

TOGETHER IN ROBOTICS

LINK TOOLKIT PLUGIN

Contents

Copyright notice.....	1
Disclaimers.....	2
Plugins.....	3
Link Toolkit Plugin.....	3
Getting the plugins for the robot.....	3
Installing plugins.....	6
Deactivating plugins.....	9
Activating plugins.....	10
Updating plugins.....	10
Overview of working with plugins.....	13
Configuration of the Link Toolkit Plugin.....	15
Configuring events.....	15
Events, reactions, and properties.....	16
Link Toolkit actions.....	19
Path Player action.....	20
Parameters of the Path Player action.....	20
Creating a path.....	22
Python Script action.....	23
Parameters of the Python Script action.....	24
Creating a script directly in the Python text area.....	25
Using a script that is already prepared.....	26
Seek action.....	27
Parameters of the Seek action.....	27
Variables.....	30
Python Script data as a variable.....	30
Seek data as a variable.....	31
Programming manual.....	33

Introduction to the Link Toolkit programming guide.....	33
Python module.....	33
Syntax.....	34
Comment.....	34
Indentation.....	35
Variable.....	36
Variable management.....	38
get_variable.....	38
set_variable.....	39
IO Access.....	40
config_analog_input_to_current.....	40
config_analog_input_to_voltage.....	41
config_analog_output_to_current.....	41
config_analog_output_to_voltage.....	41
config_wrist_analog_input_to_current.....	42
config_wrist_analog_input_to_voltage.....	42
config_wrist_analog_output_to_current.....	43
config_wrist_analog_output_to_voltage.....	43
config_wrist_channel_to_digital_input.....	44
config_wrist_channel_to_digital_output.....	44
get_analog_input_value.....	44
get_analog_output_value.....	45
get_digital_input.....	46
get_digital_inputs.....	47
get_digital_output.....	48
get_digital_outputs.....	48
get_safety_digital_input.....	50
get_safety_digital_inputs.....	50
get_safety_digital_output.....	51
get_safety_digital_outputs.....	52
get_wrist_analog_input_value.....	52
get_wrist_analog_output_value.....	53
get_wrist_digital_input.....	54
get_wrist_digital_output.....	54
set_analog_output_value.....	55
set_digital_output.....	56
set_digital_outputs.....	57
set_wrist_analog_output_value.....	58

set_wrist_digital_output.....	60
Mathematics.....	60
align_z_axis.....	60
binary_list_to_int.....	61
express_vector_in_frame.....	62
express_wrench_in_frame.....	62
int_to_binary_list.....	63
interpolate_pose.....	64
point_dist.....	65
pose_inv.....	66
pose_transform.....	66
wrench_trans.....	67
Robot.....	68
forward_kin.....	68
get_feedback.....	68
get_flange_pose.....	72
get_gravity.....	72
get_joint_currents.....	72
get_joint_positions.....	73
get_joint_speeds.....	73
get_joint_torques.....	73
get_payload.....	74
get_tcp_offset.....	74
get_tcp_pose.....	74
get_tcp_twist.....	75
get_tool_imu.....	75
get_wrench_in_base_frame.....	76
get_wrench_in_flange_frame.....	76
get_wrench_in_tcp_frame.....	76
inverse_kin.....	77
is_robot_standstill.....	78
log_event.....	78
stop_program.....	79
wait.....	80
USB storage.....	80
External communication.....	81
tcp_socket.....	81
xmlrpc.....	82

modbus.....	82
Paho MQTT.....	83
Changes since the last release.....	84
Changes made in version 1.1.....	84
Glossary.....	85
C.....	85
E.....	85
F.....	85
G.....	85
N.....	86
P.....	86
R.....	86
T.....	87
U.....	87
W.....	87

Copyright notice

The information in the Link Toolkit Plugin user guide contains proprietary information and, as such, no part may be reproduced, in part or in whole, or transmitted in any form or by any means, electronically or mechanically, including photocopying, recording, or any information or storage retrieval system, without prior permission in writing from Kinova.

Disclaimers

Kinova[®] and the Kinova logo are registered trademarks of Kinova inc., herein referred to as Kinova.

Kortex[™] is a trademark of Kinova inc.

All other brand and product names are trademarks or registered trademarks of their respective owners.

The Link Toolkit Plugin is an evolving product. Kinova continually works towards improving the reliability and performance of its products. Kinova does its best to ensure the content of the Link Toolkit Plugin user guide is accurate and precise with respect to the latest version of the robot. However, Kinova does not take responsibility for any erroneous, inaccurate, or missing information.



Note: Colors of the user interface in the guide may appear differently than in the actual user interface in an attempt to increase readability.



Note: The user guide may be updated from time to time. Please make sure to look at the latest version on our website.

The version of the document is in the header of the pages of the document.

Any questions or comments concerning this document, the information it contains or the product it describes may be addressed through support@kinova.ca.

Plugins

Link 6 is designed modularly, which makes it easy to expand on its capabilities. The extensions are added on as [plugins](#).

All installed plugins start to run when the controller is switched on. However, they can be stopped so the settings can be modified or deactivated.

The Link Toolkit plugin is not pre-installed on Link 6. It must be downloaded, installed, and activated to be able to use it.

Many plugins contain actions that the user can use when programming the robot with the visual programming interface in Kortex Web App. Each plugin may also contain parameters that can be modified in Kortex Web App from the corresponding plugin page. **Deactivate** the plugin before modifying the plugin parameters.



Important: When a USB device is connected to the controller while the plugin is already running, the plugin must be restarted so that it can access the USB device.

All plugins have two pages: **Configurations** and **About**.

Link Toolkit Plugin

Install the plugin when you need Link 6 to create a program as quickly as possible, using knowledge about the task.

Using the Toolkit plugin shortens the amount of time any given robot is down for installing and testing new programs.

The plugin contains different configuration roles.

- Configure the actions that the robot can perform for visual programming.
- Configure the robot behavior using the action scripts.
- Configure the robot to move in one direction until a specific condition is met.

Getting the plugins for the robot

Upload and installation must be done from the teach pendant and a [USB storage device](#) connected to the controller or from Kortex Web App running on a computer that has access to the files.

Before you begin

All plugins have dependencies on different aspects of the robot.

Make sure to verify all dependencies before installing the plugin.

About this task

The plugin installation file is named `link_toolkit_<version>.kp`, where `<version>` is the version of the plugin. Contact Kinova to get the appropriate plugin file if you are unsure which files you need.

A computer or [USB storage device](#) is needed to upload and install the plugin installation file.



Note: Plugins are updated from time to time. Always make sure you have the latest plugin for your installation.



Important: If the plugin is not compatible with the currently running version of Link 6, an error dialog launches.

Procedure

1. Plug the USB storage device on the computer.
2. Download the plugin installation file `link_toolkit_<version>.kp`.
3. Save the plugin installation file in the root directory of the USB storage device.
4. Eject the USB storage device after the transfer of the plugin installation file is complete.
5. Plug the USB storage device on the [controller](#).



6. Switch on the *teach pendant*.



7. Enter the administrator credentials.

KINOVA

Kortex™ Web App

Username

admin

Password

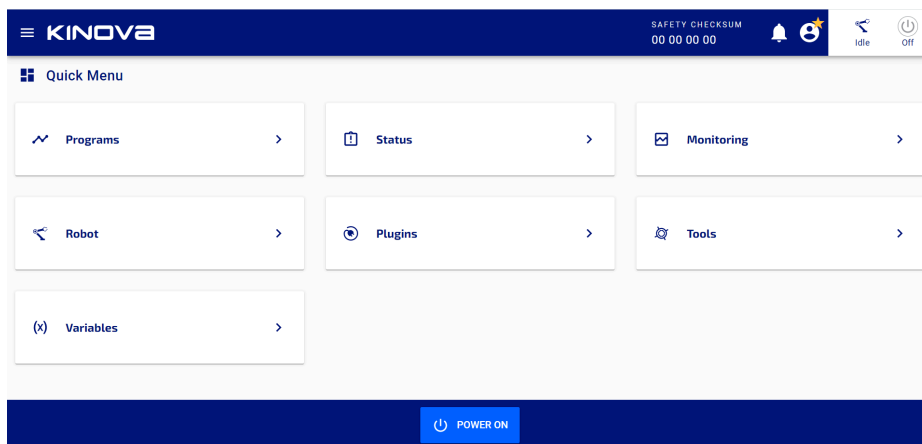
.....

👁

☐ MAINTENANCE MODE

CONNECT

8. Tap **CONNECT**.



9. Tap **POWER ON**.

The arm is switched on.

What to do next

Install the required plugins from the **Plugins** page in [Kortex Web App](#).

Related topics

[Installing plugins](#) on page 6

Installing plugins

Before you begin

The plugin files must be on a [USB storage device](#).



Important: The USB must be plugged into the controller when the plugin is being installed on Link 6. Refer to the documentation of other robots for the location of USB connections.

About this task

Access all installed plugins by tapping **Systems > Plugins**.

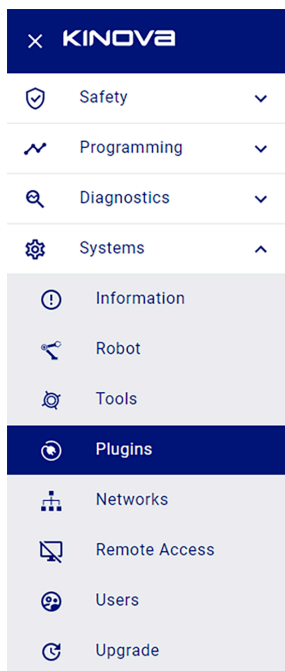
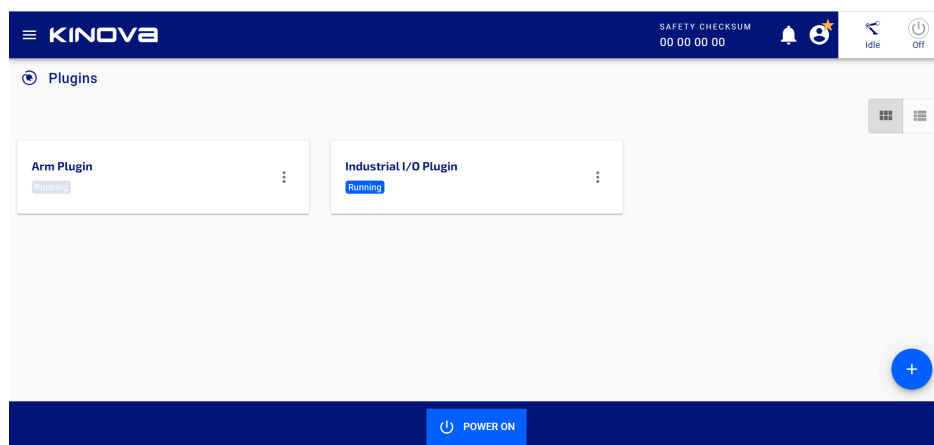


Figure 1: Accessing the plugins

Procedure

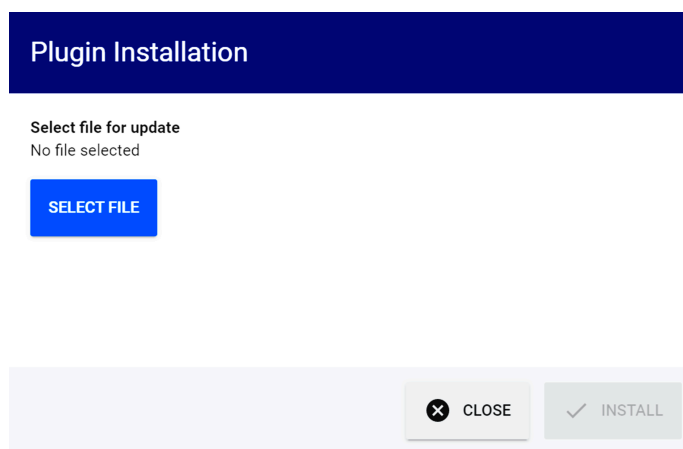
1. Tap **Systems > Plugins**.

