



AX Portal User's Guide

Original instructions

©Seiko Epson Corporation 2026

Rev. 2.0
ENM265S8833M

Table of Contents

1. Introduction.....	10
1.1 Programming & Controlling.....	11
1.2 Graphical User Interface (GUI).....	11
1.3 Software Modules.....	11
1.3.1 TCP Connection Module.....	11
1.3.2 REST Connection Module.....	11
1.3.3 ROS Connection Module.....	12
1.3.4 Modbus Connection Module.....	12
1.3.5 Panel Interface Module.....	12
1.3.6 Cognex Camera Module.....	12
2. Admin Interface (GUI).....	13
2.1 Introduction.....	14
2.1.1 Interface Language.....	15
2.1.2 Dashboard.....	15
2.2 User Management.....	15
2.2.1 User Levels.....	15
2.2.2 Login Page.....	15
2.2.3 Manage My Account.....	16
2.2.4 Manage Other Users.....	17
2.3 System Settings.....	18
2.3.1 Network Settings.....	18
2.3.2 Time Settings.....	19
2.3.3 SSL Certificate.....	19
2.4 Software Update.....	21
2.5 Help & Support.....	23
2.6 Secure Authentication.....	24
2.6.1 Export SSL Certificate.....	24
2.6.2 User Login.....	24
2.6.3 Connection Registration.....	25
3. USB Export and Recovery Update.....	26
3.1 Overview.....	27
3.2 USB Export.....	27
3.3 Run Recovery Update.....	27
4. Desktop Interface (GUI).....	28
4.1 Introduction.....	29
4.2 Main Window.....	30
4.2.1 Menu Overview.....	30
4.2.2 Cockpit.....	31
4.2.3 Active Control.....	33

4.2.3.1 Active Control Station.....	33
4.2.3.2 Passive Observer.....	34
4.2.4 Program Status.....	34
4.2.5 Operation Modes.....	36
4.2.5.1 Automatic.....	36
4.2.5.2 Manual.....	36
4.2.6 Jogging Controls.....	37
4.2.7 Motion Controls.....	38
4.2.7.1 Motion Limits.....	38
4.2.7.2 Global Parameters.....	39
4.2.7.3 Global Frames.....	40
4.3 Building Blocks & Database.....	41
4.4 Project Menu.....	41
4.5 Configuration Menu.....	43
4.5.1 Database (Download & Upload).....	43
4.5.2 I/O Configuration.....	44
4.5.3 Tool Configuration.....	47
4.5.4 Tool Editor.....	47
4.5.4.1 Single Output Mode.....	49
4.5.4.2 Dual Output Mode.....	49
4.5.4.3 Separate Trigger Mode.....	49
4.5.5 Robot Power-Up.....	50
4.6 Teach-In Menu.....	51
4.6.1 Pose Editor.....	53
4.6.2 Path Editor.....	55
4.6.3 Grid Editor.....	57
4.7 Application Menu.....	60
4.7.1 Application Editor.....	61
4.7.2 Application Code.....	61
4.7.3 Application Settings.....	62
4.7.4 Application Variables.....	64
4.7.5 Shared & Global Variables.....	65
4.7.6 Folder Editor.....	66
4.7.7 Block Programming.....	67
4.7.7.1 Using Application Variables and Expressions.....	67
4.7.7.2 Move Block.....	68
4.7.7.3 Path Block.....	69
4.7.7.4 Pick Block.....	69
4.7.7.5 Place Block.....	69
4.7.7.6 Container Block.....	69
4.7.7.7 Next Grid Pose Block.....	69

4.7.7.8 Reset Grid Pose Block.....	70
4.7.7.9 Set Variable Block.....	70
4.7.7.10 Set Output Block.....	70
4.7.7.11 Wait Block.....	70
4.7.7.12 Loop Block.....	70
4.7.7.13 Exit If Block.....	71
4.7.7.14 Condition Block.....	71
4.7.7.15 Comment Block.....	71
4.7.7.16 Quit Block.....	71
4.7.7.17 Dialog Block.....	71
4.7.7.18 Print Block.....	71
4.7.7.19 Log Block.....	71
4.7.7.20 Notify Block.....	71
4.7.7.21 Warning Block.....	71
4.7.7.22 Error Block.....	71
4.7.7.23 Python Code Block.....	72
4.8 I/Os Menu.....	72
4.9 Visualization Menu.....	73
4.10 Hand-Guided Control.....	74
4.11 System Monitor.....	75
4.11.1 Robot State.....	75
4.11.2 Application Output.....	75
4.11.3 Application Log.....	75
4.11.4 Message History.....	75
4.12 Help Menu.....	76
4.12.1 Robot Information.....	76
4.12.2 Keyboard Shortcuts.....	76
4.12.3 Code Documentation.....	76
4.12.4 Open Source License.....	76
4.12.5 Contact Information.....	76
4.13 Settings Menu.....	77
4.13.1 GUI Settings.....	78
4.13.2 Customize Cockpit Buttons.....	78
4.13.3 Start-Up Settings.....	79
4.13.4 Motion Behaviors.....	79
4.13.4.1 Interrupt Behavior.....	79
4.13.4.2 Interrupt Dialog.....	80
4.14 Safety Settings Menu.....	82
4.14.1 User Interface.....	82
4.14.2 Safety Settings.....	84
4.14.2.1 Safely Limited Position (SLP).....	84

