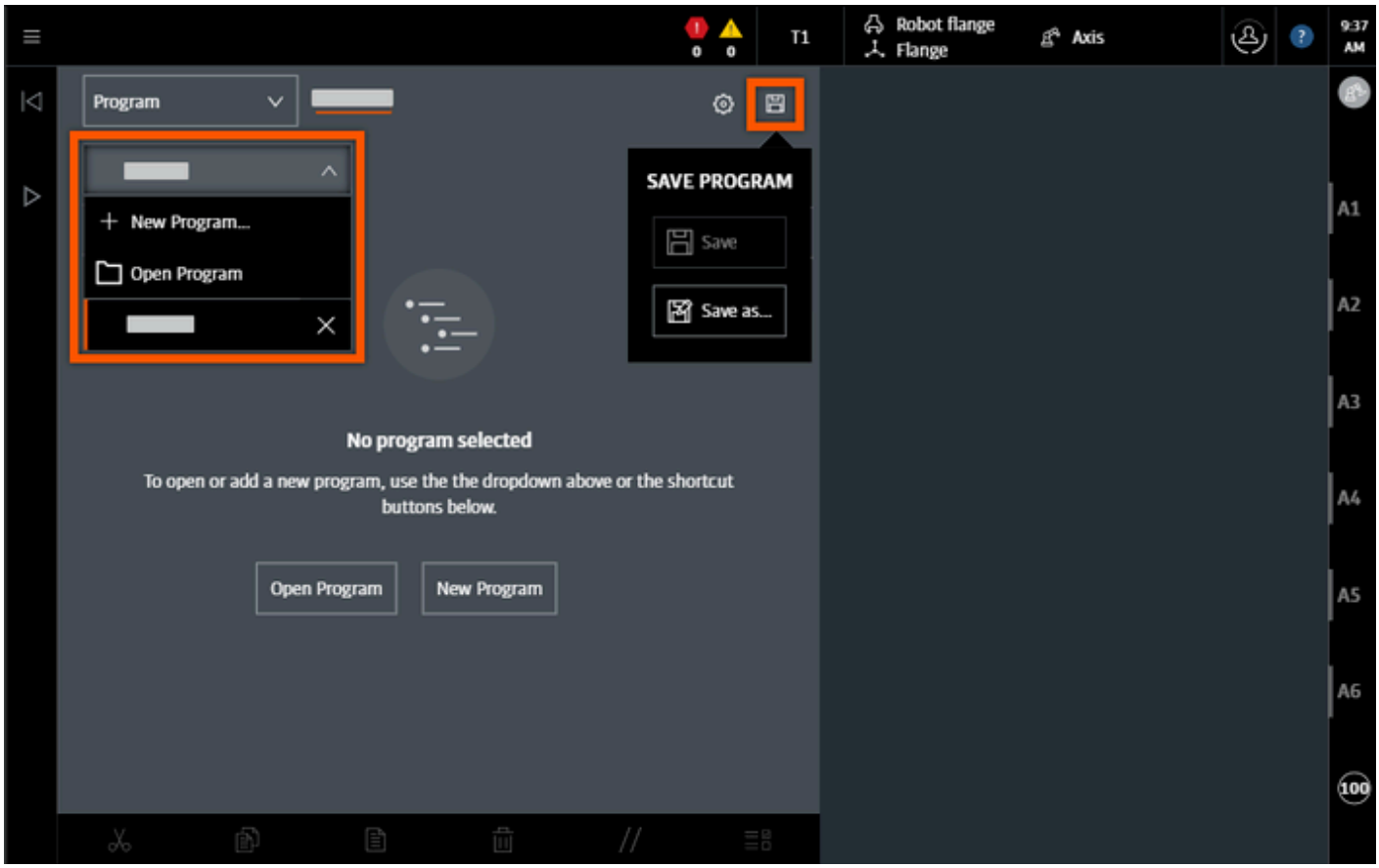
KUKA | Xpert

Identification number: CS429

Programming

Creating and managing programs

The system can contain multiple programs that can be opened, edited or executed if required. The various programs are managed in the Feature menu **Program**.



Managing programs

These functions are available:

- Create a new program
- Manage existing programs in the **OPEN PROGRAM** dialog:
  - Open an existing program
  - Duplicate an existing program
  - Delete an existing program

  Recently opened programs are displayed directly in the program menu and can be opened from there.

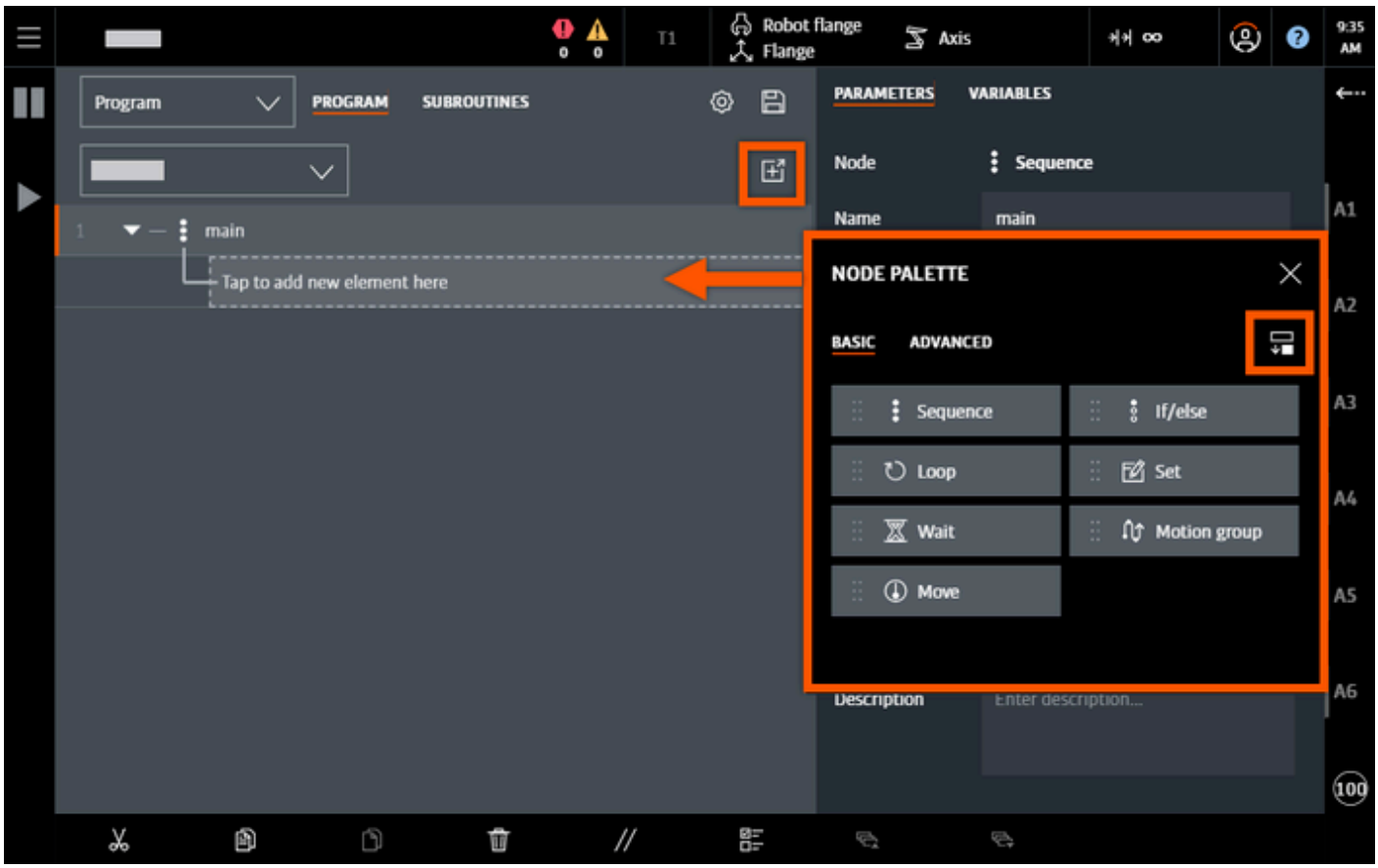
- In the **Save** dialog:

- Save the program currently open
- Save a copy of the current program under a new name

Node Palette: inserting nodes (program elements)

A program consists of multiple program elements or program lines, so-called nodes. Nodes have a hierarchical structure and can be nested in each other. New nodes are inserted into the program via the **Node Palette**.

It is advisable to start an application from a defined position. The command for addressing this position should be an **Axis (PTP)** motion that is inserted into the new program as the first motion command.



Node Palette

- The node palette can be moved freely on the interface so that it does not conceal any currently required content.
- The nodes in the pallet are divided into the categories **BASIC** and **ADVANCED**.
- Press a node to insert it below the node selected in the program.

  New nodes can be inserted as the last element below the current node (**Last child**), or at the same hierarchy level behind the current node (**Sibling after**).

  The insertion position is indicated in the program tree structure.

- Alternatively, press a node and drag it to the desired position in the program to place it there.

  If a program line is marked in yellow after insertion, the hierarchy is invalid or there are still entries missing in the command that are required to ensure that it is complete.

Basic nodes

Sequence

The **Sequence** node is used to structure the program, similar to files in a folder system. The **Sequence** node only calls all child nodes one after the other. Several nodes of the type **Sequence** can be nested in each other.

Each program contains a **Sequence** node named **main** at the highest level. This node can be renamed but not deleted.

Motion group

Description

The node can be used to program a motion group. A motion group always consists of the node **Motion group** and one or more nodes of type **Move** (motion).

Parameters

The following parameters can be defined on the **Parameters** tab:

- **Motion type**: Motion type with which the motions of the group are carried out
  - **Toolpath (Cartesian)**

    Cartesian motion in which the tool is moved along a defined path
  - **Axis (PTP)**

    Axis motion of the robot
- **TCP**: TCP with which the motions of the group are carried out

  The TCP that is currently selected in the status bar is preselected by default.
- **Speed**: Velocity at which the motions of the group are carried out
- **Acceleration**: Acceleration at which the motions of the group are carried out

The parameters set here are automatically applied to all motions below the **Motion group** node. Other values for velocity and acceleration can be programmed for individual motions (node **Move**).

 When teaching frames within a motion group, the parameters (TCP, velocity and acceleration) set in the **Motion group** node are used and not the TCP selected in the status bar.

 It is advisable to start an application from a defined position. The command for addressing this position should be an **Axis (PTP)** motion that is inserted into the new program as the first motion command.

 **Risk of injury and damage to property due to subsequent modification of the payload**

If the payload (and thus also the TCP) is modified after motion segments have already been inserted, all motion segments use the new payload. This results in a different robot pose and the robot may execute unexpected motions. Injuries or damage to property may result.

- Always test new or modified programs in operating mode T1 first.

Sensitivity

Settings to move the robot under impedance control can be made on the **Sensitivity** tab.

(>>> Programming of a compliant robot)

Triggers

A trigger can be programmed on the **Triggers** tab.

(>>> Trigger programming)

Error executing a Motion group

When executing a **Motion group**, it is possible that the program will be stopped with the following error message: **Motion command execution did not succeed. State: {motionState}.**

To remedy this, it is advisable to check the programming as follows:

- Check whether all end frames in the affected **Motion group** are accessible, for example by manually addressing the frames with **Move to**
- Check whether the frame coordinates are located in the workspace of the robot, for example via the scene
- Check whether the correct payload has been selected
- Check whether the correct base has been selected

If the **Toolpath (Cartesian)** motion type has been selected:

- Check whether adding one or more intermediate frames changes the behavior
- Check whether the motion type can be changed to **Axis (PTP)** (it is possible that the path will change!)

Move (motion)

Description

The **Move** node is always part of a motion group (**Motion group**). The node can be used to program the following motions:

- Absolute motion

  By default, absolute motion is preselected.
- Relative motion

Absolute Motion

If absolute motion is selected, the **Move** node defines the motion type and the target frame of the motion:

- The motion type must conform to the motion type of the motion group (**Motion group** node):
  - A Cartesian motion group can contain linear, circular and spline motions.
  - An axis-specific motion group can only contain Axis (PTP) motions.
- Target frames are the coordinates of the target position relative to the TCP. The frame coordinates can be entered manually or taught. When adding a new **Move** node, an unshared frame is automatically created.

  The functional principle of frames is described in the 3D scene.
- Unshared frames can subsequently be converted to project frames and vice versa. To do so, change the selection in **Target Type**. Project frames are known in the entire project on the robot controller and can be used in every program.

> **NOTICE** **Unexpected robot motion due to retaught frames**
>
> If a frame is retaught, the change affects all motions that move to this frame.
>
> - If **Move** nodes with frames that have already been taught are copied in order to use them for new motions, make sure to create and teach new frames in the respective segments.

- To create a new frame, select **Select or add frame** and then **New project frame** or **New unshared frame** in the **Frame** menu.

  Do not press the **Touch up** button. This button does not create a new frame, but overwrites the existing frame with new coordinates. If it is a global frame, it is modified in all programs that use it.

- The link **Open in scene** can be used to display the current frame in the 3D scene in order to be able to visualize its position better.

  In the scene, press the program name in the status bar at the top to return to the program.

Relative Motion

If relative motion is selected, the **Move** node defines the motion type and the target or rotational position of the motion:

- The motion type must conform to the motion type of the motion group (**Motion group** node):
  - A Cartesian motion group can contain linear, circular and spline motions.
  - An axis-specific motion group can only contain Axis (PTP) motions.
- The reference determines the direction of the relative motion. By default, the reference is set to the TCP of the **Motion group**. Alternatively, the reference can be set to the WORLD coordinate system.
- The **Position offset** parameter can be used to specify the relative motion from the current position of the robot. Depending on this position and the respective reference coordinate system, the target position of the relative motion is obtained.
- The **Angle offset** parameter can be used to specify a relative rotation about each axis. This rotation is carried out relative to the previous rotational position and depends on the respective reference coordinate system.