The **Plugins** page launches.



2. Tap **+**.

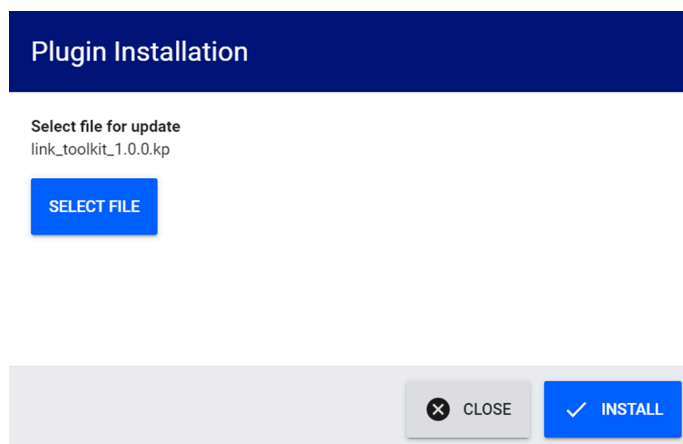
A **Plugin Installation** dialog launches.



3. Tap **SELECT FILE**.
4. Optional: Browse to the plugin installation files.
5. Select a plugin installation file.

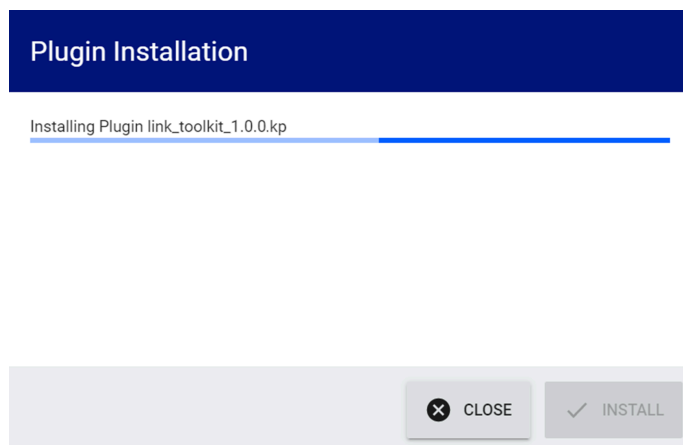
6. Tap **Open**.

The **Plugin Installation** dialog shows the name of the selected file.

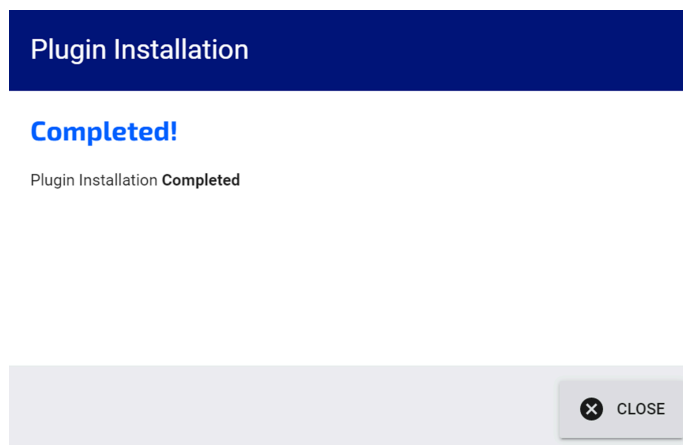


7. Tap **INSTALL**.

The **Plugin Installation** dialog shows that the plugin installation is in progress.

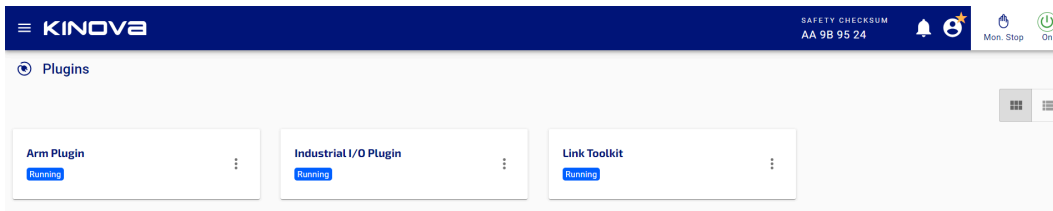


8. Tap **CLOSE** when installation is complete.



Results

The plugin is displayed on the **Plugins** page and is running.



Related topics

[Overview of working with plugins](#) on page 13

[Link Toolkit actions](#) on page 19

[Variables](#) on page 30

Deactivating plugins

When the actions associated with a plugin are not needed in the program, deactivate the plugin without uninstalling it.

Before you begin

The plugin must be installed on the robot and must be active.

About this task

The actions associated with Link Toolkit Plugin are available on the **Programs** page.

Procedure

1. Tap **Link Toolkit Plugin** on the **Plugins** page.
 - The **Configurations** page of the **Link Toolkit Plugin** launches.
 - **Link Toolkit Plugin** actions appear under the **Action** menu on the **Program** page.
2. Tap **Inactive** to deactivate the plugin.

Results

- The toggle is white and is closest to **Inactive**.
- **Stop** is not available to stop the plugin.



- The **Link Toolkit Plugin** button on the **Plugins** page changes its appearance to indicate that the plugin is not active.
- **Link Toolkit Plugin** does not appear in the Plugins **Action** menu on the **Program** page.

Activating plugins

When a plugin is installed, but does not appear in the **Action** menu on the **Program** page, the plugin is deactivated. Activate the plugin to use the actions of the plugin.

Before you begin

The plugin must be installed on the robot. The plugin must be deactivated.

About this task

The actions associated with Link Toolkit Plugin are available on the **Programs** page only when the plugin is activated.

Procedure

1. Tap **Link Toolkit Plugin** on the **Plugins** page.
 - The **Configurations** page of the **Link Toolkit Plugin** launches.
 - No **Link Toolkit Plugin** actions are in the **Action** menu on the **Action Program** page.
2. Tap **Active** to activate the plugin.

Results

- **Active** is blue.
- **Stop** is available to stop the plugin before deactivating it.



- The status of the **Link Toolkit Plugin** changes to **Running** on the **Plugins** page.
- **Link Toolkit Plugin** actions appear under **Action** on the **Action Program** page.

What to do next

Create a program with Link Toolkit actions.

Updating plugins

Integrate improvements to the plugins by updating them.

Before you begin

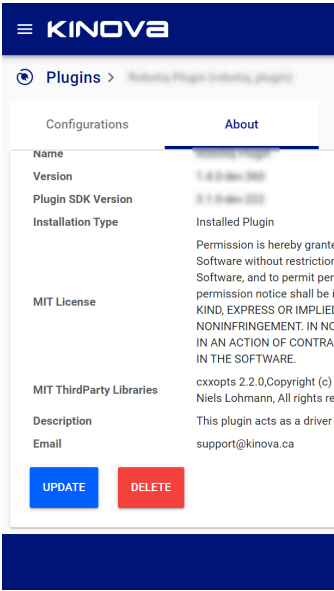
Download the latest version of the plugin. Contact Kinova to get the appropriate plugin file if you are unsure which files you need.

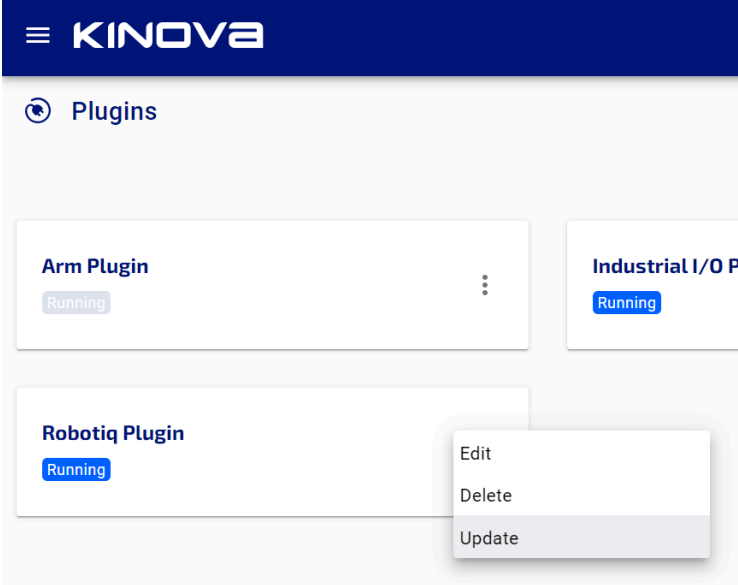
About this task

Use a computer or a [teach pendant](#). Update plugins directly from the **Plugins** page or from the **About** page of the plugin.

Procedure

1. Get the latest version of the plugin.
2. Tap **Systems > Plugins**.
3. Tap the plugin that you want to update.
4. Access the update dialog in one of two ways.

Option	Description
Update from the About page of the plugin	a. Tap About. The About page launches. 
Update from the Plugins page	a. Navigate to the Plugins page.

Option	Description
	b. Tap the <i>kebab menu</i> () of the plugin that you want to update. 

5. Tap **Update**.
- The **Plugin Update** dialog launches.

Plugin Update

Selected file

No file selected

SELECT FILE

☒ Options

☒ Keep configuration

☒ Keep data

☒ Keep tools

☒ Keep variables



CLOSE

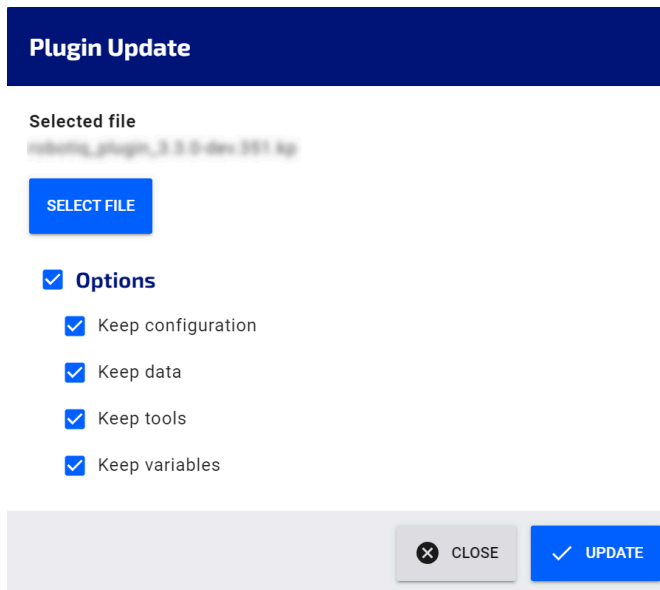


UPDATE

6. Tap **SELECT FILE**.

7. Select the plugin file.

The name of the file is displayed as the **Selected file**.



8. Tap **Options**.

By default, all data is selected. Deselect any information you want to ignore. Data that can be saved include the configuration, data, tools, and variables.

9. Select the type of data you want to keep from the current plugin.

10. Tap **UPDATE**.

The **Plugin Update** indicates that the plugin is updated.



11. Tap **CLOSE**.

The **Plugin Update** dialog closes.

Related topics

[Getting the plugins for the robot](#) on page 3

Overview of working with plugins

Tapping a plugin from the **Plugins** page launches the page of that plugin.

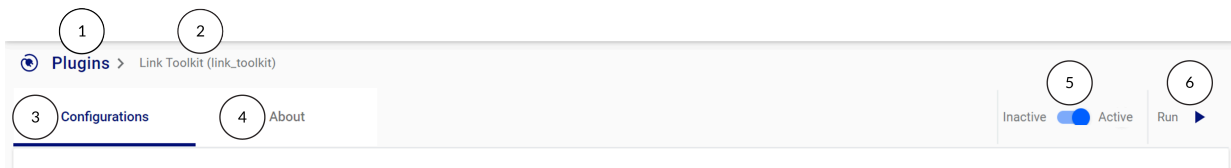


Figure 2: Example of a plugin page of a plugin that is stopped

- 1 Tap **Plugins** to return to the list of plugins.
- 2 Name of the selected plugin with the name of the *namespace* associated with the selected plugin in parentheses.
- 3 Tap **Configurations** to view and modify the current configurations of the selected plugin.
 - ! **Important:** Tap **Stop** (Stop ■) before you attempt to modify the current configurations.
 - ! **Important:** Tap **Start** (Start ►) after you modify the configurations. The plugin runs with the latest configuration values.