4.14.2.2 Safely Limited Joint Angles.....	85
4.14.2.3 Safely Limited Speed (SLS).....	86
4.14.2.4 Safe Power and Force Settings.....	87
4.14.2.5 Safety I/Os and Operator Limitations.....	88
4.14.2.6 Verification Data.....	89
4.14.2.7 Safety Output Functions.....	89
4.14.2.8 Connecting Optional Devices.....	89
5. Panel Interface (GUI).....	90
5.1 Introduction.....	90
5.2 Configure Applications for Panel Interface.....	92
5.3 Cockpit and Settings.....	94
5.4 Controlling Applications.....	95
6. Kinematic Definitions.....	96
6.1 Coordinates.....	97
6.1.1 Nautical Angles.....	97
6.1.2 Coordinate Systems.....	98
6.2 Configuration.....	99
6.3 Pose.....	99
6.4 Point.....	99
6.5 Path.....	99
6.6 Segment.....	100
6.7 Grid.....	100
7. Connection Modules.....	101
7.1 Connections, Authentication, and Active Control.....	102
7.1.1 Connection Channels.....	102
7.1.2 Authentication.....	102
7.1.3 Active Control Station.....	103
7.2 TCP Connection Module.....	103
7.2.1 Establishing a Connection.....	103
7.2.2 Sending Commands.....	105
7.2.3 Settings.....	106
7.2.3.1 Encoding Standard.....	106
7.2.3.2 End Character.....	106
7.2.3.3 Send Heartbeat.....	106
7.2.4 Message Protocol - Receiving Commands.....	107
7.2.4.1 Status Info.....	107
7.2.4.2 Action Return Value.....	108
7.2.4.3 Error.....	108
7.2.4.4 Message History.....	108
7.2.4.5 State Group Define.....	109
7.2.4.6 State Group Data.....	109

7.2.4.7 Open Dialog.....	110
7.2.4.8 Script Send.....	110
7.2.5 Message Protocol – Sending Commands.....	111
7.2.5.1 Token Login.....	111
7.2.5.2 Request Active Control.....	111
7.2.5.3 Release Active Control.....	112
7.2.5.4 Request Data.....	112
7.2.5.5 Power Up.....	112
7.2.5.6 Power Down.....	113
7.2.5.7 Pause.....	113
7.2.5.8 Resume.....	113
7.2.5.9 Enabling Hand-Guided Control (HGC).....	114
7.2.5.10 Disabling Hand-Guided Control (HGC).....	114
7.2.5.11 Stop.....	115
7.2.5.12 Change Project.....	115
7.2.5.13 Execute Script.....	116
7.2.5.14 Test Script.....	116
7.2.5.15 Script Receive.....	117
7.2.5.16 Move Pose.....	117
7.2.5.17 Play Path.....	118
7.2.5.18 Toggle Tool.....	118
7.2.5.19 Write Digital & Analog Outputs.....	119
7.2.5.20 Answer Dialog.....	120
7.2.5.21 Set State Variable.....	120
7.2.5.22 Set Global Variable.....	121
7.2.5.23 Get Global Variable.....	121
7.2.5.24 Set Shared Variable.....	122
7.2.5.25 Get Shared Variable.....	122
7.3 REST Connection Module.....	123
7.3.1 Establishing a Connection.....	123
7.3.2 Sending Commands.....	124
7.3.3 Settings.....	124
7.4 ROS Handler Module.....	125
7.4.1 Installation.....	125
7.4.2 Configuration.....	126
7.4.3 Network Discovery with Cyclone DDS.....	126
7.4.3.1 Network Interface.....	126
7.4.3.2 Discovery Methods.....	127
7.4.4 Establishing a Connection.....	129
7.4.5 Access Robot Data Through ROS.....	130
7.4.6 ROS 2 Functionalities in Scripts.....	130

7.4.7 Call an Action.....	130
7.4.8 Authentication Settings.....	131
7.5 Modbus Connection Module.....	132
7.5.1 Overview.....	132
7.5.2 Settings.....	132
7.5.3 Writing Registers (Simple Description).....	133
7.5.3.1 Command Registers.....	133
7.5.3.2 Property Registers.....	134
7.5.3.3 User Writing Registers.....	134
7.5.4 Writing Registers (Detailed Description).....	135
7.5.4.1 Do Nothing.....	135
7.5.4.2 Power Up.....	135
7.5.4.3 Power Down.....	135
7.5.4.4 Stop.....	135
7.5.4.5 Pause.....	135
7.5.4.6 Resume.....	136
7.5.4.7 Calibrate.....	136
7.5.4.8 Enable Hand-Guided Control (HGC).....	136
7.5.4.9 Disable Hand-Guided Control (HGC).....	136
7.5.4.10 Change Project.....	136
7.5.4.11 Start Script.....	137
7.5.4.12 Move Pose.....	137
7.5.4.13 Play Path.....	137
7.5.4.14 Tool Place.....	137
7.5.4.15 Tool Pick.....	137
7.5.4.16 Write Digital Outputs.....	138
7.5.4.17 Set Script Events.....	138
7.5.4.18 Acknowledge Error.....	138
7.5.4.19 Read Position.....	138
7.5.4.20 Request Active Control.....	138
7.5.4.21 Release Active Control.....	139
7.5.4.22 Switch to Manual Mode.....	139
7.5.4.23 Switch to Automatic Mode.....	139
7.5.4.24 Reset Safety.....	139
7.5.4.25 Start Motion Interrupt.....	139
7.5.5 Reading Registers.....	140
7.5.5.1 Status Registers.....	140
7.5.5.2 User Reading Registers.....	143
7.5.5.3 Position Registers (Joints 1 through 6).....	143
7.5.5.4 Coordinate Register (x, y, z, roll, pitch, yaw).....	143
7.5.6 Error Register Mapping.....	144

7.5.6.1 Example.....	144
7.5.7 Python Code Examples.....	145
7.5.7.1 Example 1.....	145
7.5.7.2 Example 2.....	145
7.5.7.3 Example 3.....	145
7.5.7.4 Example 4.....	146
8. Cognex Camera Module.....	147
8.1 Overview.....	148
8.2 Camera Setup.....	148
8.3 Camera Application.....	148

Trademarks

Cognex® and In-Sight Explorer® are trademarks or registered trademarks of Cognex Corporation in the United States and other countries.

Google Chrome™ and Chrome™ are trademarks of Google LLC.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Apple®, macOS® and Safari® are registered trademarks of Apple Inc., in the U.S. and other countries and regions.