Move type

- **Axis (PTP):**

  The robot guides the TCP (Tool Center Point) along the fastest path to the target point. Since the motions of all robot axes to the target point are simultaneous and rotational, the resulting path is a curved path.

  The axis (joint) motion is a fast positioning motion. The exact path of the motion cannot be predicted and depends on the override, velocity and acceleration, but can always be reproduced identically as long as these general conditions are not altered.

- **Linear:**

  The robot guides the TCP (Tool Center Point) at a defined velocity and acceleration along a straight line to the target point.

- **Circle:**

  The robot guides the TCP (Tool Center Point) at a defined velocity and acceleration along a circular segment to the end point. In order to define the radius of the circle segment, a circular segment requires two end points: the end point of the motion (**Target frame 2**) and an auxiliary point (**Target frame 1**) situated on the circular segment that is passed through on the way to the end point.

- **Spline:**

  Spline is a motion type that is particularly suitable for complex, curved paths. The path is defined by several commands of type **Move** whose end frames are located on the path. The desired path can thus be generated easily. The path always remains the same, irrespective of the override setting, velocity or acceleration. The robot moves as fast as possible within the constraints of the programmed velocity, i.e. as fast as its physical limits will allow.

Loop

A loop repeats all subordinate nodes in the specified order until a certain condition is met.

Possible conditions for exiting the loop are:

- **Count**

  The loop is executed exactly n times.

- **Timeout**

  The loop is executed for n milliseconds.

- **Infinite**

  The loop is executed for an infinite duration and not exited.

  This is suitable if the break condition is to be programmed within the loop.

- **While**

  The loop is executed as long as a condition is met, e.g. as long as a specific signal is present.

- **Until**

  The loop is executed as long as a condition is met, e.g. until a specific signal is present.

Conditions for exiting the loop cannot be nested or linked.

Wait

With this node, the program pauses until a condition is met.

Conditions for resuming the program can be the following:

- **While**

  The program is paused as long as a condition is met, e.g. as long as a signal is present.

- **Until**

  The program is paused until a condition is met, e.g. until a signal is present or a return value takes on a specific state.

- **Millis**

  The program is paused for n milliseconds.

Conditions cannot be nested or linked.

Set

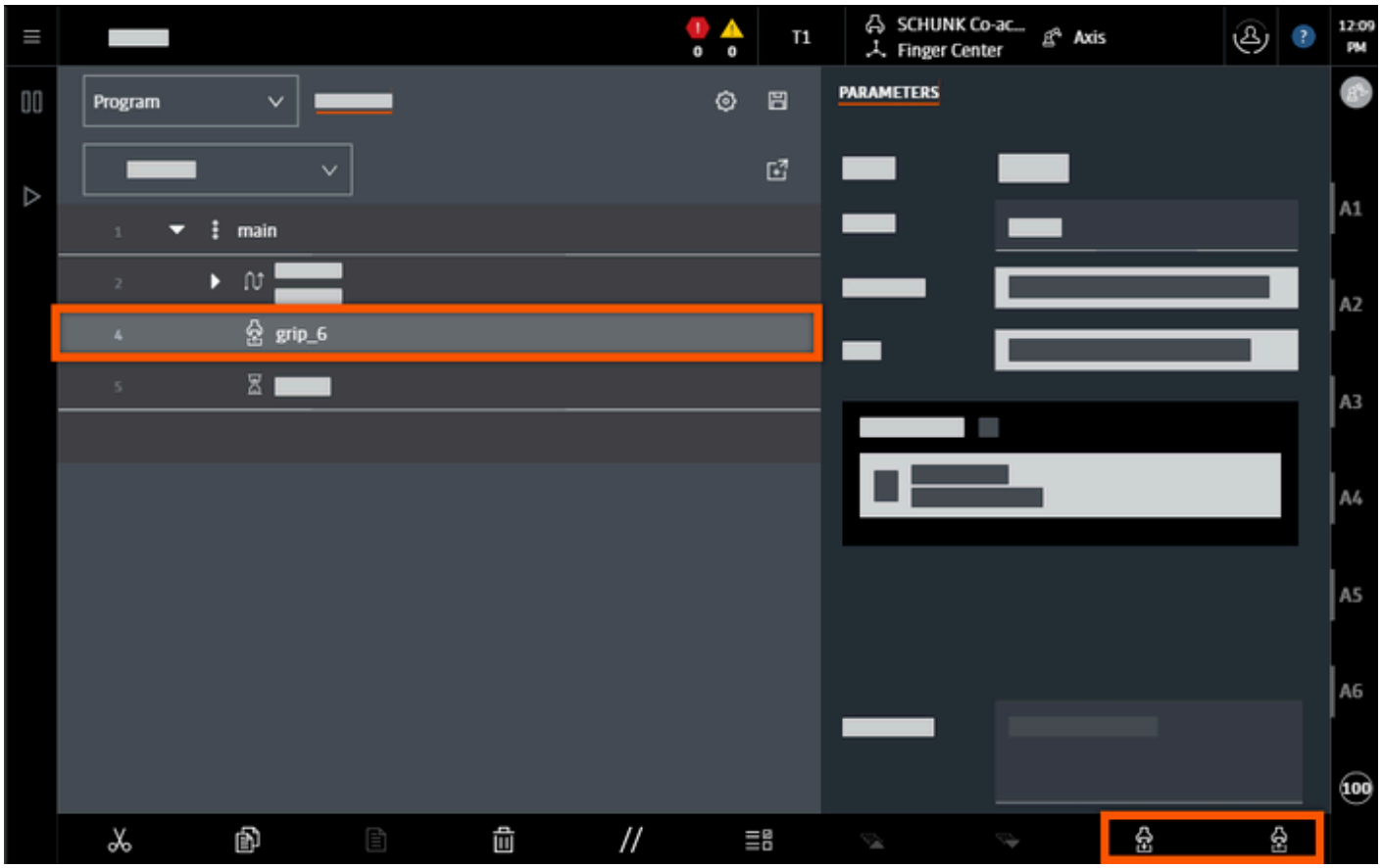
This node is used to assign values by means of expressions. It is possible, for example, to set the values of signals.

 Further information about signals can be found in the help function under “I/O configuration”.

Grip & Release

The two nodes of type **Capability** are only available if a gripper is selected from a preinstalled toolbox. Both nodes open or close the gripper on the robot flange. The gripper must be configured for this.

 Further information can be found in the help function under “Payload configuration”.



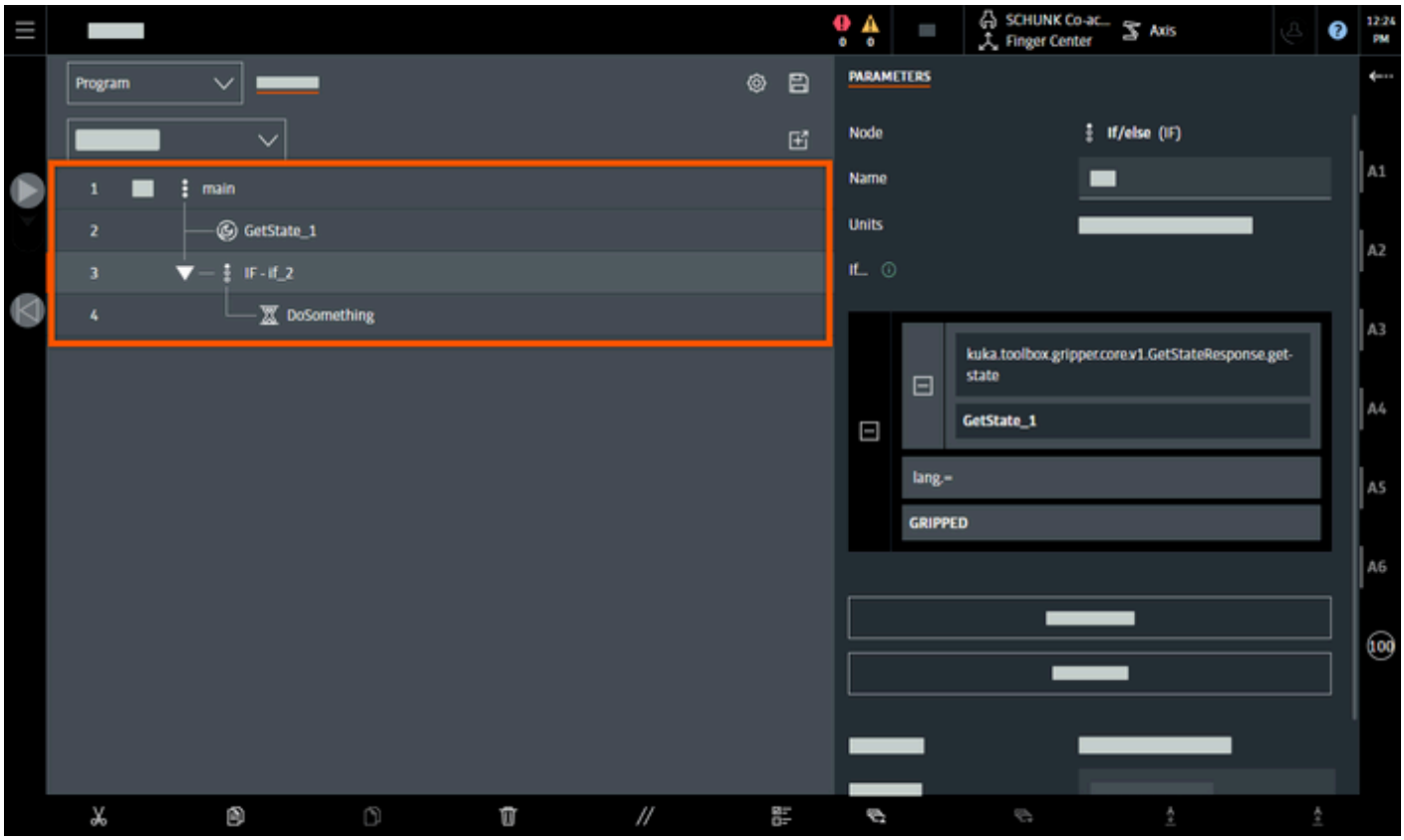
Grip / **Release**

- For every **Grip** / **Release** command, a set of **Capabilities** must be selected with which it is to be executed. The tool that executes the commands is determined automatically by this set.
- To simplify program creation, the commands can be tested using the buttons in the tool bar. For this, the robot must be in T1 mode, no program may be executed and the enabling switch must be pressed.
- If several capabilities are configured, e.g. for a Custom Dual Gripper, the capability to be operated can be assigned to the buttons in the toolbar by means of quick access.
  - Pressing briefly if not yet assigned: Assign capability
  - Pressing briefly if already assigned: Operate assigned capability
  - Pressing longer: Reassign capability
- Depending on how the set of **Capabilities** is configured, the command returns a return value. This return value can be used in another following node, e.g. as a break condition for a loop or with the **Wait** command.

  This can be used, for example, to ensure that the gripper is completely open or closed before the program is resumed.

GetState

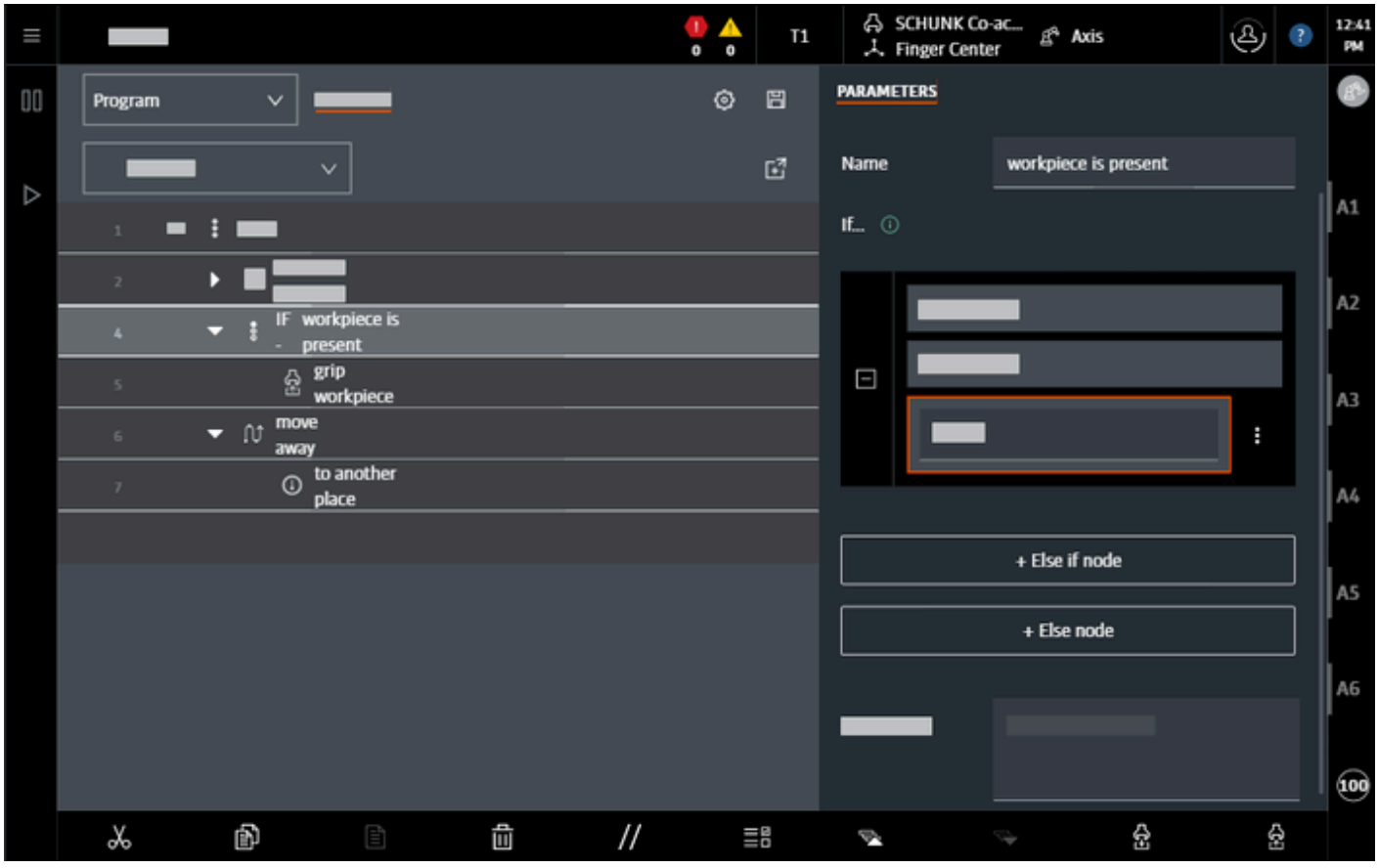
This node of type **Capability** is only available if a gripper is selected from a preinstalled toolbox. This node can be used to query the status of the gripper. The answer, e.g. **Gripped**, is saved in a variable. This variable can be used in subsequent nodes, e.g. If nodes. The variable can be inserted in the selection menu of the Expression Editor under **Library** > **Variables** > **Return Variables**.