When there are no modifiable configurations in the plugin, a message is displayed.

This plugin does not expose any configuration.

- 4 Tap **About** to view important information about the selected plugin.
 - Name
 - Version
 - Plugin SDK Version
 - Installation Type
 - Licenses
 - Libraries
 - Description
 - Email
- 5 Toggle **Active** to deactivate plugins that need to be installed but are not being used. When the toggle is white, it is off.

Tap **Active** to activate a deactivated plugin. When the toggle is blue, it is on.
- 6 The current operating state of the plugin.

Tap **Stop** (Stop ■) to stop the plugin from running. The button changes to **Start** (Start ►).

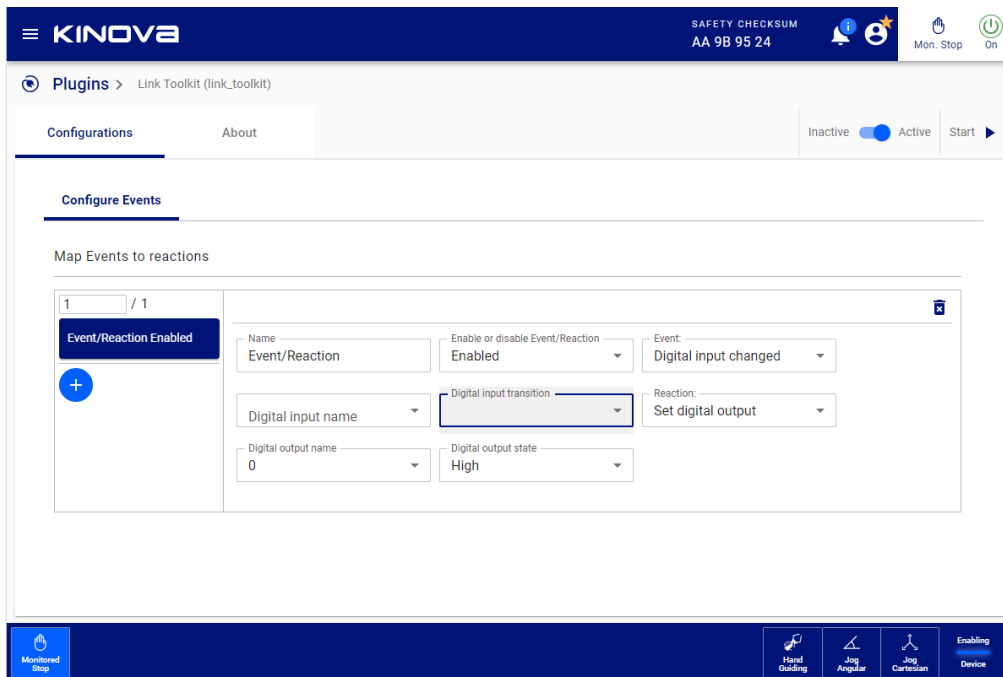
Tap **Start** to restart the plugin after modifying its settings. The button changes to **Stop**.

Configuration of the Link Toolkit Plugin

Create and configure events from the **Configurations** page of the Link Toolkit Plugin.

To access the **Configurations** of the Link Toolkit Plugin plugin, tap **Systems > Plugins > Link Toolkit**.

Figure 3: Configurations page of the Link Toolkit Plugin



Each configurable [event](#) must be associated with a [reaction](#). Each event-reaction pair needs be created with logical transitions. Each event-reaction pair has its own set of additional parameters that require configuration to be considered to be complete.



Remember: The configured event-reaction pair can work only when no one is controlling the arm with Kortex Web App.

Configuring events

Use events to control the arm without being logged into Kortex Web App.

About this task

To configure events, you must define the [reaction](#) and associate properties.



Note:

Procedure

1. Tap **Stop** (Stop ) before you attempt to modify the current configurations.
2. Tap .
3. Enter a **Name**.
4. Set the event as **Enabled** or **Disabled**.
5. Select an event.
6. Select a reaction.
7. Select parameters associated with the event-reaction pair.
8. Tap **Start** (Start ) after you modify the configurations. The plugin runs with the latest configuration values.

Events, reactions, and properties

Each event must have a reaction. Each event-reaction pair may have additional properties.

Events and reactions

There is one set of events available for configuration. There is one set of reactions available for configuration. Make sure the event-reaction pair makes sense.

Table 1: Events and reactions

Events	Reactions
Program status changed	Set program status
Wrist button pressed	Set digital output
Digital input changed	Set wrist digital output
Arm power changed	Set power
Fault status changed	Run a python script
	Reset fault
	Exit recovery

Properties

Events have properties. Reactions also have properties. Some properties also have properties.

Table 2: Events and their properties

Event	Properties	Sub properties	Description
Program status changed	Program status	Running	The program is started or resumed by tapping Play (▶).
		Not running	Program is completed or failed.
		Paused	Program is paused by tapping Pause .
		Stopped	Program is stopped by tapping Stop .
Wrist button pressed	Wrist buttons	+	The + button on the wrist is pressed.
		-	The - button on the wrist is pressed.
Digital input changed	Digital input name	0 to 7	Digital input names represent each of the terminal points on the controller for digital input.
	Digital input transition	Rising Edge	The digital input is transitioning from a low to a high state.
		Falling Edge	The digital input is transitioning from a high to a low state.
Arm power changed	Power	On	The arm is switched on.
		Off	The arm is switched off.
Fault status changed	Fault Status	Fault	The robot is in a fault state.
		No fault	The robot is recovering from a fault state.

Table 3: Reactions and their properties - rebuilt

Reaction	Property	Sub property	Other	Description
Set program status	Program status	Start	Program name (Id)	Name or identifier of the current program
			Operating mode	Program is running in Hold to Run or Automatic mode.
			- Hold to Run - Auto	
		Resume		The paused program starts where it left off as if Play (▶) is tapped.
		Pause		The program is paused as if Pause is tapped.

Reaction	Property	Sub property	Other	Description
		Stop		The program is stopped as if Stop is tapped.
Set digital output	Digital output name	0 to 7		Digital output names represent each of the terminal points on the controller for digital output.
Set wrist digital output	Digital output identifier	0 to 7		Wrist digital output names represent each of the terminal points on the controller for wrist digital output.
	Digital output state	High		The state of the wrist digital output is high.
		Low		The state of the wrist digital output is low.
Set power	Power	On		The arm is on.
		Off		The arm is off.
Run a python script	Python text area			Python commands written in the Python text area are performed.
Reset fault				The robot is reset after being in a fault state.

Link Toolkit actions

The Link Toolkit Plugin has its own set of actions available for further configuration when programming Link 6.

All actions associated with the Link Toolkit Plugin are available as tiles that can be dragged into the **Sequence editor** pane of the current program.

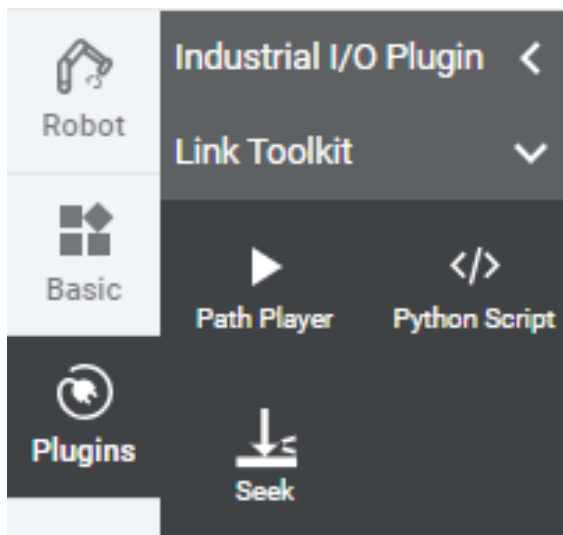
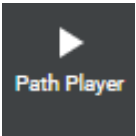
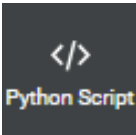


Figure 4: Actions available from the Link Toolkit Plugin

Table 4: Parameters of Link Toolkit plugin actions

Action	Icon	Parameter
Path Player		Records a granular trajectory that can be played in a program.
Python Script		Accepts a Python script that uses the Script library functions. Write the script in the Tile configuration pane or associate a Python script with the action.
Seek		Moves the robot in one direction until the robot comes into contact with an object or surface.

Path Player action

Record trajectories and replay them with a scaled time frame.

The first step in using the **Path Player** action is to record a trajectory. Recording begins from the moment you tap **RECORD**. Move the arm in Hand Guiding or Hold to Run mode. When working in Hold to Run mode, use the **Jog Angular** and **Jog Cartesian** buttons. All hesitations and movement corrections are saved in the recording.



Important: Do not use the **Path Player** action in a program that is intended to replicate trajectories that require a steady pace, such as welding trajectories. Use the action for complex assemblies that require more granular control over a trajectory.

When you drag the **Path Player** tile into the **Sequence editor** pane, the **Tile configuration** pane expands to expose all associated parameters. Use the controls to help define the granular position of the arm.

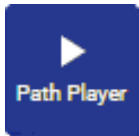


Figure 5: The Path Player expands in the Sequence editor

Figure 6: The Tile configuration pane when Path Player is the current tile

Parameters of the Path Player action

Each instance of the **Path Player** is configurable.

Table 5: Path Player parameters

Item	Description
Path Recording	Pane contains 2 controls for recording a path.

Item	Description
	Tap to record the movement of the arm.  Tip: The recording is time-based. The time taken between pressing RECORD and moving the arm is saved in the recording. Every hesitation while moving the arm is also recorded.
	Tap to stop recording the movement of the arm.
Move to position	Move the arm to one of the recorded positions.
	Tap to move the arm to the first recorded position.
	Tap to stop the arm when it is transitioning from start to end, or end to start.
	Tap to move the arm to the last recorded position.
Reference Frame	Select a frame of reference. The pose converts automatically with respect to the frame of reference. • Base Frame: the trajectory begins at the position relative to the base • Current position: the trajectory begins at the current position of the arm  Important: If the arm is moved after the recording, the trajectory replay starts from where it is at the time. • Custom Frame: the trajectory begins at the position relative to the custom frame
Custom Frame	Enter the global variable that contains the pose of the arm.
Speed Mode	Select the type of speed at which the arm should move. • Factor: Scales the trajectory speed by the specified factor. Example: A factor of 2 runs the trajectory twice as fast. • Constant: Runs the trajectory at the specified speed.

Item	Description
Factor	When selected, define the Speed Factor. Minimum: 0.1 Maximum: 5 Default 1.0  Tip: Access a previously defined variable by tapping (X) beside the text field.
Constant	When selected, define the Speed and the Acceleration .
Speed	Define the speed at which the arm is to move. Minimum: 10.0 mm/s Maximum: 2,000.00 mm/s Default: 250.0 mm/s
Acceleration	Define the acceleration of the arm. Minimum: 10.0 mm/s² Maximum: 5,000.00 mm/s² Default: 2,000.0 mm/s²
Reverse	Tap to run the trajectory backwards.

Creating a path

Record a trajectory that can be reused in programs.

About this task

Use Hand Guiding mode, or use **Jog Angular** and **Jog Cartesian** in Kortex Web App to move the arm.