Microsoft®, Windows®, Microsoft Edge®, and Microsoft Visual C++® are trademarks (or registered trademarks) of Microsoft Corporation and/or its affiliates in the United States and other countries.

Mozilla and Firefox® are trademarks of the Mozilla Foundation in the U.S. and other countries.

General Notice: Other product names used herein are for identification purposes only and may be trademarks of their respective owners. Epson disclaims any and all rights in those marks.

All screenshots are used for illustrative and educational purposes only. No affiliation with or endorsement by any trademark holder is intended or implied.

Terms of Use

No part of this instruction manual may be reproduced or reprinted in any form without express written permission.

The information in this document is subject to change without notice.

Please contact us if you find any errors in this document or if you have any questions about the information in this document.

1. Introduction

1.1 Programming & Controlling

The software framework AX Portal provides the user of an Epson AX series robot with a very intuitive way to communicate with this robot. Using AX Portal, the user has full access to manipulate and control the robot as well as writing program applications using the programming language **Python 3.13**. The basic functionality for programming with Python can be found online. Some very basic code examples however are also provided in the built-in code documentation.

In addition to a huge variety of basic Python functions, AX Portal applications support a large set of functions specifically designed for controlling, maintaining, and communicating with AX robots. The complete list of scripting functions supported by AX Portal can be found in the built-in code documentation.

1.2 Graphical User Interface (GUI)

The GUI has been designed as a web application that is able to run on common web browsers of basically any device. The following browsers are officially supported:

- Google Chrome (version ≥ 147)
- Mozilla Firefox (version ≥ 150)
- Apple Safari (version ≥ 26)
- Microsoft Edge (version ≥ 147)

Therefore, there is no need to download a specific software module for your controlling device, as long as the robot and device are in the same network or share a direct communication channel. In addition to the default [Desktop Interface](#) of AX Portal, which is mainly optimized for programming on desktop computers and laptops, AX Portal provides a [Panel Interface](#), which comes with only the necessary functions for operating an already pre-programmed robot. You will find more information about the GUIs and all of their supported functions further down in this document.

1.3 Software Modules

In addition to the basic functionality of AX Portal, which is explained in detail throughout this document, some built-in software modules with additional control possibilities are available on AX robots. Please note that all [Connection Modules](#) as mentioned below provide a possibility to connect remotely to the AX robot and therefore provide possibilities for controlling and observing the robot.

1.3.1 TCP Connection Module

The [TCP Connection Module](#) can be used as an alternative way to communicate with the robot. The robot is usually controlled by the user through the GUI in the web browser. However, if the robot is supposed to be controlled by a custom interface or by another machine, it directly accepts TCP commands. With the TCP connection module, everything that can be achieved by the browser, can also be achieved via TCP.

1.3.2 REST Connection Module

The [REST Connection Module](#) can be used as an alternative way to communicate with the robot. The robot is usually controlled by the user through the GUI in the web browser. However, if the robot is

supposed to be controlled by a custom interface or by another machine, it directly accepts REST commands. With the REST connection module, everything that can be achieved by the browser, can also be achieved via REST.

1.3.3 ROS Connection Module

The [ROS Connection Module](#) provides functionality to integrate the robot into a ROS2 environment. The robot allows for other clients in the ROS2 network to send actions for controlling and monitoring the robot.

1.3.4 Modbus Connection Module

The [Modbus Connection Module](#) can be used as an alternative way to communicate with the robot. It uses the Modbus TCP protocol and lets the user read and write standard Modbus registers. Use this module to directly control the robot via a PLC. The most important functionalities such as powering the robot, running applications and triggering outputs can be achieved by the Modbus module.

1.3.5 Panel Interface Module

The [Panel Interface Module](#) provides a graphical user interface with limited functionalities for robot operators rather than programmers. This interface mainly allows to launch and monitor applications, and its content can be modified by the programmer to fit your use case. The panel interface is touch screen compatible and its design resembles the well-known app-like structure of state-of-the-art mobile devices, making it intuitive even for users without technical background.

1.3.6 Cognex Camera Module

The [Cognex Camera Module](#) provides a communication interface between a Cognex camera and AX Portal. Cameras are usually either integrated into an end-effector, or mounted stationary observing the scene in which the robot operates. This module is perfect for customizable visual inspection, supporting the full function range of Cognex cameras.

2. Admin Interface (GUI)

2.1 Introduction

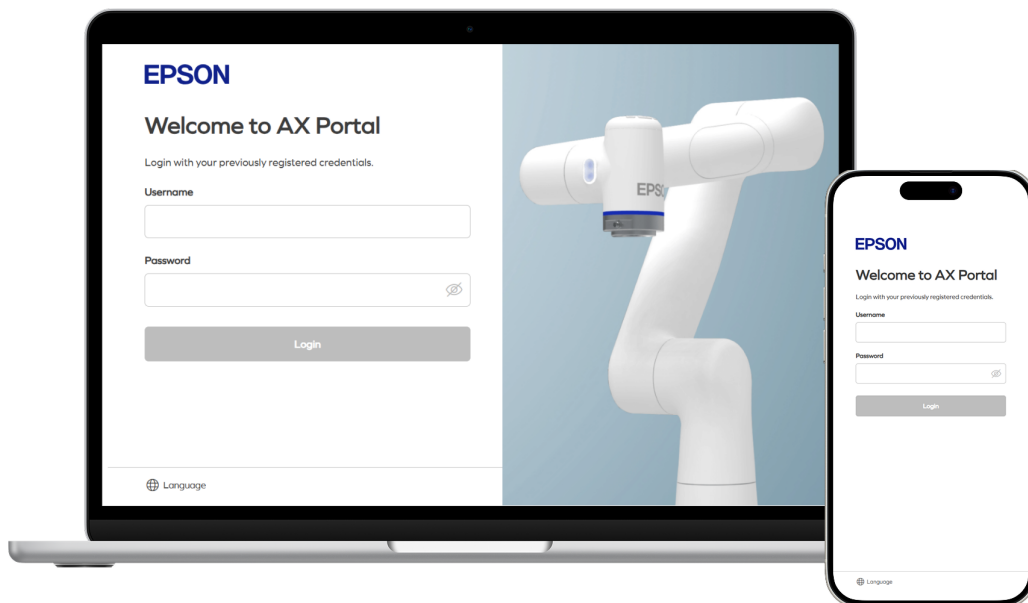


Figure 2.1.a: Example Screenshots of the Admin Interface

The Admin Interface of AX Portal is a graphical user interface (GUI) that allows users to manage user accounts and change some initial settings that are bound to the operating system the robot runs (like network settings, time settings, etc.). These settings are usually set just once on robot installation, before a programmer even starts to write robot control applications. This interface can be accessed via web browser (e.g. Google Chrome) by entering the IP address of the robot in the URL of that browser, which will forward you to `https://<IP-ADDRESS>/admin` (refer to the “Robot Controller RC-A101 Manual” for the correct IP address).