GetState

If/else – simple branch

This node can be used to program the simplest form of a branch: If a specific condition is fulfilled, then all nodes under the If node are executed. Otherwise, they are skipped and the program is resumed at the next node.



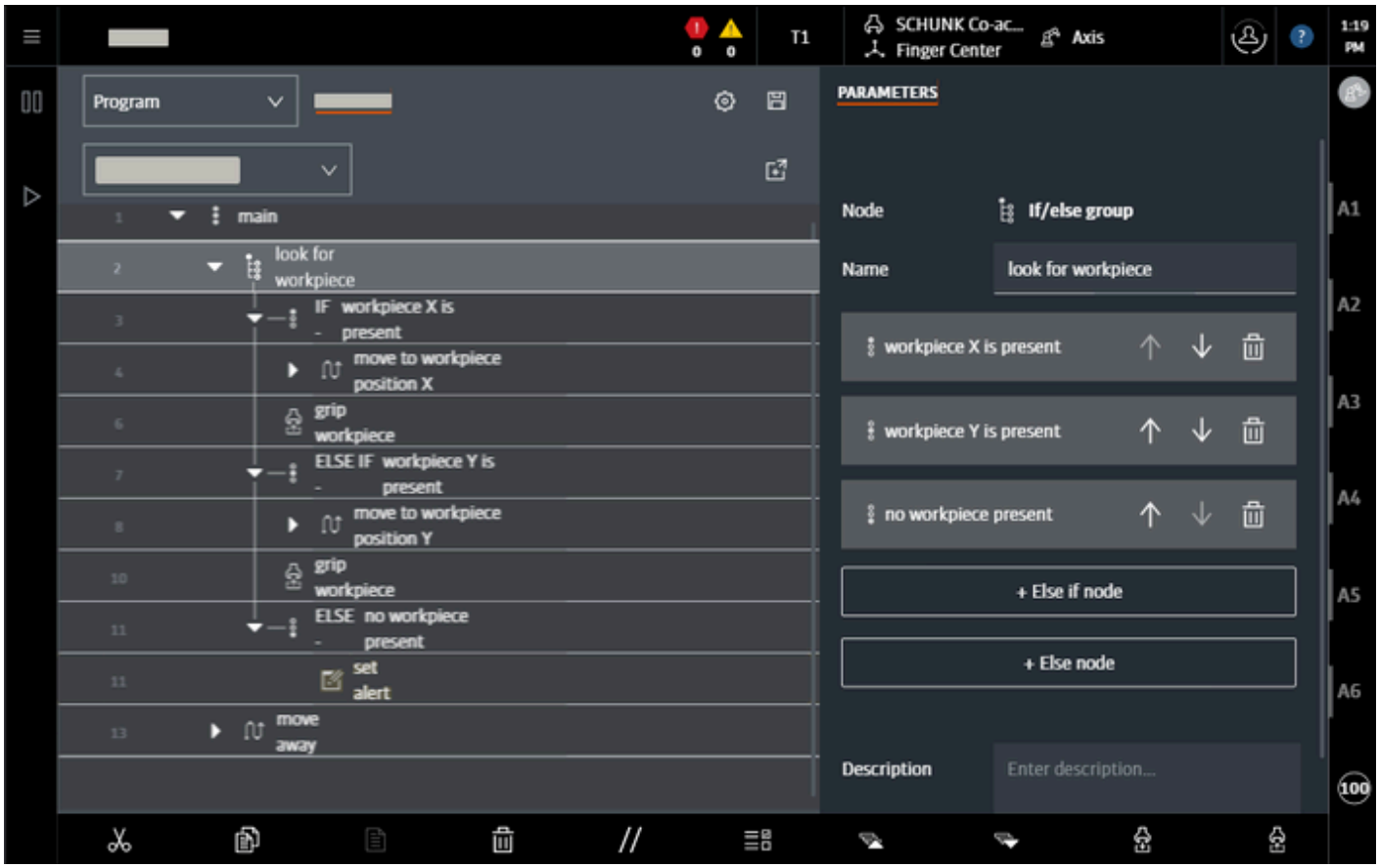
If/else

- Conditions which are mutually exclusive, i.e. no more than one of which is to be executed in any case, are programmed in an **If/else group**.
- An If node can easily be converted to an **If/else group** by adding an Else If or an Else node in the parameter view.

Advanced nodes

If/else group – advanced branch

Advanced branches can be programmed using **If/else group**. It is possible to formulate multiple conditions of which at most one or exactly one is executed.



If/else group

- Every **If/else group** begins with an If node.

  This condition is checked first. If it is met, the rest of the **If/else group** will be skipped and the program will be resumed after the **If/else group**.

- Multiple Else If nodes can be inserted after the first If node.

  These conditions are checked one after the other if the previous condition is not met. If an Else If condition is met, the subordinate branch is executed and the remainder of the **If/else group** is skipped.

- If one of the branches is to be executed in any case, insert an Else node as the last node. This is executed if none of the other conditions is met.
- In the parameter view of the **If/else group**, the order of the conditions can be changed and individual conditions can be deleted.

Break (exit loop or resume)

Description

The **Break** node can be used to exit a loop or resume it. The following break types can be selected in the parameter view of the node:

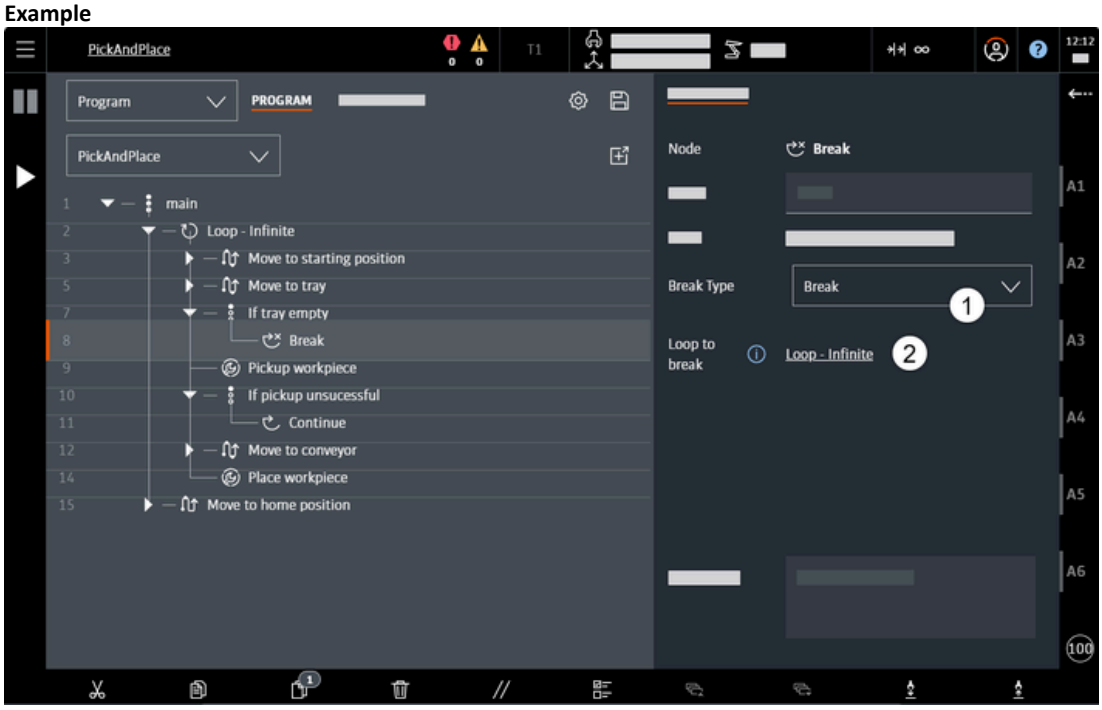
- **Break**

  Aborts the execution of the loop. The loop is exited and the program is resumed with the next command.

- **Continue**

  Aborts the current loop execution. The loop is resumed with the next loop execution.

Example



Break node in a loop

1	Selection of break type
2	Name of the loop that is aborted

The robot takes a workpiece out of a compartment and places it on a conveyor. The loop is executed as long as workpieces are available. If a workpiece is available, but the robot cannot pick it up, the robot attempts it again from the start position.

Line	Description
2 … 14	Endless loop
7 … 8	**Break** When the compartment is empty, the loop is exited and the program is resumed at line 15.
10 … 11	**Continue** If the removal of the workpiece was unsuccessful, the rest of the loop is skipped. The program jumps back to line 3.

Halt (pause or reset program)

There are two different types of **Halt type** available:

- **Pause**

  The program is paused. Pressing the Start key again resumes it at the same point; alternatively, the program can be reset manually.

- **Stop and reset**

  The program is terminated and reset. The next time it is executed, it is started from the beginning.

Subroutine (creating a subprogram)

The **Subroutine** node creates a new subroutine (subprogram). Subroutines are small portions of a program that can be used multiple times in a program. The subroutine is created on the **Subroutines** tab and can be edited there. In the program, the subroutine is declared below the first **Sequence** node.

(>>> Creating and executing subroutines)

Execute (execute subprogram)

The **Execute** node enables the execution of a predefined subroutine. In the parameter view, the subroutine that is to be executed can be selected under **Subroutine**. The subroutine is executed in the program at the point where the node is located in the program structure. Once the subroutine is completed, the program structure is resumed at the next node.

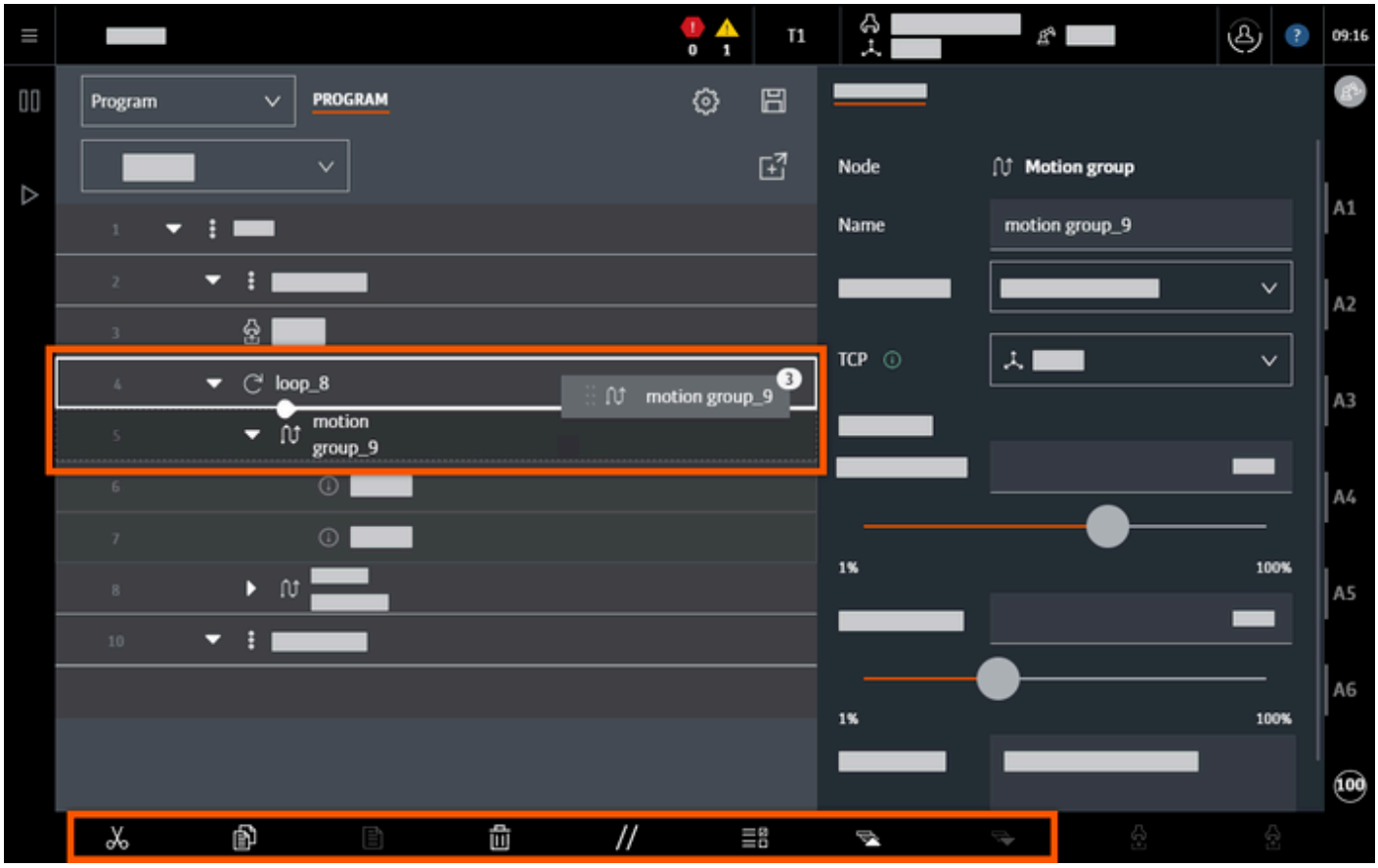
Position hold (holding position under impedance control)

The motion command **Position hold** makes it possible to hold an impedance-controlled robot at its Cartesian setpoint position for an adjustable period of time.