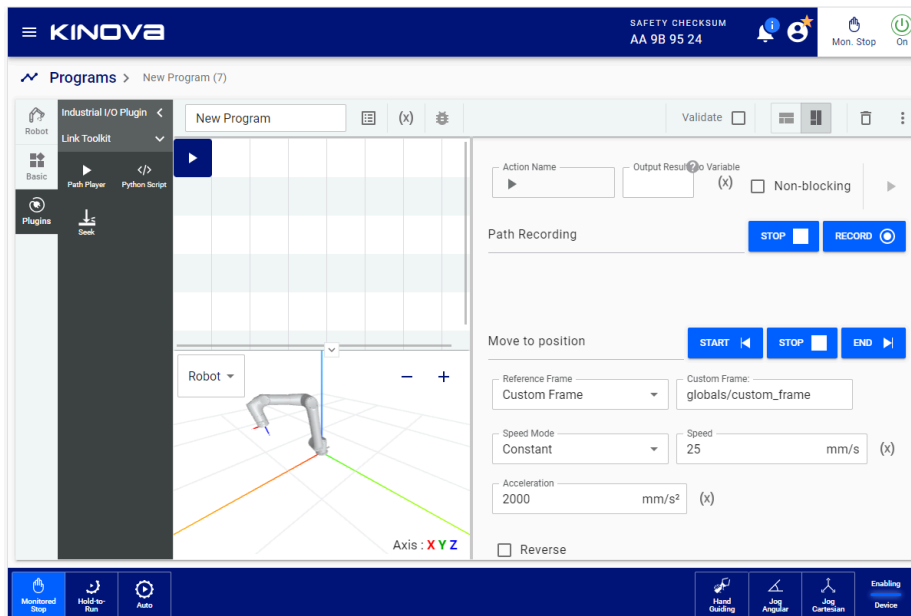


Figure 7: Path Player

Procedure

1. Tap **RECORD** () in the **Path Recording** pane.
2. Tap **Hold to Run** in the .
3. Move the arm along the desired trajectory.
4. Tap **STOP** () in the **Path Recording** pane.
5. Replay the created path to make sure it exactly how you want it.

Option	Description
Full trajectory	Tap  .
Move arm to the first position in the path	Tap START ()
End position	Tap END ().

Python Script action

Scripts speed up the process of creating programs.

When you drag the **Python Script** tile into the **Sequence editor** pane, the **Tile configuration** pane expands to show the name of the action along with an area where you can enter text. The text that can be entered must be a program written in the Python programming language.

Add all code, using Python and the functions available from the Script library, in the **Python text area** after the `import` command. Alternatively, create a Python script externally and bring the file into the list of **Python files available**.

The action outputs information that can be used in subsequent actions in the program.



Figure 8: The Python Script expands in the Sequence editor

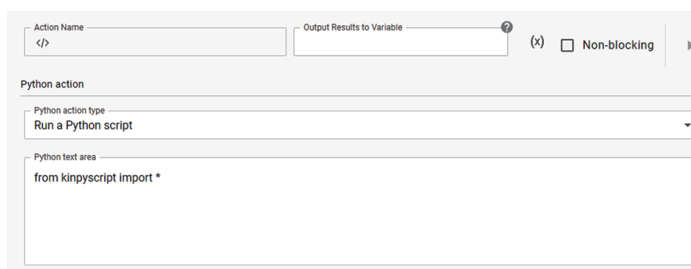


Figure 9: The Tile configuration pane when Python Script is the current tile

Parameters of the Python Script action

Each instance of the **Python Script** is configurable.

Table 6: Python Script parameters

Item	Description
Python action type	Choose one option. - Run a Python script - Run a Python file
Run a Python script	Tap to enter a Python script directly in the text area.

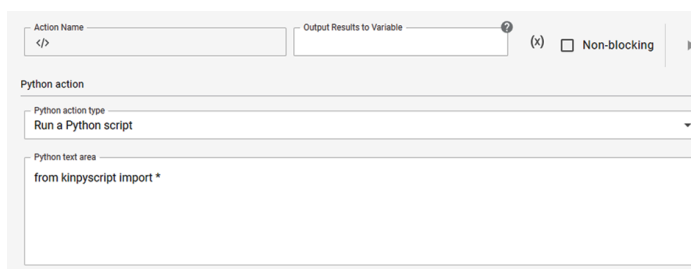


Figure 10: Configuration pane for Run a Python script

Item	Description
Python text area	Enter Python code. Never remove the first line in the text area. <pre>from kinpyscript import *</pre> Add Python code in the Python text area on a new line after the <code>import</code> command. Any errors in the code in the Python text area are displayed in an error dialog with the name Python script Action with a description of the runtime error and a suggestion for correcting it.
Run a Python file	Select a Python file to run.

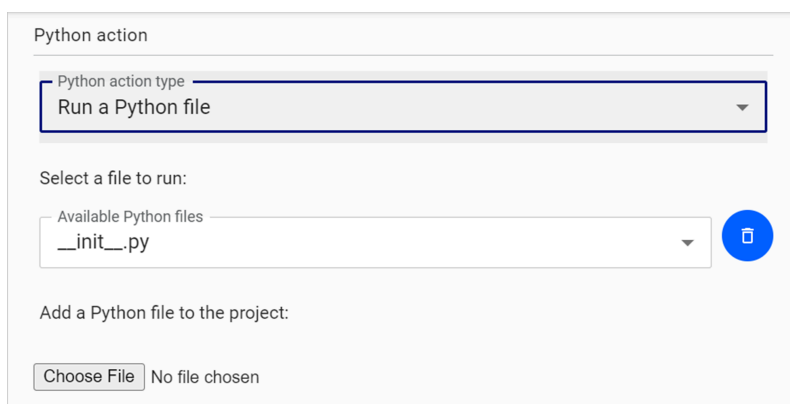


Figure 11: Configuration pane for Run a Python file

Python files available	Select the Python file you want to run.
	Tap to delete the currently selected Python file.
Choose File	Tap to browse for a Python file and add it to the list of files in Python files available .

Creating a script directly in the Python text area

Each script is unique. However, the basic workflow of how to work with scripts is the same each time.

Before you begin

Make sure you know the Python programming language.

About this task

Write the Python script from the **Program** page.



Important: Any script written in the **Python text area** of the **Tile configuration** pane cannot be saved. The result of the script is saved in a variable.

Procedure

1. Tap **Programs** > .
2. Enter a meaningful name for the script by replacing the name "New Program".
3. Tap **Plugins** > **Link Toolkit**.
4. Drag the **Python Script** action onto the **Sequence editor** pane.
The **Tile configuration** pane expands with a **Run a Python script** menu and a **Python text area** pane.

The **Run a Python script** menu displays `Execute a python script`.

The **Run a Python script** displays the initial line of code that is necessary for the script to work.
5. Write code in the **Python text area** pane, using the `kinpyscript` library of functions.
6. Tap  to enter Hold to Run.
7. Test the script by tapping **Play** (.

Using a script that is already prepared

Scripts are used as part of a larger program.

About this task

The program currently loaded in the **Program** page needs to be expanded through the use of pre-existing scripts. These scripts are created and stored either on the computer or on a [USB device](#).

Procedure

1. Tap **Plugins** > **Link Toolkit** on the **Program** page.
2. Drag the **Python Script** action from the Link Toolkit action menu onto the **Sequence editor** pane.
3. Select a script from the **Run a Python script** menu.
4. Optional: Tap **Choose File** to bring the file into list of **Python files available**, and select the file you need, when the script you need to use is not in the list of **Python files available**.

5. Drag actions onto the **Sequence editor** pane until the program is complete.

For each **Python Script** tile dragged onto the **Sequence editor** pane, select the appropriate script from the **Run a Python script** menu.

6. Tap  to enter Hold to Run.

7. Test the script by tapping **Play** (.

8. Tap **Validate** to make the program available to all users.

9. Optional: Tap **Export JSON** to save the entire program in a JSON file.

Seek action

Moving a robot in a specific direction until a specific condition occurs reduces programming time.

The **Seek** tile can be used to speed up programming the robot to move in one direction under circumstances.

The action can be configured through parameters; it outputs information that can be used in subsequent actions in the program.

Table 7: Examples of when to use the Seek action

Application	Description
Stacking and unstacking	Stack and remove items from a pile by taking into account the current height after the last movement.
Assembly	Push a part until it reaches the bottom of the hole.
Machine tending	Push a particular part on the side of the vice.
Welding	Relocate the part before starting to weld.
Reference frames	Probe many locations when the frame of reference changes dynamically.

Parameters of the Seek action

Programming is simplified when the robot can move in one direction until it comes into contact with a specific object.

Use the **Seek** action in different scenarios, where the location of the objects involved in the task must be updated.

- Stacking and unstacking items from a pile that changes height
- Assembling objects

- Machine tending
- Welding

When you drag the **Seek** tile into the **Sequence editor** pane, the **Tile configuration** pane expands and exposes the parameters that can be configured.



Figure 12: The Seek expands in the Sequence editor pane

The configuration pane for the **Seek** action includes the following fields:

- Action Name:** A text input field with a download icon.
- Output Results to Variable:** A text input field with a question mark icon and a close button (x).
- Force Threshold:** A numeric input field set to 10, with units N and a close button (x).
- Speed:** A numeric input field set to 50, with units mm/s and a close button (x).
- Maximum Displacement:** A numeric input field set to 100, with units mm and a close button (x).
- Reference Frame:** A dropdown menu set to Tool Frame.
- Direction Axis:** A dropdown menu set to Z+.

Figure 13: The Tile configuration pane when Seek is the current tile

Table 8: Seek entities

Item	Description
Force Threshold	Enter the amount of force required to stop the Seek action. The value is scalar, relative to the initial force. It is not related to direction. Minimum: 0 N Maximum: 200 N Example: The force sensor shows 10 N. The seek threshold is 20 N. The robot behaves as if the initial force is 0 N and the threshold is 10 N.
Speed	Enter the speed of the arm during the Seek action. Minimum: 0 mm/s Maximum: 250 mm/s
Maximum Displacement	Enter the maximum distance the arm is to travel during the Seek action. Minimum: 0 mm

Item	Description
Reference Frame	Select a frame of reference. • Base Frame • Custom Frame • Tool Frame
Direction Axis	Select the axis on which the arm travels during the Seek action. • X- • X+ • Y- • Y+ • Z- • Z+

Variables

It is possible to create variables that are based on the output values of the actions from the plugin.

Program variables can be one of several types, one of which is **Object**. The **Object** type can take on one of many different schemas related to the pose and angles of the robot, as well as output values coming from the pre-installed Arm plugin. The schema must match the selected action.

In the case of the Link Toolkit plugin, the variable can be set to return the data that is found in the data structure that is output by the Link Toolkit actions.

- **Python Script**
- **Seek**

Python Script data as a variable

Create a variable using the output value of a **Python Script** action on the **Select Variable** page.

The Schema of output variables of the **Python Script** action is `Link Toolkit - Python Script - Output`.

Name	Type	Schema	Unit	Value	Actions
PythonScript output	Object	Link Toolkit - Python Script - Output	N/A	Value {}	 

Figure 14: Data structure returned by the Python Script action

Use the output variable of the action when you are programming with the visual programming interface.

Tap the {} of the variable to see the value returned in the **JSON Object Editor**.

JSON Object Editor

PythonScript

output

Default String...

CANCEL

SAVE

Figure 15: JSON Object Editor of the Python Script action

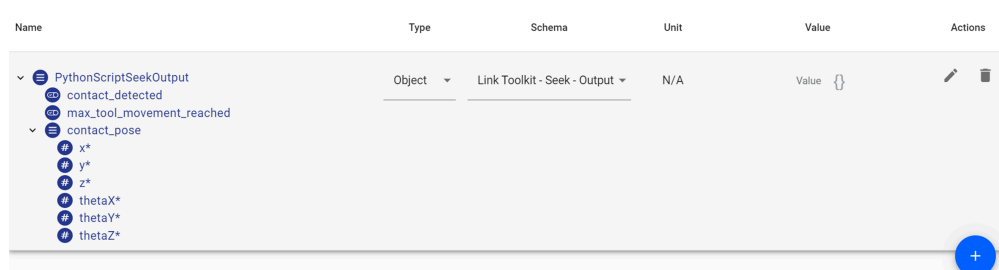
Table 9: Information in Link Toolkit actions that have output data

Data element	Description
output	A string that is the result of running the Python Script action.