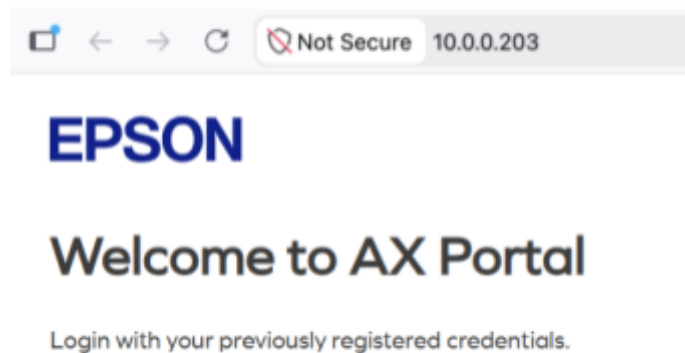


Figure 2.1.b: Connect to the Admin Interface via URL

In this chapter, all menus and elements of the Admin Interface are explained in detail.

2.1.1 Interface Language

To change the interface language, click on the Language dropdown on the left bottom corner of the interface and select your preferred language. Your browser will simplify your life by remembering your language setting across sessions and interfaces.

2.1.2 Dashboard

Once logged in, the dashboard of the Admin Interface allows you to navigate to the different submenus, to navigate to other interfaces of AX Portal, and to logout. Please note that the dashboard only shows the options that you currently have access to, depending on your user level (administrator, safety officer, programmer, operator). If you cannot navigate to other interfaces, then AX Portal is currently not properly running, which e.g. is the case while a software update is in progress.

2.2 User Management

2.2.1 User Levels

In order to prevent the robot from being configured, programmed, or operated by unauthorized users, we introduce different authentication levels. This user level for any user is defined by the administrators only. The following user levels are available, from highest to lowest authentication level:

- An **administrator** has full access to all robot control interfaces and functionalities, has access to manage other users as well as to the most important system settings of the operating system.
- A **programmer** has access to all robot control interfaces and functionalities and can therefore prepare and program applications for operators to be executed later on. Within the Admin Interface, the programmer basically has only access to his own account details, and to navigation between interfaces.
- A **safety officer** is a programmer (see above) that is allowed to change the safety settings of the robot. Other users are just allowed to observe the safety settings.
- An **operator** has access only to the basic operator interfaces, which allows for operating the robot but not changing any of the settings or programs. Within the Admin Interface, the operator basically has only access to his own account details.

2.2.2 Login Page

The login page contains two different forms, depending on whether an administrator account has already been created.

- If an administrator has already been created, the usual login page appears where you need to enter your username and password to login. Passwords are hidden while you enter them, unless you click on the show password button inside the password field.
- If an administrator has not yet been created, you are directly asked to do so, and you cannot continue until you create an administrator account. Refer to the "Collaborative Robot: 6-Axis Robots AX6 Quick Start / Setup Manual" for details. You can only define the absolute necessary data here, if you want to see and define all settings for your account, check out the 'Manage My Account' menu.

After each logout you will be redirected to the login page.

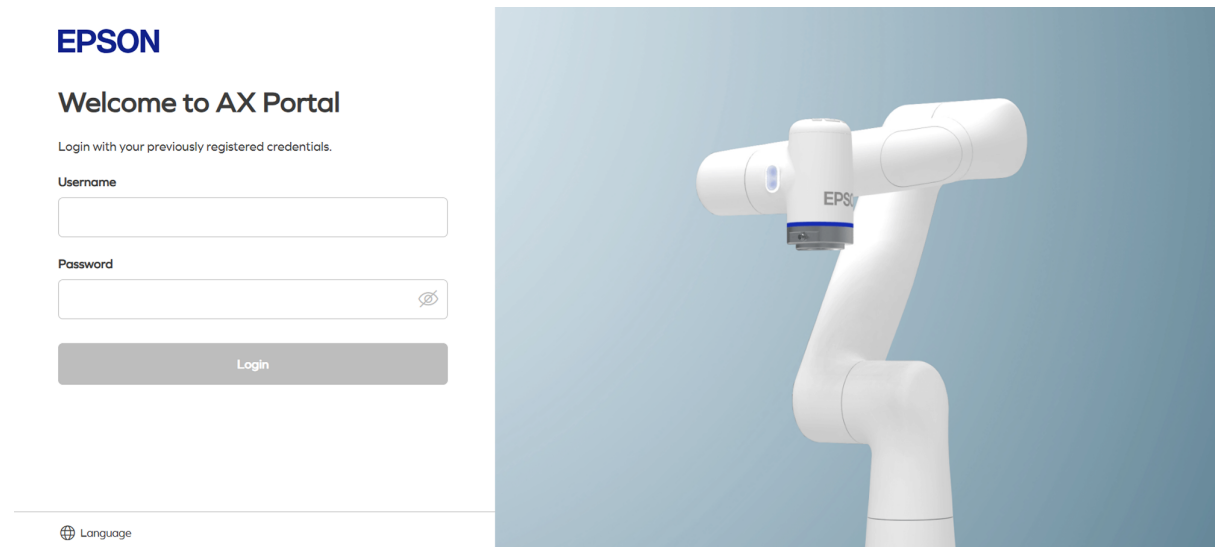


Figure 2.2.2: Login Page

2.2.3 Manage My Account

In the 'Manage My Account' menu you may change your own profile, more specifically your name, password, profile picture, and the start-up interface:

- The 'full name' is the name that is used to address you in the interface and in messages. It is a mandatory field intended for your first and last name, but you may also define your own naming rules.
- The 'username' field cannot be changed after a user has been created, the administrator defines the username on user creation.
- The 'start-up interface' field defines which interface you want to be forwarded to after logging in. Different user levels have different default start-up interfaces predefined, which you may change here.
- The 'change password' button allows you to change your account password. It will ask you to re-enter your current password and enter your new password twice, to minimize accidental errors on password definition. Your password can also be changed by any administrator, in case you forget it. Passwords are never saved in plain text, hence neither you nor any other user will be able to extract your password.
- Changing the 'profile picture' is optional. It will display a default user icon if you do not define a profile picture.

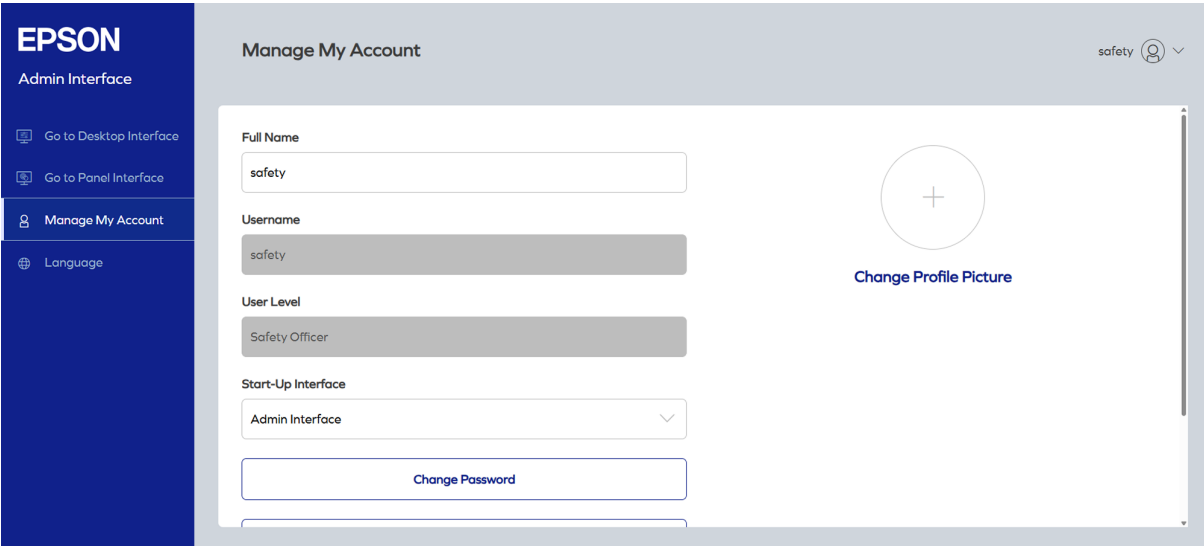


Figure 2.2.3: Manage My Account Page

2.2.4 Manage Other Users

In the 'Manage Other Users' menu you will see an overview of all currently available users. You have the following options here:

- Adding a new user allows you to create a user of predefined user level and predefined initial password. The initial password should always be changed by any user after the first login.
- Modifying a user allows you to change most of the properties of that user, except for the username, which cannot be changed at all.
- Deleting a user removes it permanently from the database. This user won't be able to trigger robot control functions anymore.

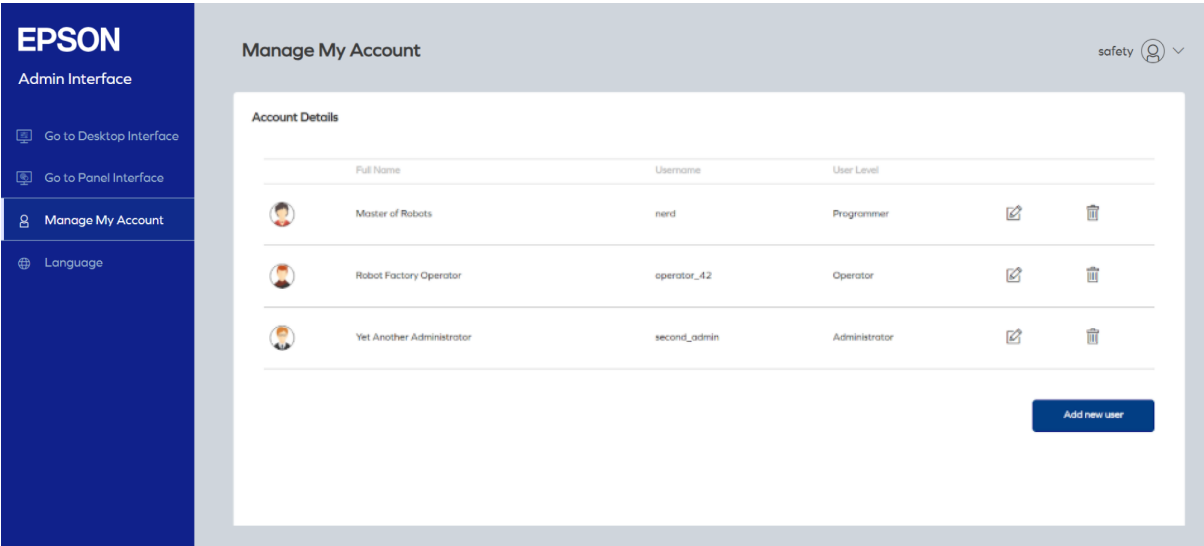


Figure 2.2.4: Manage Other Users Page

2.3 System Settings

2.3.1 Network Settings

The 'Network Settings' menu contains all relevant settings to define the Ethernet connection of the robot. Be aware that once you change settings of the currently active connection (e.g. you are connected via Ethernet and you change the Ethernet IP address), you will lose the connection. In case you accidentally enter invalid settings, you may no longer be able to connect to the robot afterwards. Therefore it is strongly recommended that you make sure that two different connections are working, before you change the settings of the only working connection.