The **Position hold** motion command cannot be used in a motion group (**Motion group**) and combined with **Move** nodes.

(>>> Holding position under impedance control)

Editing the program structure



Editing the program structure

- To move a node and all its child nodes, hold it down until it is highlighted.

- Drag the node to the desired position. A white line displays the insertion position and the hierarchy level at which the node will be inserted.
- If nodes are nested in an impermissible manner and the hierarchy is invalid, the node in question is highlighted in yellow.
- The tool bar at the bottom of the screen contains additional functions for modifying the program structure:
  - Cut selected nodes.
  - Copy nodes.
  - Paste copied nodes.
  - Delete selected nodes or entire branches.
  - Deactivate selected nodes.

    Deactivated nodes are skipped during program execution and are not executed; they are commented out.
  - Toggle between single selection and multiple selection of nodes.
  - Expand or collapse all child nodes of the node currently selected.
- Nodes can also be exchanged between different programs via the commands in the tool bar.
- Nodes are inserted with default designations. Each node can be renamed to provide a better overview.

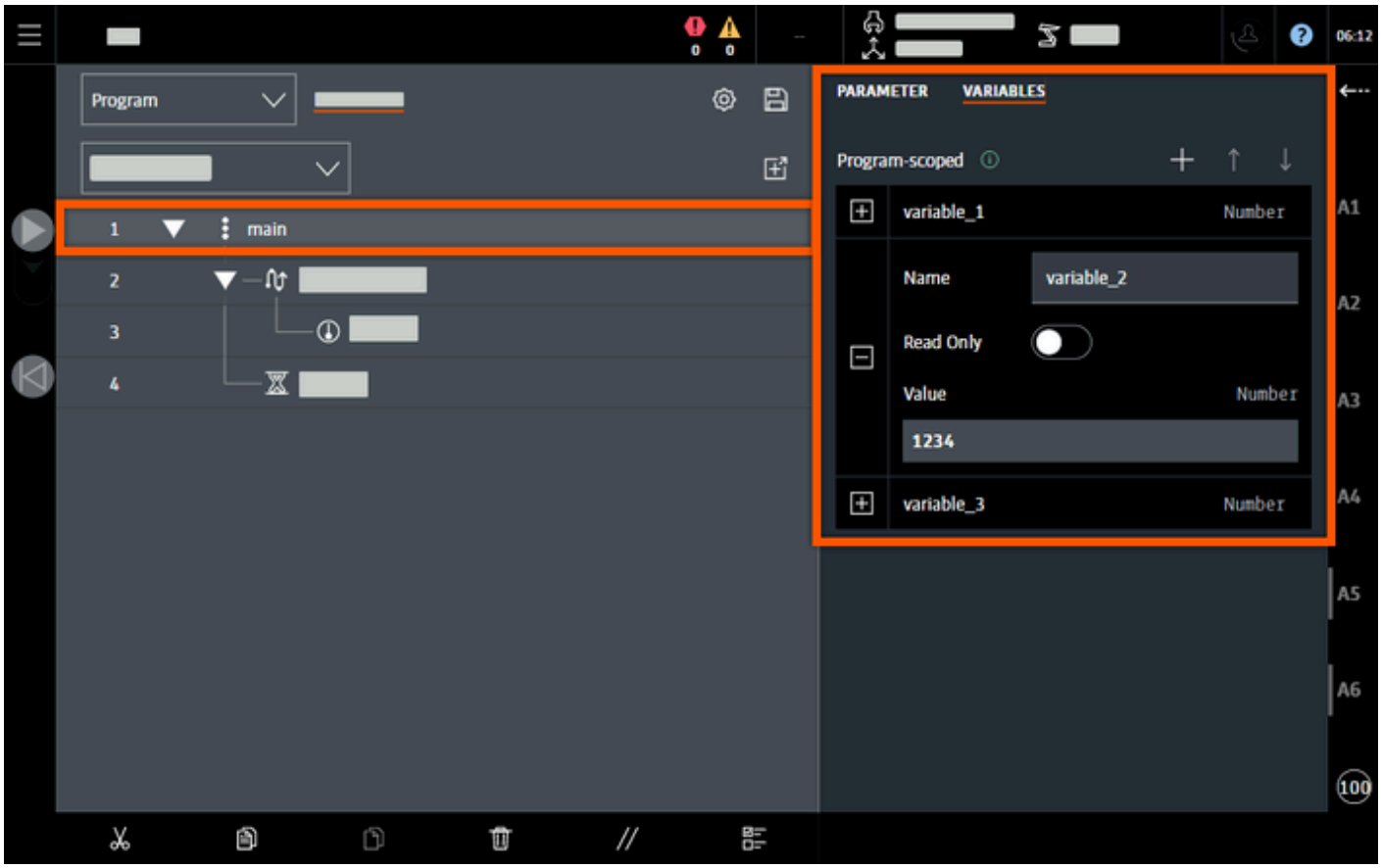
 Node deletion cannot be undone and moved nodes can only be manually moved back to their previous position. For this reason, save changes regularly during programming. Undesired changes can then be discarded by closing the program without saving.

Program variables

Overview

The **Variables** tab is available in the first node of a program (sequence with the default name **main**). There, variables for the program can be created, edited with the Expression Editor or deleted.

- The variables are valid throughout the program and are initialized when the program is started
- Variable values can be defined as constant
- Common data types are, for example, number, boolean, Vec3, quaternion
- The variables are available in the selection menu of the Expression Editor under **LIBRARY** > **Variables**
- The variables can be used in expressions:
  - Variable values can be read in corresponding nodes, e.g. in “if” conditions
  - Variable values can be written in the **Set** node
- If the program is restarted, the variables are initialized with the specified value.



Variables

Creating value with new data type

Description

By default, a newly created variable receives a value of type “Number”. To change the data type, proceed as follows. The procedure also applies if the data type has to be changed again during subsequent processing of the variable.

Procedure

1. Deleting default value/current value: To do so, select the value and select **Delete** from the context menu.
2. The value is now undefined. Select a new data type in the selection menu of the Expression Editor.

Example of a program variable

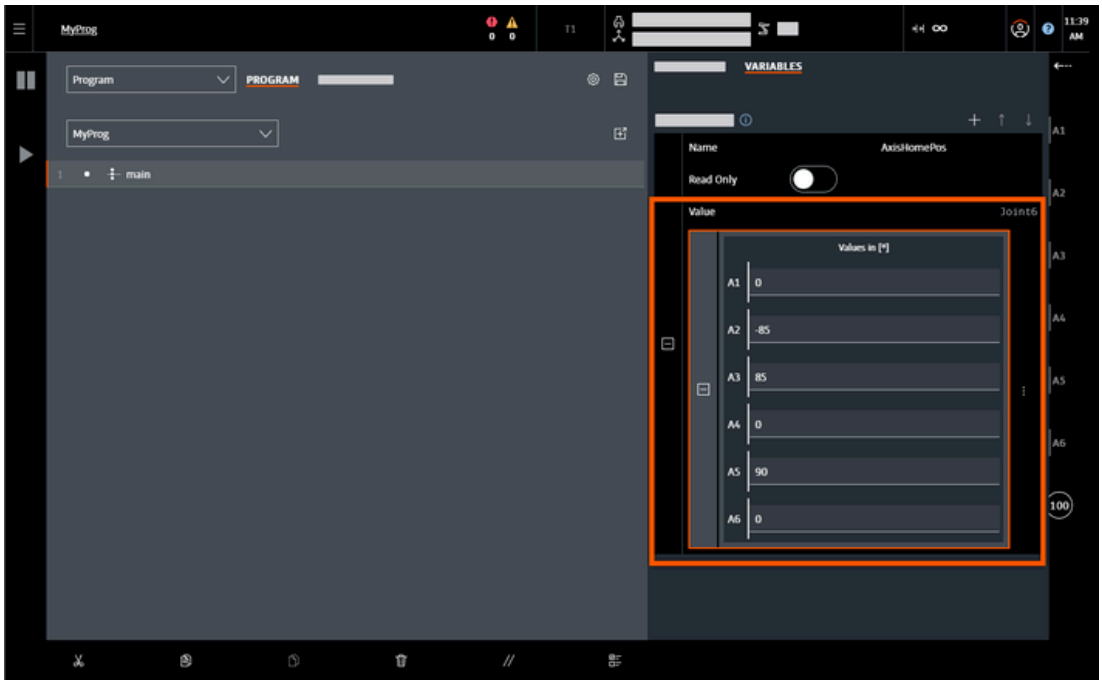
Description

An axis-specific home position is created as a variable for a program and used in a PTP motion to the home position.

Procedure

Create variable:

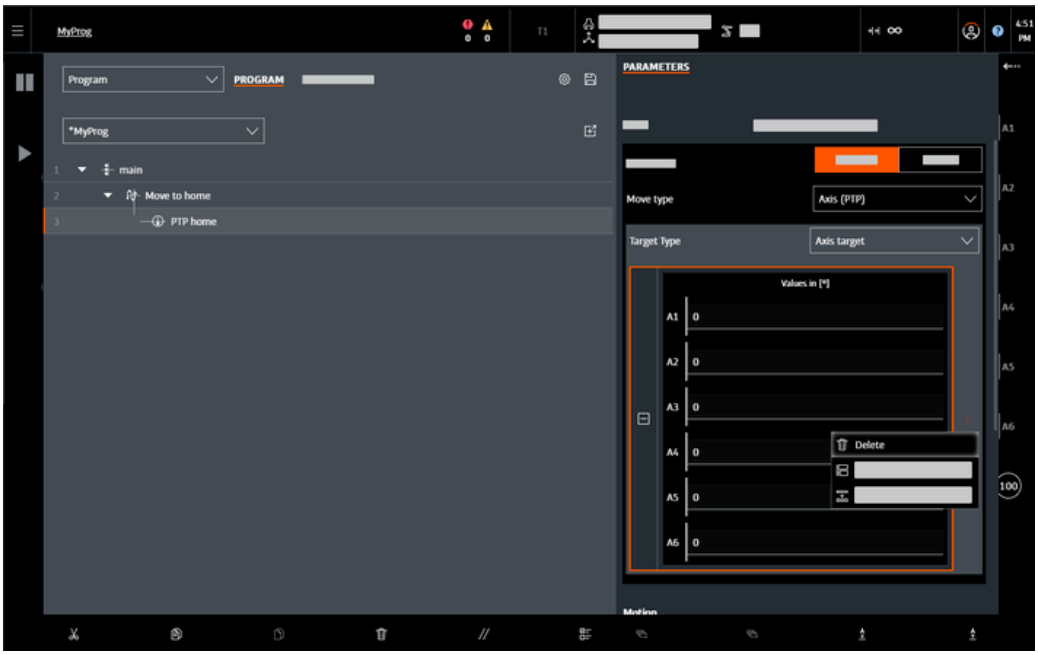
1. Add a new variable, select the default value and delete it via the context menu.
2. To enter axis positions in degrees, select **LIBRARY** > **Functions** > **robot** > **Joint6** > **from-deg()** in the selection menu.



Axis-specific home position as a variable

Use variable in motions:

1. In the motion group, select the motion type **Axis (PTP)** and the robot flange as the TCP.
2. In the motion, select the target type **Axis target**.
3. Select the axis values in the target type and delete them via the context menu.



Deleting axis values in target type

4. Apply the variable in the target type: For this, select the variable under **LIBRARY** > **Variables**.

Creating and executing subroutines

Description

Subroutines (subprograms) are small portions of a program that can be used multiple times in a program. Subroutines can be created in the Feature menu **Program** via the **Subroutines** tab. The subroutines in the program

can be executed via the **Execute** node. A subroutine can use all variables whose validity range includes the **Subroutine** node.

Procedure

Creating subroutines:

1. In the **Program** Feature menu, open the **Subroutines** tab.
2. Press **New subroutine**.
3. Assign a name for the subroutine.
4. Create the subroutine by adding nodes.
   Nodes can also be copied from the program and inserted into the subroutine.
5. Save the subroutine.