Seek data as a variable

Create a variable using the output value of a **Seek** action on the **Select Variable** page.

The Schema of output variables of the **Seek** action is Link Toolkit - Seek - Output.



Name	Type	Schema	Unit	Value	Actions
PythonScriptSeekOutput	Object	Link Toolkit - Seek - Output	N/A	Value {}	
contact_detected					
max_tool_movement_reached					
contact_pose					
x*					
y*					
z*					
thetaX*					
thetaY*					
thetaZ*					

Figure 16: Data structure returned by the Seek action

Use the output variable of the action when you are programming with the visual programming interface.

Modify the values in the **JSON Object Editor** that launches after you tap the {} of the variable.

JSON Object Editor

PythonScriptSeekOutput

☐ Contact detected

☐ Maximum tool movement reached

Contact pose

x
0 mm

y
0 mm

z
0 mm

thetaX
0 °

thetaY
0 °

thetaZ
0 °

Figure 17: Example of JSON Object Editor of the Seek action

Table 10: Information in Link Toolkit actions that have output data

Data element	Description
Contact detected	Select to send a signal whenever contact is detected.
Maximum tool movement reached	Select to send a signal when the maximum tool movement is reached.
Contact pose	When contact is detected, the current Cartesian (x, y, z) and angular (thetaX, thetaY, thetaZ) position of the robot is stored.

Programming manual

Introduction to the Link Toolkit programming guide

The Link Toolkit Plugin is a special plugin that gives the integrator a chance to extend the capabilities of the robot.

Python module

A Python module is a set of files that contain statements and definitions written in Python. Modules are small and independent, making it easier to reuse code and build systems.

For a list of pre-installed modules, refer to the Python3 documentation at <https://docs.python.org/3/py-modindex.html>.

In addition to pre-installed modules, there are pre-imported modules.

```
import csv
import json
import math
import numpy as np
import random
import scipy
import socket
import time

from math import acos
from math import asin
from math import atan
from math import atan2
from math import ceil
from math import cos
from math import exp
from math import fabs
from math import floor
from math import isfinite
```

```
from math import isinf
from math import isnan
from math import log
from math import log10
from math import pow
from math import sin
from math import sqrt
from math import tan
from math import trunc
rad2deg = math.degrees
deg2rad = math.radians
from scipy.spatial.transform import Rotation
from scipy.spatial.transform import Slerp
```

Syntax

Syntax is the way the code needs to be written so that the computer knows how to read the instructions and compile the program. It includes how to write comments, how to indent, how to define variables, and how to define functions.

Comment

Comments are used to explain your code to make it more readable.

Comments can also be used to prevent a line of code from running during testing.

Single line comment

A comment that is on a single line starts with a pound sign (#). All text after the # on that line is ignored by the compiler. The text on the next line that is not starting with # is read by the compiler.

Example - Single line comment:

```
#This is a comment
print("no comments")
```

```
print("comment affect a line of code") #comment this line of code
```

Multiline comment

A comment that is on multiple lines starts with three quotation marks (""") and ends with three quotation marks ("""). You can place as many lines of text as needed when you use a pair of three quotation marks to mark the comment. All text after "" and before the next "" is ignored by the compiler.

Example - Multiline comment:

```
"""
This is a comment
placed
on multiple lines
"""
a = 12
```

Indentation

Indentation refers to the spaces at the beginning of a line of code. They are very important to indicate a block of code. Indentation can be created by one or more spaces or by using the Tab key one or more times.

You must use the same indentation for each line of code that is within the same block of code. Another level of indentation within a block of code signals another, even smaller, block of code.

Example - Incorrect syntax of indentation:

```
# block of code must be indicated by indentation
if i<4:
log("i is smaller than 4")

# indentation must remain constant in a block of code
if i<4:
    log("i is smaller")
    log("than 4")
```

Example - Correct syntax of indentation:

```
if i<4:
    log("i is smaller")
    log(" than 4")
```

Variable

Variables are created when you first assign a value to it; variables do not need to be declared.

Change the type of a variable by assigning to it a value with a new type.

Example - Create variable:

```
i = 4          #create a variable of type integer
i = "four"     #i is now of type string
```

Each variable is made up of three parts.

- Name
- Type
- Scope

Each of the parts of a variable follow a specific syntax.

Name

Always provide a meaningful name; it helps when debugging code. Do not use names like x1, x2, x3, xx1, xx2, xx3.

Aside from having a meaningful name, the name must adhere to Python syntax.

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores (_).
- Variable names are case-sensitive; for instance, age, Age and AGE are three different variables.

Type

All the default, built-in Python data types are available in the Link Toolkit Plugin. The plugin defines the `Pose` data type.

The variable of type `Pose` are objects defined by the position and orientation of a coordinate frame. The Cartesian x, y, and z members represent the position in meters. The rotational values, `theta_x`, `theta_y`, and `theta_z`, represent the orientation in Euler's angles. The `Pose` object are matrices, which are multidimensional arrays. They are defined as `Pose([x = 0.0, y = 0.0, z = 0.0, theta_x = 0.0, theta_y = 0.0, theta_z = 0.0])`

Initialize a Pose variable using the default constructor, separate parameters, a list, or a `numpy.ndarray`.

```
pose_1 = Pose()      # Default constructor where all parameters are set to 0

pose_2 = Pose(0.1, -0.5, 0.4, 180, 0, 180)

pose_3_list = [0.1, -0.5, 0.4, 180, 0, 180]
pose_3 = Pose(pose_3_list)

pose_4_array = np.array([0.1, -0.5, 0.4, 180, 0, 180])
pose_4 = Pose(pose_4_array)
```

Access each element of a Pose using its name or using the slicing function of Python.

```
pose_1 = Pose(0.1, -0.5, 0.4, 180, 0, 180)

x = pose_1.x
y = pose_1.y
z = pose_1.z
theta_x = pose_1.theta_x
theta_y = pose_1.theta_y
theta_z = pose_1.theta_z

x = pose_1[0]
y = pose_1[1]
z = pose_1[2]
theta_x = pose_1[3]
theta_y = pose_1[4]
theta_z = pose_1[5]

pose_position = pose_1[0:3]
pose_orientation = pose_1[3:6]
```

Convert a Pose to a `numpy.array` with the function `to_array`.

```
pose_1 = Pose(0.1, -0.5, 0.4, 180, 0, 180)

pose_1_array = pose_1.to_array()      # Returns a numpy.array
```

Cast a Pose object with the `str()` function to print the Pose in the event logger.

```
pose_1 = Pose(0.1, -0.5, 0.4, 180, 0, 180)

pose_1_str = str(pose_1)

log_event(pose_1_str)
```

The values of the Pose are logged in the event logger of Kortex Web App.

Variable management

In all programming, two functions are required to handle variables: a setter and a getter. Only persistent variables have read and write permissions.

`get_variable(name)`

Returns the value of a variable defined in Kortex Web App.

Parameters

name

Type	String
Default value	-
Description	Namespace and name of the variable as seen in Kortex Web App

Returns

The return type depends on the type of the input variable `name`.

Variable type	Return type
String	str
Boolean	bool
Number	float
Pose	Pose
Joint angles	np.array:6
Matrix	list of np.array:6
Waypoints	list of 2 objects
	index 0: Pose
	index 1: list of np.array:6

Variable type	Return type
Other	JSON Object

Example:

```
my_var = get_variable("globals/my_var")
```

If `globals/my_var` is boolean, `get_variable()` returns a boolean value.

set_variable(name, value)

Sets the value of variable defined in Kortex Web App.

Parameters

name

Type	str
Default value	-
Description	Namespace and name of the variable as seen in Kortex Web App

value

The type of the value must match the type of the input variable `name`.

Type	bool int or float Pose Array of numbers  Note: Numbers are either of type <code>int</code> or <code>float</code> List of Pose JSON
Default value	-
Description	The type of the input variable <code>value</code> must have the same type as the input variable <code>name</code> .

Returns

The return type depends on the type of the input variable `name`.

Variable type	Return type	
String	str	
Boolean	bool	
Number	float	
Pose	Pose	
Joint angles	np.array:6	
Matrix	list of np.array:6	
Waypoints	list of 2 objects	index 0: Pose index 1: list of np.array:6
Other	JSON Object	

Example:

```
my_var = set_variable("globals/my_var", True)
# Sets persistent variable globals/my_var to True;
# globals is the namespace
```

IO Access

Multiple setters and getters exist for the sole purpose of controlling the pure electrical signal information exchange among the Kortex Web App, the controller, and the arm.

config_analog_input_to_current(channel_id)

Configures the analog input challenge to current between 4 and 20 mA.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	3
Default	–
Description	ID of the analog input channel

Example:

```
config_analog_input_to_current(0)
```

```
# Configures the analog input channel 0 to current (4 to 20 mA)
```

config_analog_input_to_voltage(channel_id)

Configures the analog input channel to voltage between 0 and 10 V.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	3
Default	–
Description	ID of the analog input channel

Example:

```
config_analog_input_to_voltage(0)
# Configures the analog input channel 0 to voltage between 0 and 10 V
```

config_analog_output_to_current(channel_id)

Configures the analog output channel to current the digital output state of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	3
Default	–
Description	ID of the analog output channel

Example:

```
config_analog_output_to_current(0)
# Configures the analog output channel 0 to current from 4 to 20 mA
```

configure_analog_output_to_voltage(channel_id)

Configures the analog output channel to voltage from 0 to 10 V.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	3
Default	–
Description	ID of the analog output channel

Example:

```
config_analog_output_to_voltage(0)
# Configures the analog output channel 0 to voltage (0 to 10 V)
```

config_wrist_analog_input_to_current(channel_id)

Configures the wrist analog input channel to current from 4 to 20 mA.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	1
Default	–
Description	ID of the analog input channel

Example:

```
config_wrist_analog_input_to_current(0)
# Configures the analog input channel 0 to current from 4 to 20 mA
```

config_wrist_analog_input_to_voltage(channel_id)

Configures the wrist analog input channel to voltage between 0 and 10 V.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	1

Default	–
Description	ID of the analog input channel

Example:

```
config_wrist_analog_input_to_voltage(0)
# Configures the analog input channel 0 to voltage from 0 to 10 V
```

config_wrist_analog_output_to_current(channel_id)

Configures the wrist analog output channel to current between 4 and 20 mA.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	1
Default	–
Description	ID of the analog output channel

Example:

```
config_wrist_analog_output_to_current(0)
# Configures the analog output channel 0 to current, between 4 and 20 mA
```

config_wrist_analog_output_to_voltage(channel_id)

Configures the wrist analog output channel to voltage between 0 and 10 V.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	1
Default	–
Description	ID of the analog output channel

Example:

```
config_wrist_analog_output_to_voltage(0)
```

```
# Configures the analog output channel 0 to voltage, between 0 and 10 V
```

config_wrist_channel_to_digital_input(channel_id)

Configures the wrist channel to digital input.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital input channel

Example:

```
config_wrist_channel_to_digital_input(0)  
# Configures the wrist channel 0 to digital input
```

config_wrist_channel_to_digital_output(channel_id)

Configures the wrist channel to digital output.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital output channel

Example:

```
config_wrist_channel_to_digital_output(0)  
# Configures the wrist channel 0 to digital output
```

get_analog_input_value(channel_id)

Gets the value of the analog input signal of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	3
Default	–
Description	ID of the analog input channel

Returns

Value	Type	Config	Min	Max	Units
Analog value	float	Voltage	0.0	11.0	V
		Current	0.0	25.0	mA

Example:

```

config_analog_input_to_voltage(0)
    # Configures analog input channel 0 to voltage from 0 to 10 V

ai_0 = get_analog_input_value(0)
    # Returns analog input 0 voltage value from 0.0 to 11.0 V

config_analog_input_to_current(0)
    # Configures analog input channel 0 to current from 4 to 20mA

ai_0 = get_analog_input_value(0)
    # Returns analog input 0 current value from 0.0 to 25.0 mA

```

get_analog_output_value(channel_id)

Gets the value of the analog output signal of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	3
Default	–
Description	ID of the analog output channel

Returns

Value	Type	Config	Min	Max	Units
Analog value	float	Voltage	0.0	11.0	V
		Current	0.0	25.0	mA

Example:

```

config_analog_output_to_voltage(0)
# Configures analog output channel 0 to voltage from 0 to 10 V

ao_0 = get_analog_output_value(0)
# Returns analog output 0 voltage value from 0.0 to 11.0 V

config_analog_output_to_current(0)
# Configures analog output channel 0 to current from 4 to 20mA

ao_0 = get_analog_output_value(0)
# Returns analog output 0 current value from 0.0 to 25.0 mA

```

get_digital_input(channel_id)

Returns the digital input signal level of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default value	–
Description	ID of the digital input channel

Returns

Variable type	Return type
Signal level	bool

Example:

```

di_0 = get_digital_input(0) # Returns True or False

```

get_digital_inputs(start_id, end_id, to_int)

Returns the digital input signal level of the controller.