The Ethernet settings category shows you and lets you decide whether the robot uses a dynamic or a manual IP address. In case the robot is configured to dynamically retrieve an IP address, you will see the selected IP address here as well. In case the robot is configured manually, you need to enter an IP address, the netmask (mandatory), and you may in addition define the gateway and up to two DNS servers (optional).

The screenshot shows the EPSON Admin Interface with a sidebar on the left containing navigation links: Go to Desktop Interface, Go to Panel Interface, Manage My Account, Manage Other Accounts, Network Settings (highlighted), Time Settings, SSL Certificate, Software Update, Help & Support, and Language. The main content area is titled 'Network Settings' and features a 'User' dropdown in the top right. The 'Ethernet IPv4' section includes a form with the following fields: IP Address (192.168.90.200), Dynamic IP Address (selected with a blue radio button), Manual IP Address (unselected with a white radio button), IP Address (text input with 192.168.90.200), Netmask (text input with 255.255.254.0), Gateway (text input with 192.168.90.1), DNS Server 1 (text input), and DNS Server 2 (text input). Below this is a 'Network Information' section with a note about using the second Ethernet port for maintenance. At the bottom are 'Cancel' and 'Save' buttons.

Ethernet IPv4	
IP Address	192.168.90.200
Dynamic IP Address	☒
Manual IP Address	☐
IP Address	192.168.90.200
Netmask	255.255.254.0
Gateway	192.168.90.1
DNS Server 1	
DNS Server 2	

Network Information

You may use the second Ethernet port to access AX Portal for network maintenance reasons. Its static IP address is 10.0.0.203 and cannot be changed by the administrator. You may connect to '<Robot-Serial>.local' instead of entering the IP address to connect to AX Portal via your browser.

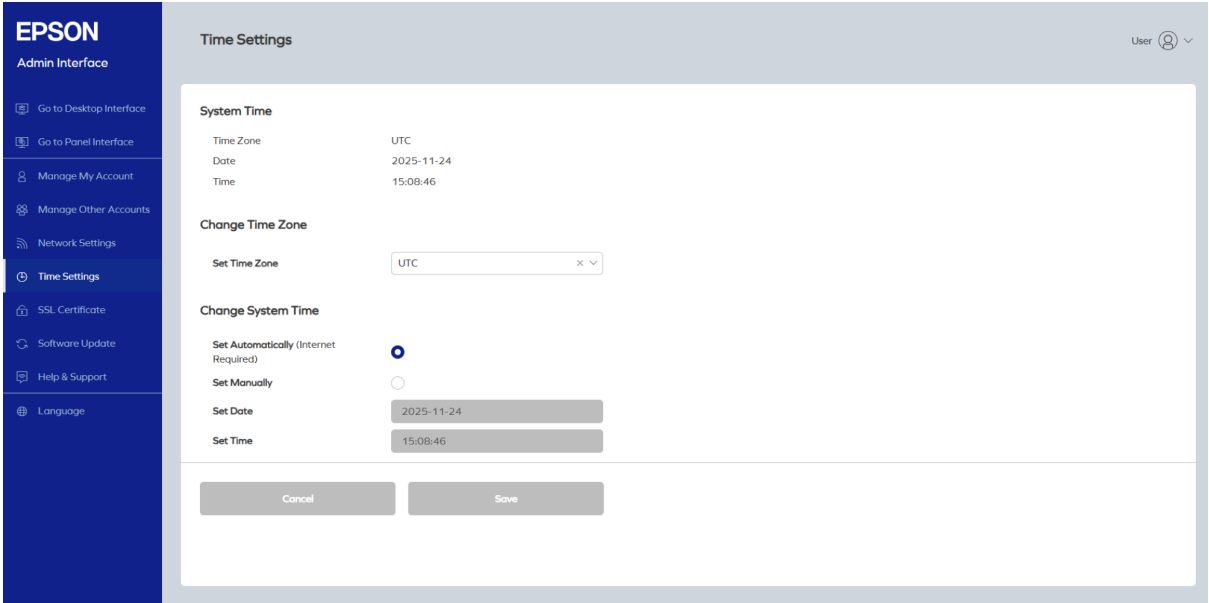
Cancel Save

Figure 2.3.1: Network Settings Page

2.3.2 Time Settings

The 'Time Settings' menu shows the current date, time, and timezone of the robot's operating system. Make sure to enter the correct time and date. By setting the correct time, the system can be used for log analysis and troubleshooting, and time-dependent functions can operate properly. If changing the time manually, the exact time is applied immediately after hitting the save button. If you want to change the timezone, you may filter the system's list of available timezones by continent (e.g. "Europe"), country code (e.g. "GB"), bigger city (e.g. "London"), or a combination of these (e.g. "eur lon" → Europe/London).

Note: If the robot has network access, you may enable the NTP (network time protocol) synchronization.



The screenshot displays the 'Time Settings' page within the EPSON Admin Interface. The left sidebar contains navigation links: 'Go to Desktop Interface', 'Go to Panel Interface', 'Manage My Account', 'Manage Other Accounts', 'Network Settings', 'Time Settings' (highlighted), 'SSL Certificate', 'Software Update', 'Help & Support', and 'Language'. The main content area is titled 'Time Settings' and includes a 'User' profile icon in the top right. The 'System Time' section shows 'Time Zone' as UTC, 'Date' as 2025-11-24, and 'Time' as 15:08:46. Below this, the 'Change Time Zone' section has a 'Set Time Zone' dropdown menu currently set to UTC. The 'Change System Time' section features two radio buttons: 'Set Automatically (Internet Required)' which is selected, and 'Set Manually'. Under the manual settings, 'Set Date' is 2025-11-24 and 'Set Time' is 15:08:46. At the bottom, there are 'Cancel' and 'Save' buttons.