Executing subroutines:

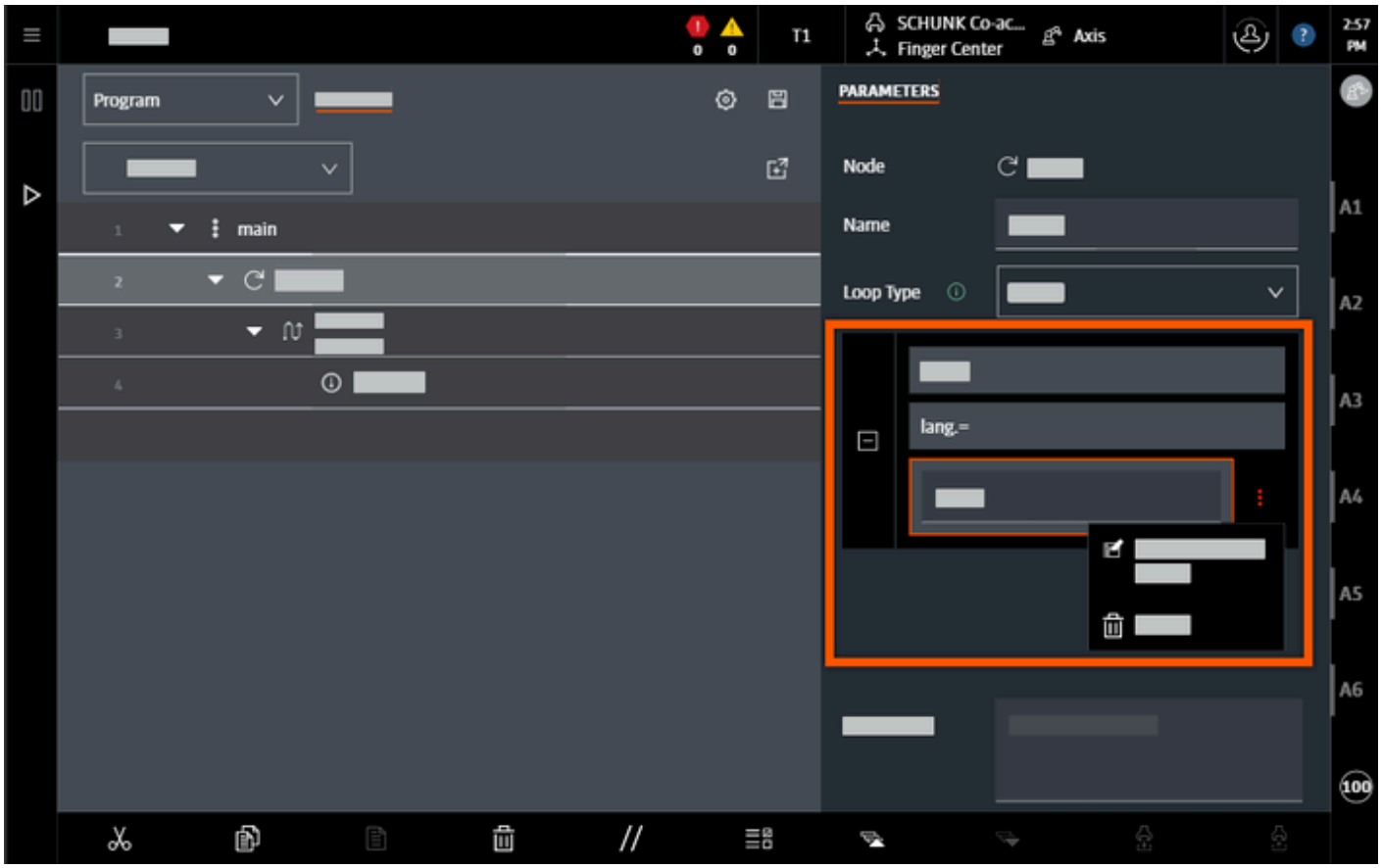
1. In the **Program** Feature menu, open the **Program** tab.
2. Open the node palette and switch to the **ADVANCED** tab.
3. Insert the **Execute** node at the desired point.
4. Select the desired subroutine in the parameter view under **Subroutine**.

Expression Editor

Description

The Expression Editor extends the programming capability of variables and nodes. Logical operators and mathematical functions enable the programming of complex expressions. The expressions are evaluated during the runtime of the program.

 Unless otherwise stated, the expressions indicate lengths in millimeters (mm) and angles in degrees (°). The system calculates in the background for lengths in meters (m) and axis angles in radians (rad). Therefore, when expressions are edited, values may be displayed in the converted unit.

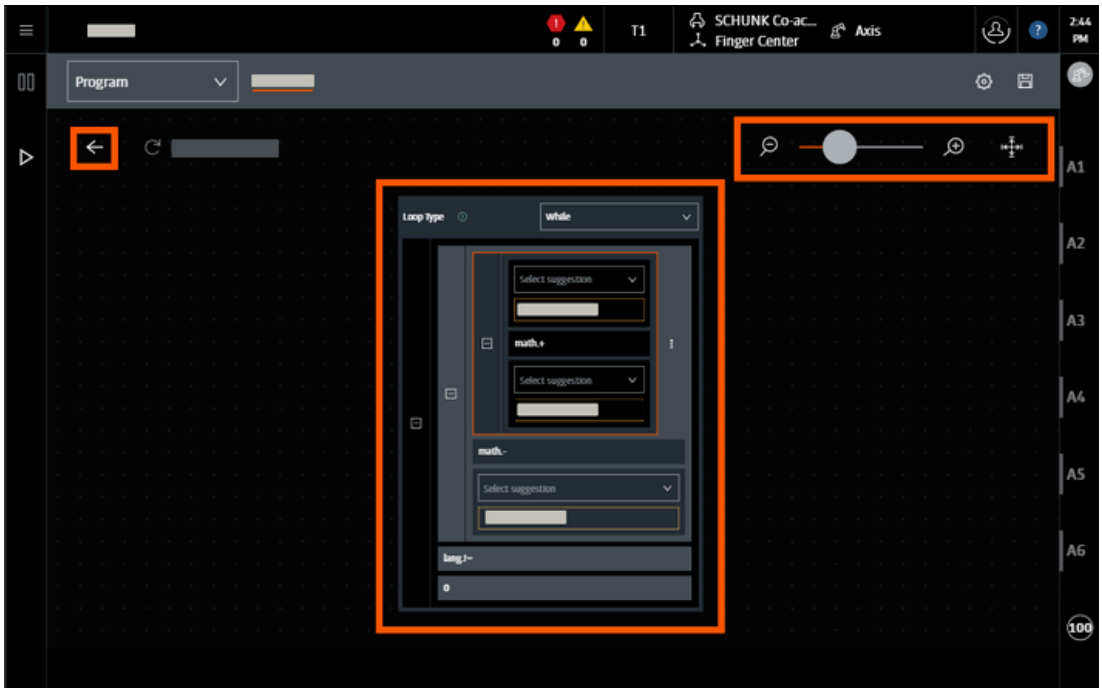


Expression Editor

- The Expression Editor is recursive. Any number of expressions can be nested together in a logic command.
- The Expression Editor only allows type-safe entries.
- Press on an empty expression to open a selection menu.
- Under **BASICS**, the selection menu suggests useful programming options for filling in the expression. All of the programming options available in the context, such as data, variables and functions, can be found under in the **LIBRARY**.
- Placeholders are displayed for mathematical operations, e.g. **? + ?** for addition. **0** must be selected for entering numbers.
- Values can be entered and edited using the keyboard.
- There is a context menu at the edge of each expression. The context menu can be used, for example, to open full screen mode or to delete the expression.
- Beyond a certain nesting depth, expressions can only be edited in full screen mode.
- Certain logic commands, e.g. the **Wait** node, are displayed in a simplified view (short form). They can also be displayed in their long form via the context menu.
- Expressions can be expanded and collapsed using the plus and minus icons.

Full screen mode

The Expression Editor can be opened in full screen mode in order to edit complex expressions in a larger view. To do so, select the desired expression and open the context menu at the side.



Expression Editor in full screen mode

- The Expression Editor can be moved freely on the screen.
- Move the slider between the magnifying glass icons to reduce or enlarge the Editor.
- Press the button (**Zoom to fit**) next to the magnifying glass icons to reset the view and align the Expression Editor to the left.
- Press the left arrow to exit full screen mode.

Functions in the Expression Editor

Description

The following namespaces are available in the Expression Editor:

Namespace	Description
geo	Functions and data types with reference to the modeled world
kuka	Functions and data types from toolboxes
lang	Basic functions and data types
math	Computational functions and mathematical data types
nodes	Data types with reference to specific nodes
robot	Functions and data types with reference to the robotics
trigger	Functions and data types for configuring triggers

The elements of a namespace are divided into the following menus by means of automatic categorization:

- **Data**

  The elements used to generate a data structure directly are always located under *Data*. These elements are referred to as constructors.

- **Functions**

  All remaining functions can be found under *Functions*.

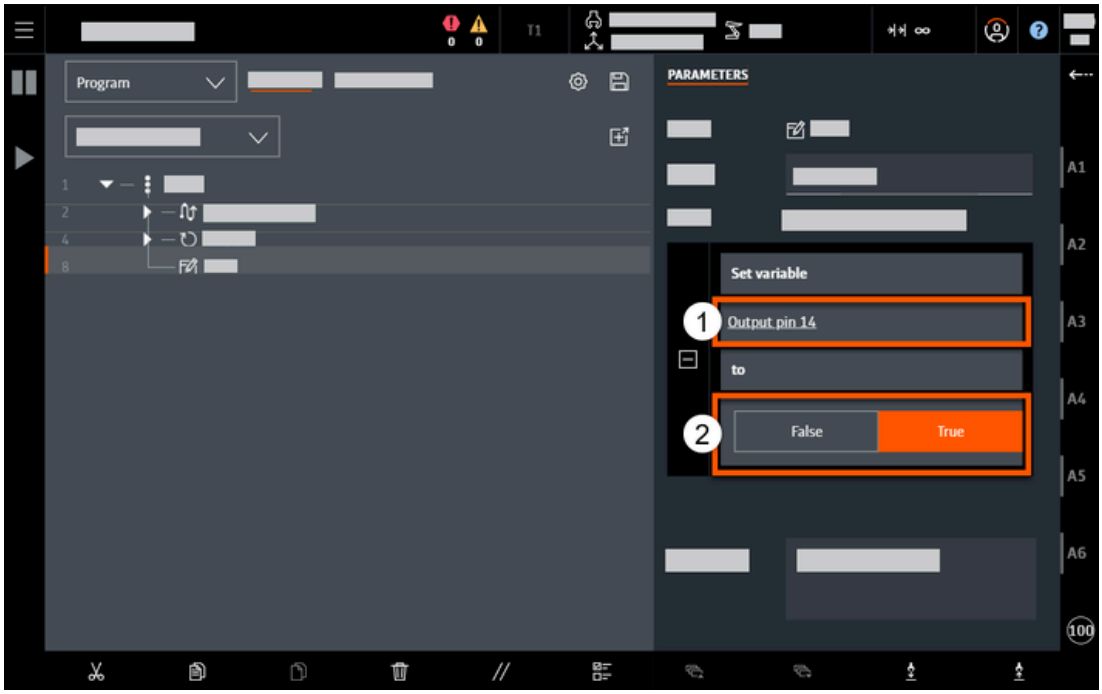
- **Variables**

  All remaining values that are not functions can be found under *Variables*.

Structure of Expressions

Expressions are composed of function calls, values, variables and some other constructs. There is exactly one constructor function for most data types.

Many functions accept 2 parameters. Functions can, however, also accept only one or multiple parameters.



Example of a function call

1	1st parameter: Variable reference
2	2nd parameter: Value of the variable reference

The values to be specified depend on the selected variable type. For example, if a variable reference of the *Boolean* data type is selected, only one Boolean value can be entered.

Functions as value

It is possible to treat functions as values. Functions proposed as "*Function as value*" can be transferred to other functions or saved as variables.

Namespace lang

Description

The **lang** namespace contains basic programming options.

Expressions

Call:

- Under Library, select **Functions** > **lang**.

Expression 1st level	Expression 2nd level	Description
Ref	get-id	Function accepts a *lang.Ref* and returns the value of the attribute "id".
? = ?		Function compares if 1st and 2nd parameters are equal.
? != ?		Function compares if 1st and 2nd parameters are unequal.
? or ?		Function checks whether 1st or 2nd parameter is true.
not(?)		Function negates a logical expression.
? and ?		Function checks whether the 1st and 2nd parameters are true.
if(cond: ?, then: ?, else: ?)		Conditional expression: Function accepts a Boolean value *cond* as well as the values *then* and *else*. If *cond* is true, the result is *then*. Otherwise, it is *else*.

lang.List data type

Description

The **lang.List** data type represents lists of any length.

Expressions

Call:

- Under Library, select **Functions** > **lang** > **List**.

Expression 1st level	Expression 2nd level	Description
Cons	get-head(?)	Function reads the header element of a non-empty list.
	get-tail(?)	Function reads the rest (all elements except header elements) of a non-empty list.
is-Nil?(?)		Function checks whether a list is an empty list.
? concat ?		Function appends the second list to the first.
get(value: ?, at: ?)		Function accepts a list value *value* and a number *at*. Function returns the list element at this index.
filter(pred: ?, value: ?)		Function returns a list containing only the *value* elements that are true when applied to *pred*.
map(func: ?, value: ?)		Function returns a new list where each element has been converted from *value* to *func* through application.
reverse(?)		Function returns a list consisting of the elements of *value* in reverse order.
set(value: ?, at: ?, to: ?)		Function returns a list. The element *at* is replaced by *to*.
length(?)		Function returns the number of characters or elements in this object.
is-Cons?(?)		Function checks whether a list is a non-empty list.

Constructors

Call:

- Under Library, select **Data** > **lang** > **List**.

Constructor	Description
Cons(head: ?, tail: ?)	Constructor generates a list from a header element and another list.
Nil	The constant *Nil* is an empty list.

Data type “lang.Maybe”

Description

The data type “**lang.Maybe**” represents optional values.

Expressions

Call:

- Under Library, select **Functions** > **lang** > **Maybe**.

Expression	Description
get-value(?)	Function returns the value of a *Maybe* if it was created via *Just*.
is-Nothing?(?)	Function checks whether a *Maybe* really is a *Nothing*. An optional value is missing.
is-Just?(?)	Function checks whether a *Maybe* really is a *Just*. An optional value is present.

