
Panic position 27
Passwort, changing 156
Path-related condition 253
Path-related switching actions 246, 258
- Performance Level 22
Personnel 24
Perspectives, display 45
Perspektive, selection 44
Plant integrator 24
Point-to-point 185
Point-to-point motion 212
Polling, robot position 233
Position and torque referencing 172
Position controller 285, 286
Position referencing 172
`positionHold(...)` 304
Post-test loop 277
POV 81, 83
Preventive maintenance work 37
Processor 41
Product description 17
Program execution 80
Program override 81, 83
Program override, changing and polling 244
Program run mode, changing and polling 243
Program run mode, selecting 82
Program, starting 83
Program, starting automatically 83
Program, starting manually 83
Programming 205
Programming (perspective) 45
Project management 111
Project, loading from the robot controller 124
Project, synchronizing 122
Project, transferring to the robot controller 123
Projects, archiving 51
Projects, loading to workspace 52
Properties (view) 45
Protected space 161, 164
Protective equipment 31
Protocol entries, filtering 309
Protocol, display 307
Protocol, view 308
PSM 15
PSM table, KUKA PSM 153
PSM table, User PSM 153
PSM table, user PSM 153
PTP 212
PTP, motion type 185
- R**
RAM 41
Reaction distance 23
Ready for motion signal, reacting to change 240
Recommissioning 34, 91
Redundancy angle 200
Redundancy information 115, 199
Reference, canceling 54
Rejecting loop 276
Renaming, variable 206
Repair 36
Retraction, robot 74
Robot application, new 49
Robot application, selecting 80
Robot base coordinate system 69

Robot controller 21
 Robot controller, switching on/off 60
 Robot position, polling 233
 Robot, repositioning 84
 RoboticsAPI 15
 RoboticsAPI, displaying the version 211
 Robots view 65
 run() 206, 284
 runCyclic() 282

S

Safe operational stop, external 26, 29
 Safeguards, external 32
 Safety 21
 Safety acceptance, overview 175
 Safety configuration 145
 Safety configuration, activating 155
 Safety configuration, deactivating 156
 Safety configuration, editing 154
 Safety configuration, opening 152
 Safety configuration, restoring 156
 Safety function, deleting 154
 Safety function, new 154
 Safety functions 22
 Safety functions, editing 154
 Safety instructions 13
 Safety of machinery 38
 Safety signals, evaluating 241
 Safety signals, polling 241
 Safety stop 24
 Safety stop 0 24
 Safety stop 1 24
 Safety stop, external 26, 28
 Safety zone 24, 25, 26
 Safety-oriented functions 26
 Safety-oriented stop reactions 29
 Safety-oriented tool 120
 Safety-oriented tool, defining 121
 Safety, legal framework 21
 SafetyConfiguration.sconf (file) 49, 152
 Service life 23
 Service, KUKA Roboter 313
 Set methods 218
 setAdditionalControlForce(...) 292
 setAmplitude(...) 298
 setBias(...) 298
 setBlendingCart(...) 220
 setBlendingOri(...) 220
 setBlendingRel(...) 219
 setCartAcceleration(...) 219
 setCartJerk(...) 219
 setCartVelocity(...) 219
 setDamping(...) 292
 setExecutionMode(...) 243
 setFallTime(...) 300
 setForceLimit(...) 299
 setfrequency(...) 298
 setHoldTime(...) 300
 setHomePosition(...) 237
 setJointAccelerationRel(...) 219, 220
 setJointJerkRel(...) 219, 220

setJointVelocityRel(...) 219, 220
 setMaxCartesianVelocity(...) 294
 setMaxControlForce(...) 293
 setMaxPathDeviation(...) 294
 setNullSpaceDamping(...) 293
 setNullSpaceStiffness(...) 293
 setOrientationReferenceSystem(...) 197, 220
 setOrientationType(...) 195, 220
 setPhaseDeg(...) 298
 setPositionLimit(...) 299
 setRiseTime(...) 300
 setStayActiveUntilPatternFinished(...) 300
 setStiffness(...) 292
 setTotalTime(...) 300
 Shearing force 248
 Signal state, polling 241
 Signal state, reacting to change 243
 Simple force oscillation, overlaying 302
 Single point of control 37
 Singularities 202
 Singularity 195
 smartHMI 14, 61
 smartPAD 14, 23, 33, 57
 Software 17, 21, 41
 Software components 18
 Software limit switches 31
 Software version, displaying 89
 Space Mouse 58
 SPL, motion type 187
 Spline segment 187
 Spline, motion type 187
 SPOC 37
 Start backwards key 58
 Start-up 34, 91
 Start/pause key 58, 59
 startEvaluation() 101
 Starting, program 83
 Starting, system software 60
 startRecording() 270
 StartRecordingAction 270
 Station configuration 127
 Station configuration, opening 127
 Station view 64
 Station_Error 140
 StationSetup.cat (file) 49, 127
 Status 200
 Status display 63
 Status keys 58
 Stop category 0 24
 Stop category 1 24
 STOP key 58
 Stop reactions, safety-oriented 29
 stopEvaluation() 101
 Stopping distance 23, 26, 163
 stopRecording() 271
 StopRecordingAction 271
 Storage 37
 Structure of a motion command 211
 Structure, robot application 205
 Sunrise I/Os, changing 133
 Sunrise I/Os, creating 130

Sunrise I/Os, deleting 133
Sunrise project, new 46
SunriseExecutionService 243
Support request 313
Surface normal 248
Switching off, robot controller 60
Switching on, robot controller 60
Symbols 209
Synchronization, project 122
System integrator 22, 24
System requirements, PC 41
System software, installing 127
System states, polling 238

T

T1 24
T2 24
Target group 13
Tasks (view) 45
TCP 15, 118
TCP force monitoring 170
Template data (view) 44
Template, for Sunrise project 46
Templates 207
Templates, user-specific 207
Terms used 14
Terms used, safety 23
Terms, used 14
Tool coordinate system 70
Tool frame, creating 118
Tool load data, determining 92
Tool, creating 118
Tool, switching off 158
Toolbars 44, 46
Tools, declaring 221
Tools, initializing 222
Torque referencing 173
Torque value determination 99
Torques, axis-specific 86
Touch screen 57
Trademarks 14
Training 13
Transportation 33
Trigger 246, 258
Trigger information, evaluating 260
Triggers, programming 258
triggerWhen(...) 259
Turn 201

U

Unmastering 91
USB connection 59
Use, contrary to intended use 21
Use, improper 21
User 23, 24
User dialogs, programming 274
User interface, KUKA smartHMI 61
User interface, KUKA Sunrise.Workbench 43
User messages, programming 273
User PSM, PSM table 153

V

Variable, renaming 206
Variables 210
Velocity 72, 83
Velocity monitoring functions 160
Version information, RoboticsAPI 210
View, protocol 308
Views, repositioning 45

W

waitFor(...) 266
Warnings 13
WHILE loop 276
Workpiece frame, creating 118
Workpiece, creating 118
Workpieces, declaring 221
Workpieces, initializing 222
Workspace 23, 25, 26, 161, 163, 166
Workspace, new 51
Workspace, Sunrise.Workbench 51
Workspace, switching 51
Workspaces, switching 51
World coordinate system 69