Parameters

start_id (optional)

Type	int
Minimum value	0
Maximum value	6
Default value	0
Description	ID of the first digital input channel

end_id (optional)

Type	int
Minimum value	1
Maximum value	7
Default value	7
Description	ID of the last digital input channel

to_int (optional)

Type	bool
Minimum value	–
Maximum value	–
Default value	False
Description	Convert the return value from a list of bool to an int

Returns

Variable type	Return type
Signal level	list of bool (int to_int=False)
	(to_int=True)

Example:

```
di_all = get_digital_inputs()
# Returns a list of bool for channels 0 to 7
```

```

di_0_3 = get_digital_inputs(0, 3)
    # Returns a list of bool for channels 0 to 3

di_4_7 = get_digital_inputs(4, 7)
    # Returns a list of bool for channels 4 to 7

di_int = get_digital_inputs(to_int=True)
    # Returns an int for channels 0 to 7

di_int_0_3 = get_digital_inputs(0, 3, True)
    # Returns an int for channels 0 to 3

di_int_4_7 = get_digital_inputs(4, 7, True)
    # Returns an int for channels 4 to 7

```

get_digital_output(channel_id)

Returns the digital output signal level of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital output channel

Returns

Variable type	Return type
Signal level	bool

Example:

```
do_0 = get_digital_output(0) # Returns True or False
```

get_digital_outputs(start_id, end_id, to_integer)

Returns the digital output signal levels of the controller.

Parameters

start_id (optional)

Type	int
Minimum value	0
Maximum value	6
Default	0
Description	ID of the first digital output channel

end_id (optional)

Type	int
Minimum value	1
Maximum value	7
Default	7
Description	ID of the last digital output channel

to_int (optional)

Type	bool
Minimum value	–
Maximum value	–
Default	False
Description	Convert the return value from a list of bool to an int

Returns

Variable type	Return type
Signal level	list of bool (to_int=False)
	int (to_int=True)

Example:

```
do_all = get_digital_outputs()
# Returns a list of bool for channels 0 to 7

do_0_3 = get_digital_outputs(0, 3)
# Returns a list of bool for channels 0 to 3

do_4_7 = get_digital_outputs(4, 7)
```

```
# Returns a list of bool for channels 4 to 7

do_int = get_digital_outputs(to_int=True)
# Returns an int for channels 0 to 7

do_int_0_3 = get_digital_outputs(0, 3, True)
# Returns an int for channels 0 to 3

do_int_4_7 = get_digital_outputs(4, 7, True)
# Returns an int for channels 4 to 7
```

get_safety_digital_input(channel_id)

Gets the signal level for the safety digital input.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital input channel

Returns

Value	Type
Signal level	bool

Example:

```
di_0 = get_safety_digital_input(0) # Returns True or False
```

get_safety_digital_inputs(to_int)

Gets the signal levels of the safety digital inputs.

Parameters

to_int

Type	bool
Minimum value	–

Maximum value	–
Default	False
Description	Converts the return value from a list of bool to an int

Returns

Value	Type
Signal levels	list of bool to_int=False
	int to_int=True

Example:

```
di_all = get_safety_digital_inputs()
# Returns a list of bool for channels 0 to 7

di_int = get_safety_digital_inputs(to_int=True)
# Returns an int for channels 0 to 7
```

get_safety_digital_output(channel_id)

Gets the signal level of the safety digital output.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital output channel

Returns

Value	Type
Signal level	bool

Example:

```
do_0 = get_safety_digital_output(0) # Returns True or False
```

get_safety_digital_outputs(to_int=False)

Gets the signal levels of the safety digital outputs.

Parameters

to_int (optional)

Type	bool
Minimum value	–
Maximum value	–
Default	False
Description	Converts the return value from a list of bool to an int

Returns

Value	Type
Signal levels	List of bool (to_int=False)
	int (to_int=True)

Example:

```
do_all = get_safety_digital_outputs()
# Returns a list of bool for channels 0 to 7

do_int = get_safety_digital_outputs(to_int=True)
# Returns an int for channels 0 to 7
```

get_wrist_analog_input_value(channel_id)

Get the signal value of the analog input of the wrist.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	1
Default	–
Description	ID of the analog input channel

Returns

Value	Type	Config	Min	Max	Units
Analog value	float	Voltage	0.0	11.0	V
		Current	0.0	25.0	mA

Example:

```

config_wrist_analog_input_to_voltage(0)
# Configures analog input channel 0 to voltage, between 0 and 10 V

ai_0 = get_wrist_analog_input_value(0)
# Returns analog input 0 voltage value, between 0.0 and 11.0 V

config_wrist_analog_input_to_current(0)
# Configures analog input channel 0 to current, between 4 and 20 mA

ai_0 = get_wrist_analog_input_value(0)
# Returns analog input 0 current value, between 0.0 and 25.0 mA

```

get_wrist_analog_output_value(channel_id)

Gets the signal value of the analog output of the wrist.

Parameters**channel_id**

Type	int
Minimum value	0
Maximum value	1
Default	–
Description	ID of the analog output channel

Returns

Value	Type	Config	Min	Max	Units
Analog value	float	Voltage	0.0	11.0	V
		Current	0.0	25.0	mA

Example:

```

config_wrist_analog_output_to_voltage(0)
# Configures analog output channel 0 to voltage, between 0 and 10 V

```

```

ao_0 = get_wrist_analog_output_value(0)
# Returns analog output 0 voltage value, between 0.0 and 11.0 V

config_wrist_analog_output_to_current(0)
# Configures analog output channel 0 to current, between 4 and 20 mA

ao_0 = get_wrist_analog_output_value(0)
# Returns analog output 0 current value, between 0.0 and 25.0 mA

```

get_wrist_digital_input(channel_id)

Gets the signal level for the digital input of the wrist.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital input channel

Returns

Value	Type
Signal level	bool

Example:

```
di_0 = get_wrist_digital_input(0) # Returns True or False
```

get_wrist_digital_output(channel_id)

Gets the signal level for the digital output of the wrist.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–

Description	ID of the digital output channel
Returns	

Value	Type
Signal level	bool

Example:

```
di_0 = get_wrist_digital_output(0)      # Returns True or False
```

set_analog_output_value(channel_id, value)

Sets the signal value for the signal value of the controller.

Parameters**channel_id**

Type	int
Config	–
Minimum value	0
Maximum value	3
Units	–
Default	–
Description	ID of the analog output channel

value

Type	float
Config	Voltage
Minimum value	0.0
Maximum value	11.0
Units	V
Default	–
Description	Analog value in voltage
Type	float
Config	Current
Minimum value	0.0

Maximum value	25.0
Units	mA
Default	–
Description	Analog value in current

Returns

A variable of one type returns that type.

Value	Type	Config	Min	Max	Units
Analog value	float	Voltage	0.0	11.0	V
		Current	0.0	25.0	mA

Example:

```
config_analog_output_to_voltage(0)
# Configures analog output channel 0 to voltage (0 to 10V)

set_analog_output_value(0, 9.5)
# Sets analog output 0 to 9.5V

config_analog_output_to_current(0)
# Configures analog output channel 0 to current (4 to 20mA)

set_analog_output_value(0, 20.8)
# Sets analog output 0 to 20.8mA
```

set_digital_output(channel_id, value)

Sets the digital output state of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	6
Default	0
Description	ID of the digital output channel

value

Type	bool
Minimum value	–
Maximum value	–
Default	–
Description	Output state

Example:

```
set_digital_output(0, False)    # Sets channel 0 to False
set_digital_output(0, True)    # Sets channel 0 to True
```

set_digital_outputs(digital_outputs, start_id, end_id)

Sets the digital output state of the controller.

Parameters

digital_outputs

Type	list or int
Minimum value	–
Maximum value	–
Default	–
Description	Digital output states

start_id (optional)

Type	int
Minimum value	0
Maximum value	6
Default	0
Description	ID of the first digital output channel

end_id (optional)

Type	int
Minimum value	1
Maximum value	7
Default	7
Description	ID of the last digital output channel

Example:

```

set_digital_outputs([False] * 8)
    # Sets all digital outputs to False using list of bool

set_digital_outputs(0)
    # Sets all digital outputs to False using int

set_digital_outputs([True] * 8)
    # Sets all digital outputs to True using list of bool

set_digital_outputs(255)
    # Sets all digital outputs to True using int

set_digital_outputs([False, True, False, True, False, True, False, True])
    # Sets all digital outputs according to list of bool

set_digital_outputs(170)
    # Sets all digital outputs according to int

set_digital_outputs([False, True, True, True], 0, 3)
    # Sets digital outputs 0 to 3 according to list of bool

set_digital_outputs(7, 0, 3)
    # Sets digital outputs 0 to 3 according to int

```

set_wrist_analog_output_value(channel_id, value)

Sets the analog output signal value of the wrist.

Parameters**channel_id**

Type	int
Configuration type	–
Minimum value	0
Maximum value	3
Units	–
Default	–
Description	ID of the analog output channel

value

Type	float
Configuration type	Voltage
Minimum value	0.0
Maximum value	11.0
Units	V
Default	–
Description	Analog value in voltage
	Current
Minimum value	0.0
Maximum value	25.0
Units	mA
Default	–
Description	Analog value in current

Returns

Value	Type	Config	Min	Max	Units
Analog value	float	Voltage	0.0	11.0	V
		Current	0.0	25.0	mA

Example:

```

config_wrist_analog_output_to_voltage(0)
# Configures analog output channel 0 to voltage, between 0 and 10 V

set_wrist_analog_output_value(0, 9.5)
# Sets analog output 0 to 9.5 V

config_wrist_analog_output_to_current(0)
# Configures analog output channel 0 to current, between 4 and 20 mA

set_wrist_analog_output_value(0, 20.8)
# Sets analog output 0 to 20.8 mA

```

set_wrist_digital_output(channel_id, value)

Sets the digital output state of the controller.

Parameters

channel_id

Type	int
Minimum value	0
Maximum value	7
Default	–
Description	ID of the digital output channel

value

Type	bool
Minimum value	–
Maximum value	–
Default	–
Description	Output state

Example:

```
set_wrist_digital_output(0, False)    # Sets channel 0 to False
set_wrist_digital_output(0, True)    # Sets channel 0 to True
```

Mathematics

The Link Toolkit Plugin interface includes mathematical functions for working with binary, pose, force, and torque values.

align_z_axis(pose1, pose2)

Aligns the Z axis of the first Pose according to second Pose.

Parameters

pose1

Type	Pose
Default	–
Description	Starting Pose of the coordinate system

pose2

Type	Pose
Default	–
Description	Pose to transform according to p_from

Returns

Value	Type
Transformed Pose	Pose

Example:

```
pose_1 = Pose(0.2, 0.5, 0.5, -180, 0, 180)
pose_2 = Pose(0.2, 0.5, 0.3, -20, -180, 0)

aligned_pose_1 = align_z_axis(pose_1, pose_2)
# Returns [0.2, 0.5, 0.5, 160, 0, 180]
```

binary_list_to_int(binary_list)

Converts a binary list to an int.