Constructors

Call:

- Under Library, select **Data** > **lang** > **Maybe**.

Constructor	Description
Just(?)	Constructor creates a *Maybe* with a contained value.
Nothing	*Nothing* is the constant for a *Maybe* with no value in it.

Namespace math

Description

The namespace **math** contains mathematical operations and data types.

Expressions

Call:

- Under Library, select **Functions** > **math**.

Expression 1st level	Expression 2nd level	Description
Vec3	get-x(?)	Function supplies the X component of the 3 vector.
	get-y(?)	Function supplies the Y component of the 3 vector.
	from-mm(x: ?, y: ?, z: ?)	Function creates a vector (in m) from components in millimeters.
	get-z(?)	Function supplies the Z component of the 3 vector.
Vec6	get-x1(?) … get-x6(?)	Function supplies the X1 … X6 component of the 6 vector.
Quaternion	Rotation(rx: ?, ry: ?, rz: ?)	Generates a quaternion from three Euler angles in the KUKA convention. Here, the angles are specified in degrees.
	get-w(?) … get-z(?)	Function supplies the W, X, Y or Z component of the quaternion.
? + ?		Function adds the 1st and 2nd parameter.
cos(?)		Cosine function The argument is expected in radians.
? * ?		Function multiplies the 1st and 2nd parameter.
? >= ?		Function compares whether the 1st parameter is greater than the 2nd parameter or equal to the 2nd parameter.
? <= ?		Function compares whether the 1st parameter is smaller than the 2nd parameter or equal to the 2nd parameter.
?- ?		Function subtracts the 2nd parameter from the 1st.
? / ?		Function divides the 1st parameter by the 2nd.
neg(?)		Function negates the parameter.
deg-to-rad(?)		Function converts from degrees to radians.
sin(?)		Sine function The argument is expected in radians.
rad-to-deg(?)		Function converts from radians to degrees.
? < ?		Function compares whether the 1st parameter is smaller than the 2nd parameter.
? > ?		Function compares whether the 1st parameter is greater than the 2nd parameter.
? % ?		Function carries out integer division using the 1st and 2nd parameter and returns the remainder.

Constructors

Call:

- Under Library, select **Data** > **math**.

Constructor	Description
Rotation(rx: 0, ry: 0, rz: 0)	Constructor creates a rotation from three Euler angles. The resulting value is of type *math.Quaternion*.
Vec3(x: ?, y: ?, z: ?)	Constructor creates a 3 vector.
Vec6(x1: ?, x2: ?, x3: ?, x4: ?, x5: ?, x6: ?)	Constructor creates a 6 vector.

Programming of a compliant robot

Overview

Robots with integrated joint torque sensors can be moved under impedance control. Impedance control can be parameterized for a motion group (node **Motion group**) and for the motion command **Position hold**.

(>>> Holding position under impedance control)

The following control modes are available on the **Sensitivity** tab of the specified nodes:

- **Position Control** (default)

  Position control is preselected by default. In position control, the robot is not compliant and follows the programmed path as accurately as possible. Position control has no adjustable parameters.

- **Cartesian impedance**

  Cartesian impedance control can be used if the robot is to react in a compliant manner to external forces/torques, e.g. to obstacles on the programmed path or process forces.

  (>>> Cartesian impedance control)

- **Axis impedance**

  Axis impedance control can be used if one or more axes are to react in a compliant manner to external forces/torques.

  (>>> Axis impedance control)

The control mode selected in a **Motion group** node as well as its parameters apply to all motions of this motion group.

Cartesian impedance control

Description

The Cartesian impedance control 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. As a result, the robot reacts in a compliant manner to externally applied forces or torques.

(>>> Calculation of the forces on the basis of Hooke's law)

Compliance can also be used to apply Cartesian forces or torques to the robot in a targeted manner. These applied forces and torques overlay the forces and torques generated by the spring stiffness.

Cartesian impedance control always refers to the TCP (motion frame) set in **Motion group** node or **Position hold** node. The values for stiffness, damping and overlaid forces/torques can be set for each of the 6 degrees of freedom for this frame.

- Translational degrees of freedom: X, Y, Z
- Rotational degrees of freedom: Rx, Ry, Rz

Stiffness

The stiffness parameters determine how much the robot yields when a force is applied.

- If a low stiffness is set for a degree of freedom, the robot is very compliant in this direction. It reacts to obstacles and external forces by deviating from its path.
- Conversely, high stiffness leads to good path tracking with low compliance along the degree of freedom.

WARNING **Risk of unpredictable motions with zero stiffness**

Setting zero stiffness can cause the robot to move freely and uncontrollably in the corresponding directions. This particularly applies in the case of incorrect load data and joint torque sensors that are not properly mastered. Injuries or damage to property may result.

- Use zero stiffness values only in contact situations combined with external force application in order to achieve a controlled application of force.

Damping

The damping parameters determine how much the robot oscillates after a force has been applied.

- If high damping is set for a degree of freedom, the oscillation is reduced in this direction.

Force/Torque overlay

External Cartesian forces and torques can also be applied here. These overlay the Cartesian forces and torques that result from the spring stiffness.

⚠ WARNING Risk of injury and damage to property due to strong acceleration of the robot in the direction of force application

An application of force can result in strong acceleration and rapid motion of the robot in the corresponding direction. Injuries or damage to property may result.

- Only apply forces if the robot has already moved on contact in the corresponding direction.

Example

High stiffness is set in the Z direction of the TCP (motion frame) and low stiffness in the X/Y direction. In this way, the TCP can follow the programmed path well during a motion in the tool direction Z and at the same time swerve in the XY plane.

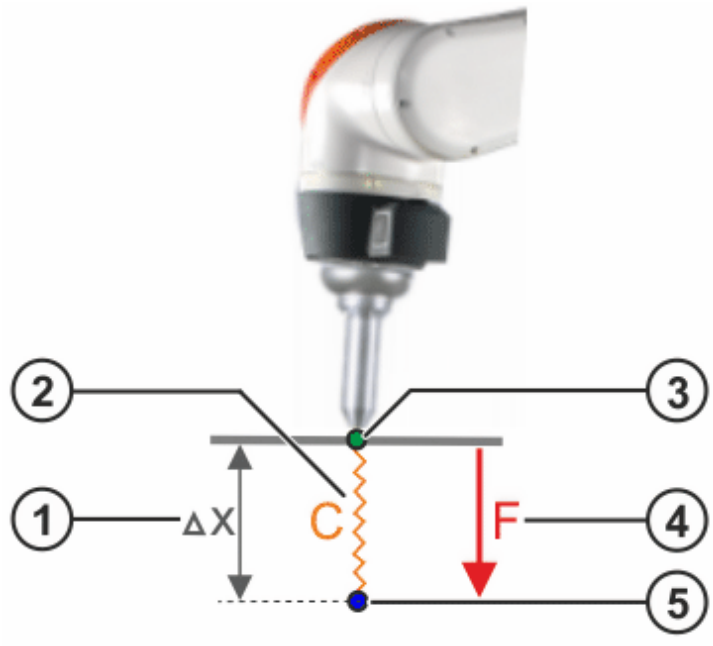
Calculation of the forces on the basis of Hooke's law

Description

If the measured and specified robot positions correspond, the virtual springs are slack. As the robot's behavior is compliant, an external force or a motion command results in a deviation between the setpoint and actual positions of the robot. This results in a deflection of the virtual springs, leading to a force in accordance with Hooke's law.

The resultant force F can be calculated on the basis of Hooke's law using the set spring stiffness C and the deflection Δx:

$F = C \cdot \Delta x$



Virtual spring with spring stiffness C

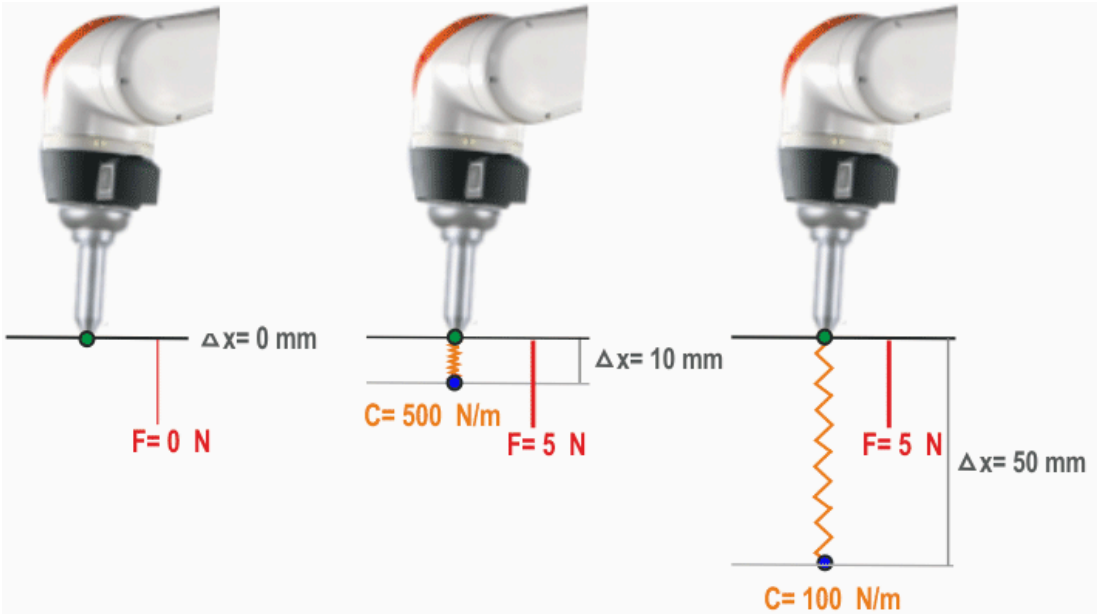
1	Deflection Δx
2	Virtual spring
3	Actual position
4	Resulting force F
5	Setpoint position

If the robot is at a resistance, it exerts the calculated force. If it is in free space, it moves to the setpoint position. On the way to the setpoint position, path deviations occur due to internal frictional forces in the joints. The extent of these deviations 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

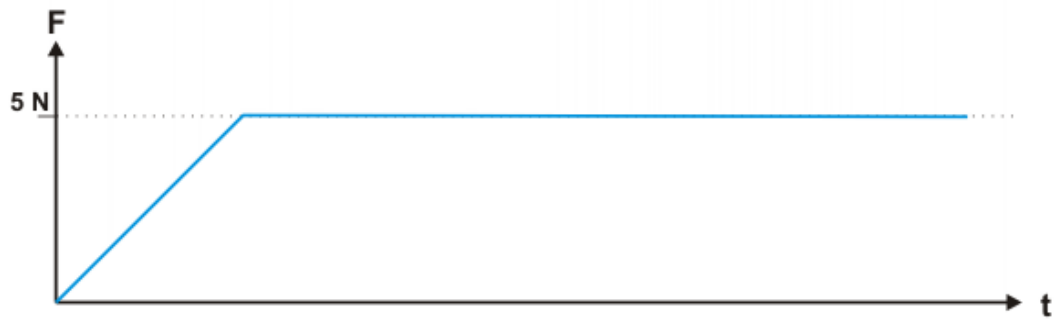
The force exerted at the contact point depends on the difference between the setpoint position and the actual position and the set stiffness.



Force exerted on contact

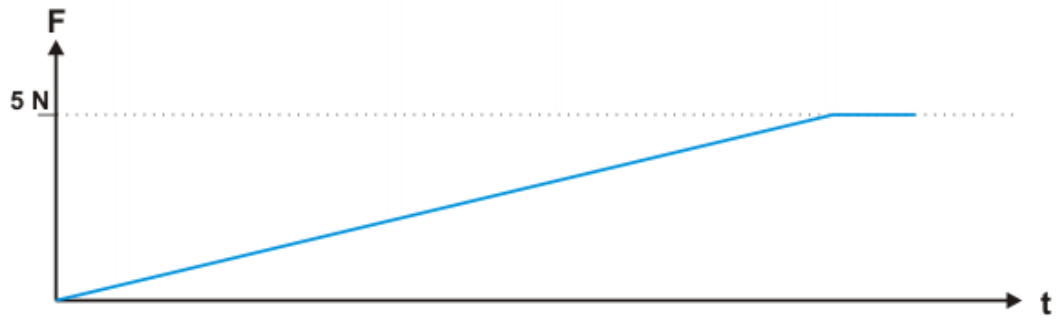
As shown in the figure (>>> Force exerted on contact), a large position difference and low stiffness can result in the same force as a smaller position difference and greater stiffness. If the force is increased by a motion in a contact situation, the time required to reach this force differs if the 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.



Force over time (high stiffness, small position difference)

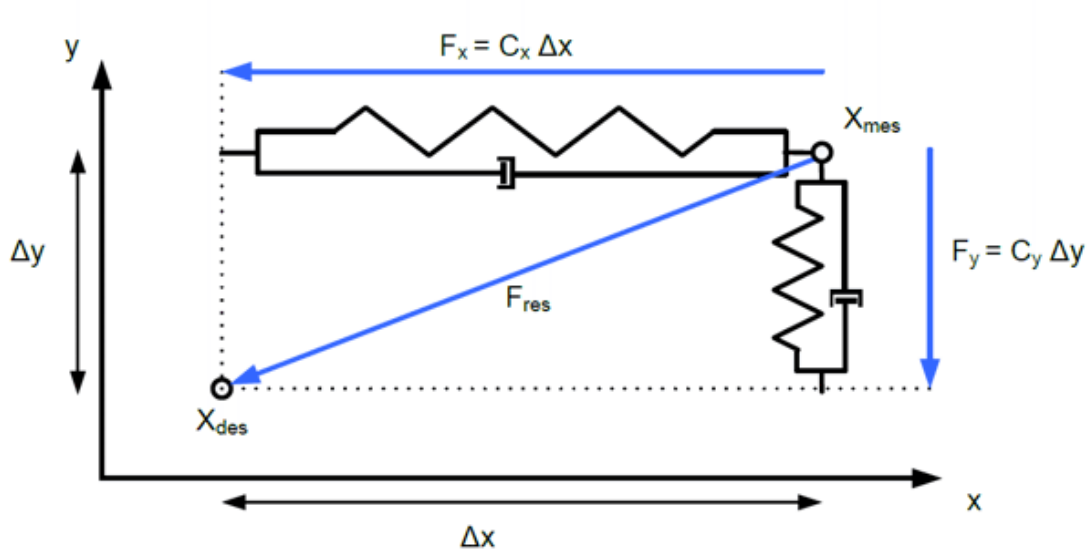
In the case of a large position difference and low stiffness, the force is built up more slowly. This can be used, for example, if the robot moves to the contact point and the impact loads are to be reduced.



Force over time (low stiffness, large position difference)

Setpoint/actual deviations in more than one direction lead to deflection of all the affected virtual springs. The magnitude and direction of the overall force results from vector addition of the individual forces for each direction.

The deflection in the X direction by Δx and in the Y direction by Δy result in force F_x in the X direction and F_y in the Y direction. The vector addition results in the overall force F_{res}.



Overall force in the case of deflection in 2 directions

Axis impedance control

Description

With axis impedance control, the following parameters can be set for each individual axis:

- **Stiffness**

  The stiffness parameter determines the degree of compliance of an axis when force is applied

  - If a low stiffness is set for an axis, the axis is very compliant and can be easily moved by hand.
  - Conversely, high stiffness for an axis results in low compliance.

  ⚠ WARNING **Risk of unpredictable motions with zero stiffness**

  Setting zero stiffness can cause the robot to move freely and uncontrollably in the corresponding directions. This particularly applies in the case of incorrect load data and joint torque sensors that are not properly mastered. Injuries or damage to property may result.

  - Use zero stiffness values only in contact situations combined with external force application in order to achieve a controlled application of force.

- **Damping**

The damping parameters determine how much an axis oscillates after a force has been applied

- If high damping is set for an axis, the oscillation is reduced.

Holding position under impedance control

Description

Using the **Position hold** motion command, the robot can hold its Cartesian setpoint position over a set period of time and remain under servo control.

This can be used in combination with corresponding settings for impedance control so that the robot exerts corresponding forces and/or torques at the current position. In addition, the impedance-controlled robot can respond in a compliant manner to external forces – thus enabling, for example, position corrections.

 During a **Position Hold**, the impedance-controlled robot can be brought away from its setpoint position by external forces. Whether and in which direction the robot moves from the setpoint position depends on the set controller parameters and the resulting forces.

 The **Position hold** motion command cannot be used in a motion group (**Motion group**) and combined with **Move** nodes.

Parameters

The following parameters can be defined on the **Parameters** tab:

- **TCP**

  The TCP defines the motion frame to which the parameters of the Cartesian impedance control refer (not relevant for position control or axis impedance control).

- **Duration**

  The following switches can be used to define how long the motion command **Position hold** is executed.

  - **Fixed**: The exact duration can be set (default: 10000 ms).
  - **Infinite**: If this switch is active, the duration is not limited.

    The command can only be aborted by means of a stop condition (trigger with action **Break**).

Triggers

In addition to the defined duration, a condition can be programmed on the **Triggers** tab in order to cancel the motion command.

(>>> Trigger programming)

Example

A robot is to move on contact and press against a workpiece for 5 seconds with a force of 15 N in the contact direction.

1. Move the robot into the contact position, e.g. by means of manual guidance or by a programmed motion used for moving on contact.

   Example of programming a motion to contact: (>>> Example)

2. Program the motion command **Position hold** for a duration of 5 seconds with Cartesian impedance control.
   - Maximum stiffness and damping in all directions, with the exception of the contact direction, zero stiffness there
   - External force application of 15 N in the contact direction

Notes on using impedance control

Setpoint-actual comparison

In a setpoint-actual comparison, the commanded position (setpoint) of the robot is compared to the actual position (actual) of the robot. This corrects a possible deviation of the setpoint position from the actual position in the case of compliant impedance control. This prevents unexpected jumps from occurring if the stiffness of the robot changes.

- If a change is made from a motion under impedance control to another motion (**Motion group** or **Position hold**), an automatic setpoint-actual comparison of the robot position is always carried out. Exception: The subsequent motion is an impedance-controlled motion with identical parameters.
- If no further motion follows, a setpoint-actual comparison is carried out when the brakes of the robot are applied. A simultaneous relaxation of any contact forces that may be present is also carried out.
- If an application is paused during the execution of a motion under impedance control, setpoint-actual comparison is carried out when the brakes are applied.

Repositioning

Setpoint-actual comparison while a motion is paused under impedance control has the effect that the application cannot be resumed until the robot has been repositioned using the Start key.

The motion back to the setpoint position is always carried out using the controller parameters of the paused motion.

 It is advisable to perform repositioning approximately from the position at which the motion was interrupted. If this is not observed, it may be that the motion is not resume from the position at which it was actually interrupted.

WARNING **Risk of injury and damage to property due to strong acceleration of the robot during repositioning with external force application**

If an impedance-controlled motion with external force application has been paused, repositioning with external force application also takes place. If the robot is in free space during repositioning, this can lead to strong accelerations of the robot. Injuries or damage to property may result.

- Manually bring the robot back in contact before repositioning.

Trigger programming

Description

Triggers can be programmed in the **Motion group** node and the **Position hold** node.

Triggers consist of a condition and an action. The action is executed if the condition is fulfilled.

The following condition types can be programmed on the **Triggers** tab of the above-mentioned nodes:

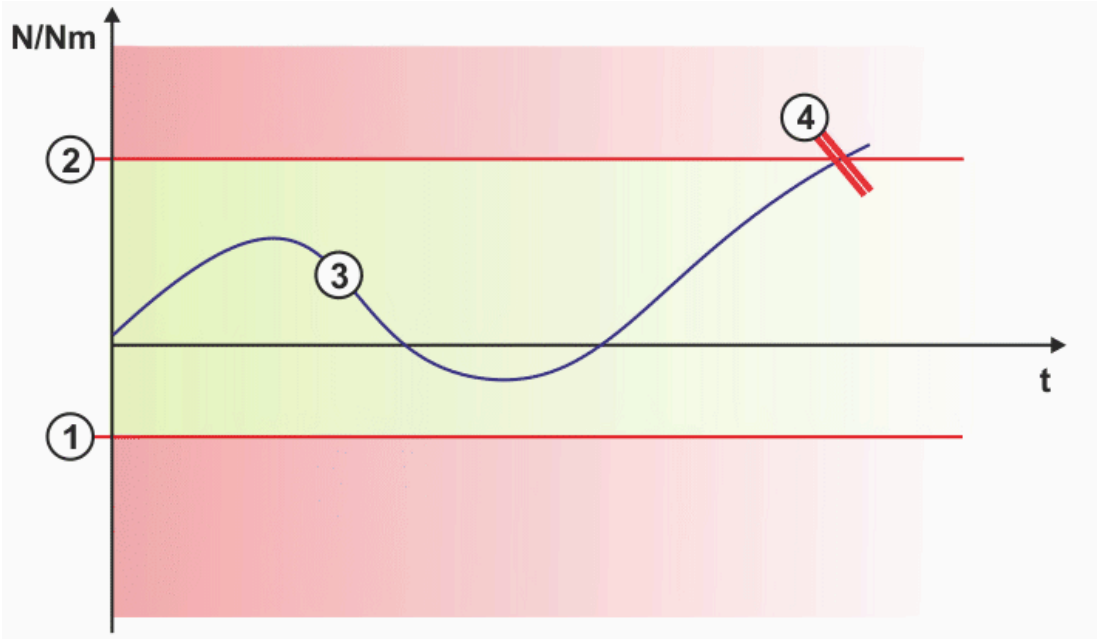
- Force component condition

  The condition is met if the external Cartesian force measured along the set axis of the set TCP lies outside a defined range (**Min**...**Max**).

- Torque component condition

  The condition is met if the external Cartesian torque measured about the set axis of the set TCP lies outside a defined range (**Min**...**Max**).

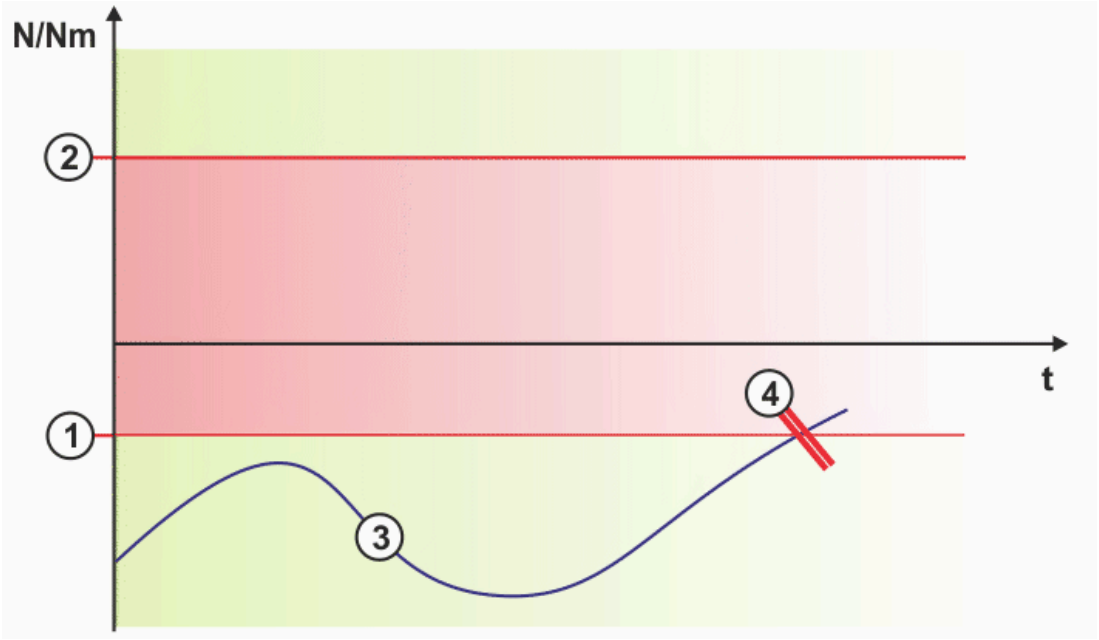
If the condition is met, the **Break** action is triggered by default. This action ends the current motion (**Motion group** or **Position hold**) and executes the next program command. Other actions cannot be programmed.



Trigger: Cancellation of motion due to force or torque

1	Minimum external force/minimum external torque
2	Maximum external force/maximum external torque
3	Measured external force/measured external torque
4	Action **Break** is triggered

Furthermore, the negation of a force component condition or torque component condition can be programmed. The negation of a condition is met if the measured external force or the measured external force torque is within a defined range (**Min**...**Max**).



Trigger: Cancellation of motion due to force or torque (negation “not”)

1	Minimum external force/minimum external torque
2	Maximum external force/maximum external torque
3	Measured external force/measured external torque
4	Action **Break** is triggered

Procedure

1. On the **Triggers** tab of the nodes **Motion group** or **Position hold**, press the **Add trigger** button.
2. Assign a name for the trigger.
3. Select the type of condition:
   - **Force Component**
   - **Torque Component**
   - **Not**

     Negation of a force component condition or torque component condition; see step 5.
4. Define the parameters of the condition:
   - **Measuring axis**

     Axis of the measurement frame at which the external force or torque is measured
   - **Measurement Frame**

     Reference frame for measuring the external force or torque
   - **Min**

     Minimum value for the external force or the external torque
   - **Max**

     Maximum value for the external force or the external torque
5. If the condition type **Not** is selected, proceed as follows:
   1. Select the following entry under **LIBRARY** in the selection menu of the Expression Editor:
      - **Data** >**Trigger** > **ForceComponent(…)** or **TorqueComponent(…)**
   2. In order to be able to define the parameters of the condition, the expression must be opened in full-screen mode. To do so, select the expression and select the corresponding entry in the context menu to the right.

Example

Move robot on contact

As soon as contact is established and a defined force is measured, a trigger is triggered and the motion is stopped.

1. Bring the robot into the pre-position for the expected contact.
2. Create the **Motion group** node.
3. Program the search run as a relative motion (LIN) in the direction of the expected contact.
   1. Select TCP.
   2. Select tool direction Z.
4. Configure Cartesian impedance control on the **Sensitivity** tab of the **Motion group** node.
   1. Set maximum stiffness and damping in all directions.
   2. Set stiffness to medium stiffness in search direction (1000 N/m).
5. On the **Triggers** tab, add a trigger with the condition **Force Component** to the search run.
   - Select measuring axis **Z** to measure the force along the Z axis of the TCP.
   - Set minimum force to-10 N.