Parameters

binary_list

Type	list or np.array
Default	–
Description	Binary list to convert

Returns

Value	Type
Integer value	int

Example:

```
int_0 = binary_list_to_int([0, 0, 0, 0]) # Returns 0
int_9 = binary_list_to_int([1, 0, 0, 1]) # Returns 9
int_83 = binary_list_to_int(np.array([1, 1, 0, 0, 1, 0, 1])) # Returns 83
```

express_vector_in_frame(T_from_to, V_from)

Expresses a vector in another frame.

Parameters

T_from_to

Type	list or np.array or Pose
Default	–
Description	Transformation frame

V_from

Type	list or np.array or Pose
Default	–
Description	Vector to transform

Returns

Value	Type
Transformed vector	np.array

Example:

```
T_from_to = [0.132283, -0.0567901, -0.0104075, 85.4591, 64.237, 56.1958]
W_from = [-12.8336, 54.8782, 60.0224, 26.1588, -25.4933, -8.29345]
vector = express_vector_in_frame(T_from_to, W_from)
```

express_wrench_in_frame(T_from_to, W_from)

Expresses the wrench in another frame.

Parameters

T_from_to

Type	list or np.array or Pose
Default	–
Description	Transformation frame

W_from

Type	list or np.array or Pose
Default	–
Description	Wrench to transform

Returns

A variable of one type returns that type.

Value	Type
Transformed wrench	np.array

Example:

```
T_from_to = [0.132283, -0.0567901, -0.0104075, 85.4591, 64.237, 56.1958]
W_from = [-12.8336, 54.8782, 60.0224, 26.1588, -25.4933, -8.29345]
wrench = express_wrench_in_frame(T_from_to, W_from)
```

int_to_binary_list(int_value, list_length)

Converts an integer value to a binary list.

Parameters

int_value

Type	int
Default	-
Description	Integer value to convert

list_length (optional)

Type	int
Default	None
Description	Length of the returned binary list. Length must be long enough to hold the integer value. If not specified, the length of the returned list is the minimum length to hold the integer.

Name	Type	Min	Max	Default	Description
int_value	int	-	-	-	Integer value to convert
list_length	int	-	-	None	Length of the returned binary list. Length must be long enough to hold the integer value.

Name	Type	Min	Max	Default	Description
					If not specified, the length of the returned list is the minimum length to hold the integer.

Returns

Value	Type
Binary list	List of bool

Example:

```
binary_list_9 = int_to_binary_list(9)
# Returns [True, False, False, True]

binary_list_83 = int_to_binary_list(83)
# Returns [True, True, False, False, True, False, True]

binary_list_19 = int_to_binary_list(19, 6)
# Returns [True, True, False, False, True, False]
```

interpolate_pose(p_from, p_to, alpha)

Interpolates the position between two poses.

Parameters

p_from

Type	Pose
Default	–
Description	Start pose of the linear interpolation

p_to

Type	Pose
Default	–
Description	End pose of the linear interpolation

alpha

Type	float
Default	–

Description

Distance from 0 to 1 of the interpolation

When `alpha=0`, `p_from` Pose is returnedWhen `alpha=1`, `p_to` Pose is returnedWhen `alpha` is between 0 and 1, the returned Pose is on a line between `p_from` and `p_to` at an `alpha` distance.**Returns**

Value	Type
Interpolated pose	Pose

Example:

```
p_from = Pose([0, 0, 0, 0, 0, 90])
p_to    = Pose([0.2, 0.4, 0.6, 90, 0, 0])

interpolated_pose = interpolate_pose(p_from, p_to, 0.5)
# Returns [0.1, 0.2, 0.3, 45, -19.47, 45]
```

point_dist(p_from, p_to)

Returns the distance between two Poses.

Parameters**p_from**

Type	Pose
Default	–
Description	Start pose

p_to

Type	Pose
Default	–
Description	End pose

Returns

Value	Type
Distance between two Poses	float

Example:

```
p_from = Pose([0.2, 0.5, 0.1, 90.0, 0.0, 180.0])
p_to   = Pose([0.5, 0.5, 0.1, 0, 90.0, 180.0])

dist = point_dist(p_from, p_to)    # Returns 0.3
```

pose_inv(pose)

Returns the inverse of a Pose.

Parameters**pose**

Type	Pose or list
Default	–
Description	Pose to invert

Returns

Value	Type
Inverted pose	Pose or list (same type as input)

Example:

```
pose = Pose(0.2, 0.4, 0.2, 90, 0, 90)

inv_pose = pose_inv(pose)    # Returns [-0.4, -0.2, -0.2, -90, -90, 0]
```

pose_transform(p_from, p_to)

Transforms a Pose.

Parameters**p_from**

Type	Pose
Default	–
Description	Starting Pose of the coordinate system

p_to

Type	
Default	–

DescriptionPose to transform according to `p_from`

Returns

Value	Type
Transformed Pose	Pose

Example:

```
T_flange_base = Pose([-0.0341, -0.5402, 0.5894, 147.608, 19.85, 32.61])
T_tcp_base = Pose([0.181, -0.794, 0.418, -135.706, -32.127, 38.049])

T_tcp_flange = pose_transform(T_flange_base, T_tcp_base)
# Returns [-0.3539557787889696, -0.21474056458795449,
-0.20412973269851353,
#          3.226836114799741, 22.694572262819005, -19.25217976891005]
```

wrench_trans(T_from_to, W_from)

Transformation of a wrench to move the point of view of a wrench.

Parameters

T_from_to

Type	list or np.array
Default	–
Description	Transformation to the new point of view

W_from

Type	list or np.array
Default	–
Description	Wrench to transform

Returns

Value	Type
Transformed wrench	np.array

Example:

```
T_from_to = np.array([0.1, 0, 0, 0, 0, 0])
wrench = np.array([0, 0, -10, 0, 0, 0])
```

```
transformed_wrench = wrench_trans(pose_inv(T_from_to), wrench)
# Returns [0, 0, -10, 0, 1, 0]
```

Robot

The Link Toolkit Plugin interface includes functions retrieving information from the robot.

forward_kin(joint_positions)

Calculates the forward kinematic transformation from the joint space to the tool space, using the calibrated robot kinematics.

Parameters

Parameters

joint_positions (optional)

Type	np.array of 6 elements
Default	–
Description	Joint positions If the joint positions are not provided, the actual joint positions are used.

Returns

Value	Type
Tool pose	Pose

Example:

```
pose_1 = forward_kin() # Forward kinematics from current joint positions
pose_2 = forward_kin([90, 0, 90, 0, 0, 0]) # Forward kinematics from
specific joint positions
```

get_feedback()

Gets feedback from the base, actuators, and the wrist.

Parameters

none

Returns

Value	Type
Robot feedback	JSON object

JSON object feedback structure :

```
base {
  active_state_connection_identifier: 0
  active_state: ARMSTATE_IDLE
  arm_voltage: 12.942907333374023
  arm_current: 0.48503151535987854
  temperature_cpu: 66.05000305175781
  temperature_ambient: 72.02999877929688
  imu_acceleration_x: 2.0
  imu_acceleration_y: 3.0
  imu_acceleration_z: 3.0
  imu_angular_velocity_x: 5.0
  imu_angular_velocity_y: 1.0
  imu_angular_velocity_z: 1.0
  tool_pose_x: 0.5
  tool_pose_y: 0.5
  tool_pose_z: 0.5
  tool_pose_theta_x: -180.0
  tool_pose_theta_y: -180.0
  tool_pose_theta_z: -180.0
  tool_twist_linear_x: 0.00012667257396969944
  tool_twist_linear_y: -0.003573123598471284
  tool_twist_linear_z: 0.004279812797904015
  tool_twist_angular_x: 7.0
  tool_twist_angular_y: 2.0
  tool_twist_angular_z: 3.0
  tool_external_wrench_force_x: 1.0
  tool_external_wrench_force_y: 4.0
  tool_external_wrench_force_z: 8.0
  tool_external_wrench_torque_x: 4.0
  tool_external_wrench_torque_y: 2.0
  tool_external_wrench_torque_z: 0.0
  fault_bank_a: 0
  fault_bank_b: 0
  warning_bank_a: 0
  warning_bank_b: 0
  commanded_tool_pose_x: 0.5
```

```

    commanded_tool_pose_y: 0.5
    commanded_tool_pose_z: 0.5
    commanded_tool_pose_theta_x: -180.0
    commanded_tool_pose_theta_y: -180.0
    commanded_tool_pose_theta_z: -180.0
    operating_mode: OPERATING_MODE_UNSPECIFIED
    enabling_device_state: true
    scu_fault_bank_a: 0
    scu_fault_bank_b: 0
    scu_fault_bank_c: 0
    scu_fault_bank_d: 0
    flange_pose {
      x: 0.5
      y: 0.5
      z: 0.5
      theta_x: 180.0
      theta_y: 180.0
      theta_z: 180.0
    }
    is_emergency_stop_activated: false
  }
  actuators [0..5] {
    status_flags: 0
    jitter_comm: 0
    position: 73.00166320800781
    velocity: 9.0
    torque: 3.0
    current_motor: 1.09049391746521
    voltage: 11.69679069519043
    temperature_motor: 42.90999984741211
    temperature_core: 75.5999984741211
    fault_bank_a: 0
    fault_bank_b: 0
    warning_bank_a: 0
    warning_bank_b: 0
    sfty_fault_bank_a: 0
    fault_bank_c: 0
    sfty_fault_bank_b: 0
  }
  wrist {
    c61 {
      feedback_id {

```

```

        identifier: 2147887592
    }
    status_flags: 16
    buttons_state: 0
    digital_inputs_state: 0
    analog_input_a_channel_1_value: 0.0
    analog_input_a_channel_2_value: 0.0
    analog_input_b_channel_1_value: 0.0
    analog_input_b_channel_2_value: 0.0
    force_x: 10.784529685974121
    force_y: 7.946324348449707
    force_z: 2.4408481121063232
    torque_x: 0.04377690330147743
    torque_y: 0.01991846226155758
    torque_z: 0.17334674298763275
    imu_acceleration_x: -0.19859111309051514
    imu_acceleration_y: 2.510502815246582
    imu_acceleration_z: -9.765658378601074
    imu_angular_velocity_x: -1.8025000095367432
    imu_angular_velocity_y: 2.5199999809265137
    imu_angular_velocity_z: -1.2337499856948853
    voltage: 48.4056396484375
    temperature_core: 49.20000076293945
    fault_bank_a: 0
    fault_bank_b: 0
    }
}

```

Example:

```

feedback = get_feedback()

base_feedback = feedback.base
actuator_0 = feedback.actuators[0]
actuator_1 = feedback.actuators[1]
actuator_2 = feedback.actuators[2]
actuator_3 = feedback.actuators[3]
actuator_4 = feedback.actuators[4]
actuator_5 = feedback.actuators[5]
wrist_feedback = feedback.wrist.c61

```

get_flange_pose()

Gets the Pose of the flange.

Parameters

none

Returns

Value	Type
Flange pose	Pose

Example:

```
flange_pos = get_flange_pose()
```

get_gravity()

Gets the gravity vector for the x, y, and z axes.

Parameters

none

Returns

Value	Type
Gravity vector x, y, z	list: 3 float

Example:

```
gravity = get_gravity()
```

get_joint_currents()

Gets the current of each joint motor in amperes.

Parameters

none

Returns

Value	Type
Joint motor currents (A)	np.array of 6 elements (joints 0 to 5)

Example:

```
joint_currents = get_joint_currents()
```

get_joint_positions()

Gets the positions of the joints.

Parameters

none

Returns

Value	Type	
List with the position of all joints	np.array of 6 elements	(joints 0 to 5)

Example:

```
joint_positions = get_joint_positions()
```

get_joint_speeds()

Gets the speeds of the joints.