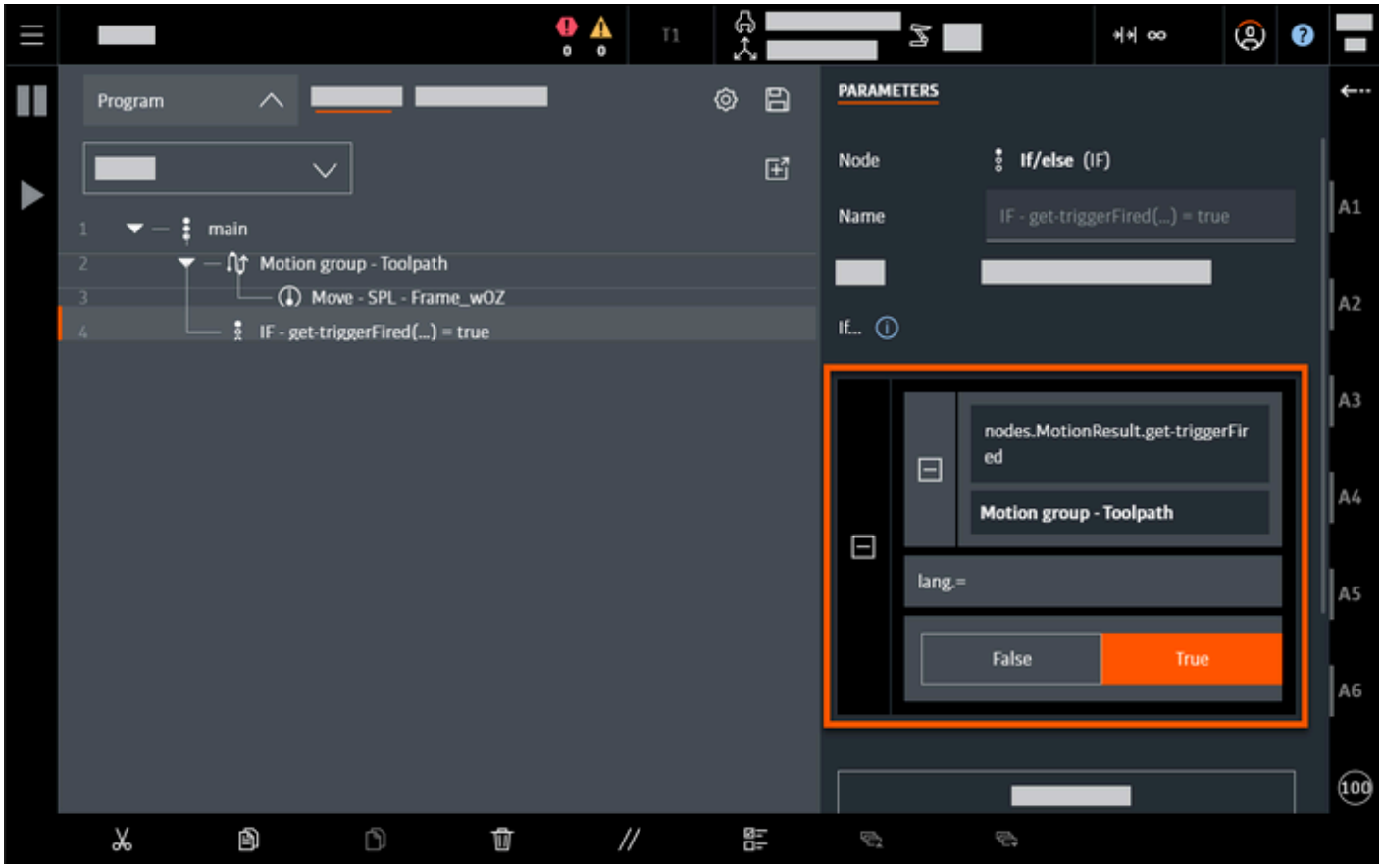
- Set maximum force to high value, e.g. to 100000 N.

If, in the search direction, a force of <-10 N is measured during the search run, the trigger is triggered and the motion stops. The measured contact forces act against the direction of travel.

Checking whether trigger is triggered

Description

Whether a trigger programmed for a motion group has been triggered can be checked as follows.



Checking whether trigger is triggered

Procedure

1. Create and select a **If/else** node in the motion group.
2. Set the 1st operator in the **If/else** node:
    1. Select 1st value: *Functions* > *nodes* > MOTIONRESULT *get-triggerFired(?)*
    2. Select 2nd value: *Variables* > RETURN VARIABLES

The return variable has the same name as the motion group.

3. Compare the result of the trigger with the desired value:

Operand *lang.*= is preset.

- Set the 2nd operator to true/false in the **If/else** node.

The result is a Boolean value:

- True: Trigger is triggered.
- False: Trigger is not triggered.

Grid Patterns toolbox

Overview

The Grid Patterns toolbox is pre-installed on the robot controller. Grid patterns make it possible to move the robot by means of a programmed pattern.

Grid patterns are defined in the scene editor and in the capabilities. Grid patterns can be integrated into programs.

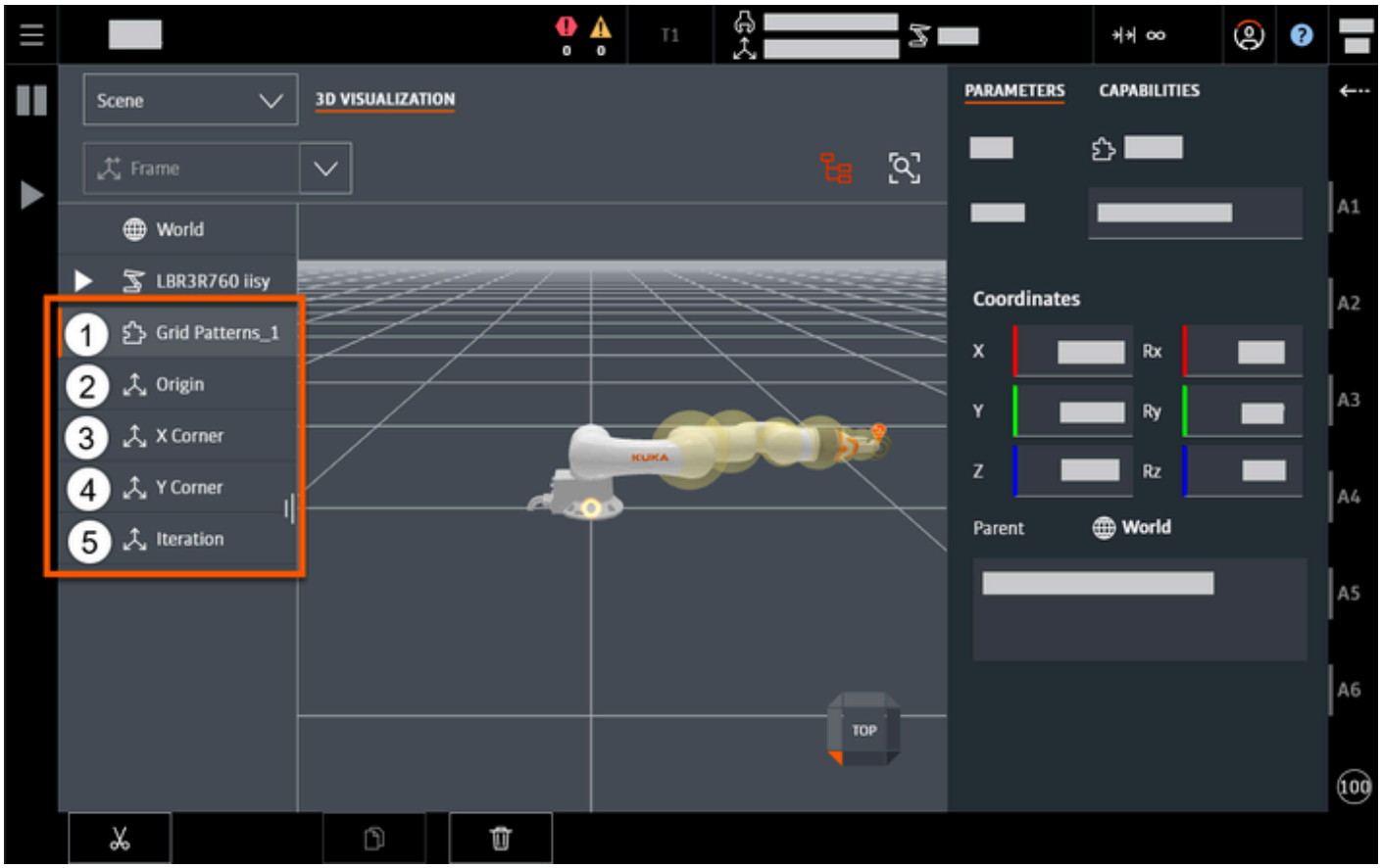
Defining grid patterns in scene

Procedure

1. In the Feature menu, select **Scene**.
2. In the object selection menu, select **Devices** > **Grid Patterns**.

    A new grid pattern can only be created under **World**.
3. Show the object tree and check whether the grid pattern has been created.
4. Rename grid pattern.
5. Manually enter coordinates in the parameter view.
6. In the object tree, select **World**.
7. Select **Frame** in the object selection menu. The first frame is added.

   Overall, the following frames must be created:
   - Origin
   - Corner X
   - Corner Y
   - Iteration
8. Specify frame accordingly.
9. Move the robot to the desired position by jogging or manual guidance and record the frame via **Touch up**.



Grid patterns in scene- Example

1	Grid Patterns
2	Origin This frame defines the gripper orientation at the removal points. By default, the cell index is 0. The cell index can be changed manually in the Feature menu **Capabilities** in the Parameter view. (>>> Defining grid patterns in capabilities)
3	Corner X This frame defines the direction of the first iteration.
4	Corner Y This frame defines the direction of the second iteration.
5	Iteration This frame can be generated at any position.

Defining grid patterns in capabilities

Precondition

- At least one grid pattern has already been created in the scene editor.

Procedure

1. In the Feature menu, select **Setup** > **Capabilities**.
2. Select the newly created grid pattern in the drop-down menu.
3. Open the grid pattern and press **Parameters**.
4. Enter the X and Y values in the parameter view.


The X and Y values correspond to the array size.

5. Select parameters for the traversal order.

   There are 2 options to choose from:

   - ZIG_ZAG
   - SNAKE

6. Select the frame references for the respective frames Origin, Corner X, Corner Y and Iteration:

   - **New project frame**
   - **Existing project frame...**

7. If **New project frame** is selected:

   - Frame is created at the current position.

8. If **Existing project frame...** is selected:

   1. A dialog is opened. Show object tree.
   2. Select frame.
   3. Press **Select target**. The assignment is applied.

9. Optional: Change index.

   The index is preset to 0 by default.

10. Click on **Apply** to save the change.

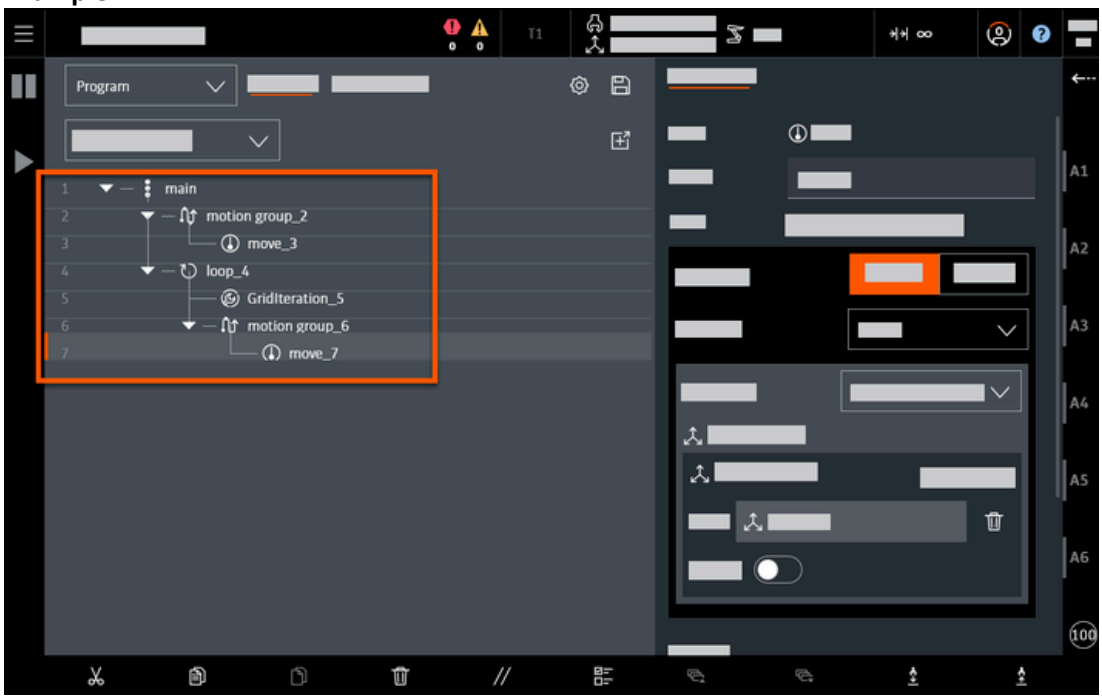
Using grid patterns in a program

Procedure

1. In the Feature menu, select **Program**.
2. Create a new program.
3. Open the node palette and select **Motion group**.
4. Add the **Loop** node.
5. Under the **Loop** node, add the **GridIteration** node.
6. Under the **Loop** node, add a **Move** node.
7. Configuring the first node **Motion group**:
   1. Select the **Motion group** node.
   2. Rename **Motion group**.
   3. Select the motion type **Axis (PTP)**.
   4. Set the motion as required.
8. Configuring the **Loop** node:
   1. Select the **Loop** node.
   2. Rename **Loop**.
   3. Select the loop type **Infinite**.
9. Configuring the **GridIteration** node:
   1. Select the **GridIteration** node.
   2. Select the available grid patterns under Capability.
10. Configuring the **Move** node under **Loop** node:

1. Select the **Move** node.
2. Select the **Linear** motion type.
3. Select the **Frame Reference target** target type. A dialog is opened.
4. Select **Use existing project frame** and confirm the selection.
5. Show the object tree and select the iteration frame.
6. Press **Select target**.

Example



Using grid patterns in a program – Example

Nodes

The following nodes are available in the node palette:

Node	Description
GridIteration	Automatically selects the available grid pattern capability. If more than one grid pattern capability is available, the desired grid pattern can be selected in the node.
GridReset	Resets the grid pattern