Parameters

none

Returns

Value	Type	
List with speeds of all joints	np.array of 6 elements	(joints 0 to 5)

Example:

```
joint_speeds = get_joint_speeds()
```

get_joint_torques()

Gets the torques of all joints.

Parameters

none

Returns

Value	Type
List with torques of all joints	np.array of 6 elements (joints 0 to 5)

Example:

```
joint_torque = get_joint_torques()
```

get_payload()

Gets the mass and center of gravity of the payload.

Parameters

none

Returns

Value	Type
mass (kg)	float
Center of mass (x, y, z)	list: 3 float

Example:

```
mass, cog = get_payload()
```

get_tcp_offset()

Gets the offset between the Pose of the flange pose and the Pose of the TCP.

Parameters

none

Returns

Value	Type
TCP offset	Pose

Example:

```
tcp_offset = get_tcp_offset()
```

get_tcp_pose()

Gets the pose of the TCP.

Parameters

none

Returns

Value	Type
TCP pose	Pose

Example:

```
tcp_pose =- get_tcp_pose()
```

get_tcp_twist()

Gets the twist of the TCP.

Parameters

none

Returns

Value	Type
TCP twist	np.array of 6 elements (x, y, z, θ_x , θ_y , θ_z)

Example:

```
tcp_twist = get_tcp_twist()
```

get_tool_imu()

Gets the inertial measurement units of the tool.

Parameters

none

Returns

Value	Type
Flange pose	np.array of 6 elements (acceleration x, acceleration y, acceleration z, angular velocity x, angular velocity y, angular velocity z)

Example:

```
tool_imu = get_tool_imu()
```

get_wrench_in_base_frame()

Gets the wrench in the base frame.

Parameters

none

Returns

Value	Type
Wrench in base frame	np.array of 6 elements

Example:

```
wrench = get_wrench_in_base_frame()
```

get_wrench_in_flange_frame()

Gets the wrench in the flange frame.

Parameters

none

Returns

Value	Type
Wrench in flange frame	np.array of 6 elements

Example:

```
wrench = get_wrench_in_flange_frame()
```

get_wrench_in_tcp_frame()

Gets the wrench in the TCP frame

Parameters

none

Returns

Value	Type
Wrench in TCP frame	np.array of 6 elements

Example:

```
wrench = get_wrench_in_tcp_frame()
```

inverse_kin(pose, near_joint_angles)

Calculates the inverse kinematic transformation from tool space to joint space. Otherwise, the solution closest to the current joint positions is returned. If no TCP is provided the currently active TCP of the controller is used.

Calculates the inverse of the kinematic transformation from the tool space to the joint space. It returns the solution closest to the current joint positions. When no TCP is provided in the parameters, it returns the solution closest to the currently active TCP of the controller.

Parameters**pose**

Type	Pose
Default	–
Description	Pose to transform in joint positions

near_joint_angles (optional)

Type	np.array of 6 elements
Default	–
Description	Near joint positions If the near joint positions are not provided, the actual joint positions are used.

Returns

Value	Type
Inverse kinematic has solution	bool
Position of each of the joints	np.array of 6 elements

Example:

```
pose_1 = Pose(0.2, 0.5, 0.5, -180, 0, 180)
```

```
pose_2 = Pose(0.575, 0.125, 0.5, 0, -180, 90)

has_solution_1, joint_positions_1 = inverse_kin(pose_1)
# Inverse kinematics from near current joint positions

has_solution_2, joint_positions_2 = inverse_kin(pose_2, [0, 0, 90, 0, 0, 0])
# Inverse kinematics from near specific joint positions
```

is_robot_standstill()

Returns True when the speed of each joint is below 0.1 °/s; otherwise, returns false.

Parameters

none

Returns

Value	Type
is robot standstill	bool

Example:

```
robot_standing_still = is_robot_standing_still()
```

log_event(title, severity, explanation, suggestion)

Logs an event in the event logger.

Parameters

title

Type	str
Default	–
Description	Title of the event

severity (optional)

Type	str
Default	"info"
Description	Severity of the event
"info"	Information event
"warning"	Warning event

"error"

Error event.

The event triggers an error pop-up dialog in Kortex Web App.

explanation (optional)**Type**

str

Default

"N/A"

Description

Explanation of the event.

suggestion (optional)**Type**

str

Default

"N/A"

Description

Suggestion of the event

Returns

none**Examples:**

```
log_event("Information message")

log_event("Warning message", "warning", "Explanation about the warning",
" Suggestion about the warning")

log_event("Error message", "error", "Explanation about the error",
" Suggestion about the error")
```

stop_program()

Stops the program that is currently running.

Parameters

none

Returns

none

Example:

```
stop_program()
```

wait(seconds)

Waits the specified number of seconds.

Parameters**seconds**

Type	float
Default	–
Description	Time to wait in seconds

Returns

none

Example:

```
wait(5.2)    # Waits 5.2 seconds
```

USB storage

You can read or write files that are on a USB drive that is connected to the controller. The USB is mapped to the folder `plugin/shared/usb-storage`. The constant `USB_STORAGE` is defined in the library that contains the path to the folder.

```
USB_STORAGE = 'plugin/shared/usb-storage'
```

Use the `USB_STORAGE` constant in the Python script action to read or write files. Add the name of the USB drive to the USB storage path.

Example: The name of the USB drive is KINOVA. A file called `log_file.txt` is read.

```
from kinpyscript import *

usb_path = USB_STORAGE + 'KINOVA/'

file_stream = open(usb_path + 'log_file.txt', 'r')
lines = file_stream.readlines()file_stream.close
```

External communication

TCP Socket

Sockets are commonly used for client and server interaction. Typical system configuration places the server on one machine, with the clients on other machines. The clients connect to the server, exchange information, and then disconnect.

In the following example,

Example:

A socket connection is open to communicate with a camera at IP address 192.168.0.50 on port 10000. Once the client is connected to the camera, it is possible to send commands to the server.

A reset command is sent.

```
import socket

class Camera:
    def __init__(self, address, port):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((address, port))

    def __del__(self):
        self.socket.close()

    def send_cmd(self, string):
        self.socket.send(string.encode())
        data = self.socket.recv(1024)
        return data.decode()

if __name__ == '__main__':
    cam = Camera('192.168.0.50', 10000)
    status = cam.send_cmd("reset")
```

XmlRpc

XML-RPC is a Remote Procedure Call method that uses XML passed via HTTP as a transport. With it, a client can call methods with parameters on a remote server (the server is named by a URI) and get back structured data.

`xmlrpc` is a package that collects server and client modules implementing XML-RPC. The modules are:

- `xmlrpc.client`
- `xmlrpc.server`

Example:

A control box is used to control a screwdriver unit at the IP address 192.168.0.120 on port 41414. Once the client is connected to the control box, it is possible to send commands to the server.

A screw tightening command is sent.

```
import xmlrpc.client

control_box = xmlrpc.client.ServerProxy("http://192.168.0.120:41414/")

control_box.sd_tighten(0, 18, 22, 3, True)
```

Modbus

Pymodbus is a full Modbus protocol implementation, using a synchronous or asynchronous (using `asyncio`) core. Supported modbus communication modes:

- TCP
- RTU-over-TCP
- UDP
- Serial
- TLS

Example:

A client connects to a Modbus RTU network connected on port `"/dev/ttyUSB0"`. Once the client is connected to the network, two write commands are sent to slave ID 86.

```
from pymodbus.client.sync import ModbusSerialClient

port = "/dev/ttyUSB0"
```

```
_slave_id = 86

client = ModbusSerialClient(method="rtu", port=_port, stopbits=1,
                             bytesize=8, parity='E', baudrate=19200)

result = client.write_register(11, 4, unit=_slave_id)
result = client.write_register(10, 7000, unit=_slave_id)
```

Paho MQTT

The MQTT protocol is a machine-to-machine (M2M)/Internet of Things(IoT) connectivity protocol. Designed as an extremely lightweight, publish/subscribe messaging transport, it is useful for connections with remote locations where a small code footprint is required or where network bandwidth is at a premium.

Example:

A client connects to a MQTT server at IP address 192.168.0.120, port 1883. Once the client is connected, it prints the connection result code and subscribes to the topic **dashboard**. When a message is published by the server, the message is printed.

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))

    client.subscribe("dashboard")
    client.publish("on_connect", True)

def on_message(client, userdata, msg):
    print("Topic: {} / Message: {}".format(msg.topic,
        str(msg.payload.decode("UTF-8"))))

if __name__ == "__main__":
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect("192.168.0.120", 1883, 60)
    client.loop_forever()
```

Changes since the last release

Modifications were made from the previous version to the current version.

Changes made in version 1.1

Version 1.1 of the Link Toolkit Plugin benefits from minor bug fixes, increased robustness, and compatibility with the latest Kortex v3.3.0.

New features

Version 1.1 of Link Toolkit Plugin introduces event configuration and an action.

- Reactions to specific events can be configured from the **Configurations** page of Link Toolkit Plugin.
- The **Path Player** action can record complex assemblies and trajectories in Hand Guiding mode and replay them within a program.

Removal of features

The **Stop Program** action is no longer part of the Link Toolkit Plugin. It is part of the **Basic** robot action menu. The action has been renamed to **Exit**.

Fixes

The current release has fixes for several issues. The fixes are described as they were when they were still issues.

- After a reboot of Link 6, the events defined on the **Configurations** page of Link Toolkit do not work unless the plugin is deactivated and then reactivated.

Known issues

Kinova is working towards addressing issues in the plugin.

- During the **Seek** action, the arm may enter a singularity, causing an error dialog to launch. Regardless of the error dialog, the robot continues the program.
- When operating in **Hold to Run** mode at **Reduced** speed, the robot can exceed elbow speed limits.

Limitations

The speed in the **Seek** action is limited to reduced safety limits, regardless of operating mode.

Glossary

C

controller

the part of the robot from where the software operates

CPU

the portion of the computer that retrieves and performs instructions

central processing unit ([CPU](#))

E

end effector

device specifically designed for attachment to the flange of the wrist

event

an action that happens, and is recognized by the software or firmware

F

flange

a mechanical interface used for attaching another object

G

GPU

originally created to enhance graphic rendering, it also has proven itself useful in many other applications, such as artificial intelligence

graphics processing unit ([GPU](#))

GUI

a way of interacting with a program of a computer that involves windows, dialogs, icons, and menus

Graphical User Interface ([GUI](#))

synonym: *HMI*

N

namespace

a way in computer science of grouping together all objects that work together such that each object has a unique name

non-blocking

the ability for a robotic program to proceed to its next step even when the current action is not finished

synonym: *synchronous operation*

P

pendant

a device used to configure and control the movements of a robot

synonym: *teach pendant*

plugin

software that is added to an pre-existing base software so as to give the application additional capabilities

R

RAM

the memory of the computer that stores data for the *central processing unit (CPU)* to use
random access memory (*RAM*)

reaction

an action taken in response to an event

robot

teach pendant, controller, arm, software, and firmware; it may also include an end effector

T

teach pendant

a device used to configure and control the movements of a robot

synonym: pendant

U

USB device

industry computer bus standard that makes it easier to transfer data between devices

Universal Serial Bus ([USB](#))

USB storage device

a portable storage device that plugs into [USB device](#) ports

synonym: thumb drive, flash drive

W

Kortex Web App

the [GUI](#) that runs on a web server on the [controller](#) and is accessed on the [teach pendant](#) or on an external computer

synonym: Web App

wrist

the part found at the end of the robot arm to which end effectors may be attached

TOGETHER IN ROBOTICS



PROUDLY MADE IN

📍 QUEBEC, CANADA

EN-UG-025-r2.0-202312

KINOVA

kinovarobotics.com | info@kinovarobotics.com | +1 (855) 654-6682

© Kinova inc 2023. All rights reserved.

¹Intel, the Intel logo, and Intel Core are trademarks of Intel Corporation or its subsidiaries.