Figure 2.3.2: Time Settings Page

2.3.3 SSL Certificate

The 'SSL Certificate' menu shows you some details about the currently loaded SSL certificate, and allows you to replace this certificate with your own private or business certificate. A SSL (Secure Sockets Layer) certificate is a digital certificate that authenticates a website's identity and enables an encrypted connection. This certificate keeps online transactions and customer information in your web browser private and secure. By default, a self signed certificate is installed which is recommended to be replaced by a certificate fitting to the user's environment and is signed by a trusted authority.

A certificate will be marked as insecure by browsers if

- the domain name/IP registered in the certificate does not match the domain/IP used for accessing the robot,
- the certificate is expired or
- the certificate is self signed.

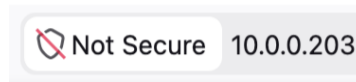


Figure 2.3.3.a: Example of Browser Complaining about Insecure or Missing SSL Certificate

In order to replace the SSL certificate, you need to upload both the private and public key file pair.

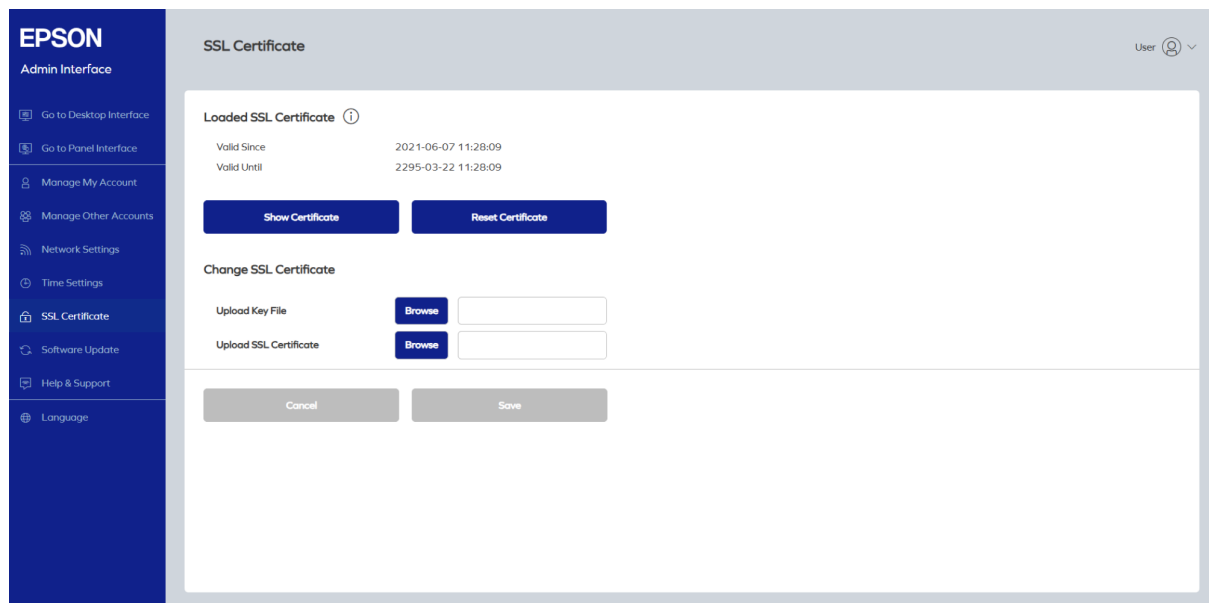


Figure 2.3.3.b: SSL Certificate Settings Page

2.4 Software Update

In the 'Software Update' menu you see the currently running version of AX Portal. You may upload and install new software versions here. A software update may be required to fix a specific issue, or to get the latest features and bug fixes.

In any case, before updating the software version, please always study the release notes first, for it contains important information about potential risks of updating to specific versions. Improvements sometimes come at the cost of deprecating older features or functionalities. We recommend you back-up your database, which contains all of your robot applications and settings (via the [Desktop Interface](#)) before triggering an update.

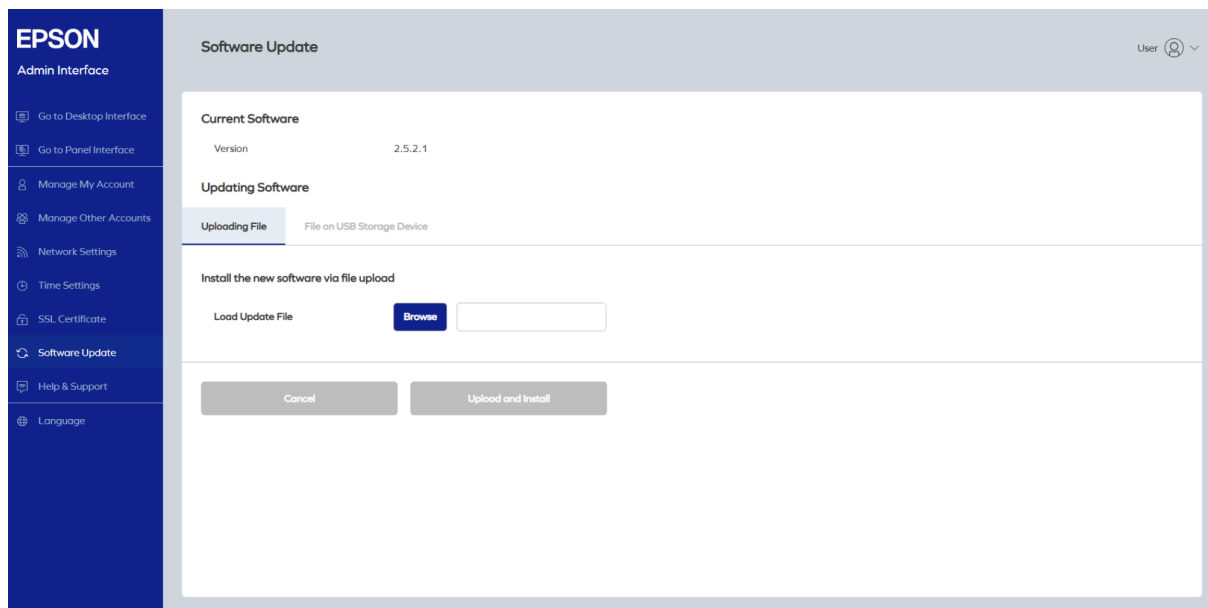


Figure 2.4.a: Software Update Page

In order to install a software update, you first need to acquire the RBIN file containing the update for the AX Portal version that you want to install. There exist three ways to install this update:

- 1) Select the first tab "Uploading File". Here you may upload the update from your device. Browse the file system and select the update RBIN file, then continue installing the update.
- 2) Select the second tab "File on USB Storage Device". Here you may put the RBIN file onto a USB stick and mount it to the robot's controller. Select the update RBIN file from the USB stick directory, then continue installing the update.
Note: currently, the filesystems exFAT and FAT32 are supported for the USB stick.
- 3) If options 1) and 2) are not applicable (e.g. the Admin Interface does not run properly on your robot), you always have the option to run the [Recovery Update](#) instead.

While an update is in progress, you see which state of the update is currently in progress, which states have been successfully completed, and in case of an error in which state the error occurred, together with the error message. An update will take **up to 15 minutes** depending on which packages are being updated. So please be patient and do not interrupt the update by e.g. cutting the robot's power. This may in the worst case result in you no longer being able to access the robot afterwards.

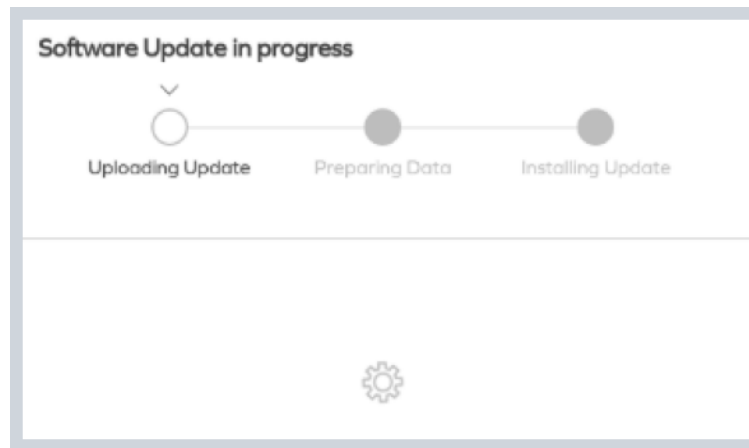


Figure 2.4.b: Update in Progress Window

After an update is completed successfully, the update must be confirmed. Until the update is confirmed, a red message bar will indicate this. Rebooting the controller will reset the update if it was not confirmed. Once confirmed, rolling back to an earlier version will no longer be possible.

2.5 Help & Support

The 'Help & Support' menu shows the contact information you may use to contact Epson in case you need help. Please make sure to study the documentation first before asking for support via eMail or Phone.

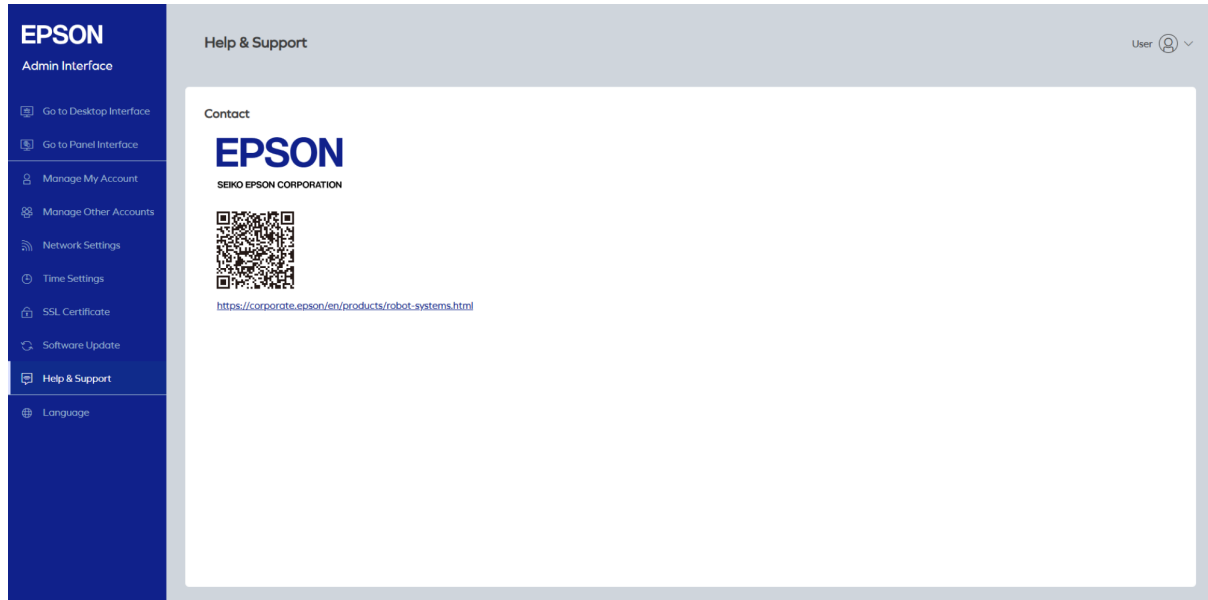


Figure 2.5: Help & Support Page

2.6 Secure Authentication

If you are using [Connection Modules](#) to access AX Portal to control the robot (without GUI), you need to authenticate using a secure two-step procedure before actually controlling the robot.

2.6.1 Export SSL Certificate

In the first step, you need to send a login request to the built-in authentication server. If you want to download the currently loaded SSL certificate, you can do so by exporting the certificate via your browser. The following figure shows how this may be done using Google Chrome. Other browsers support similar possibilities.

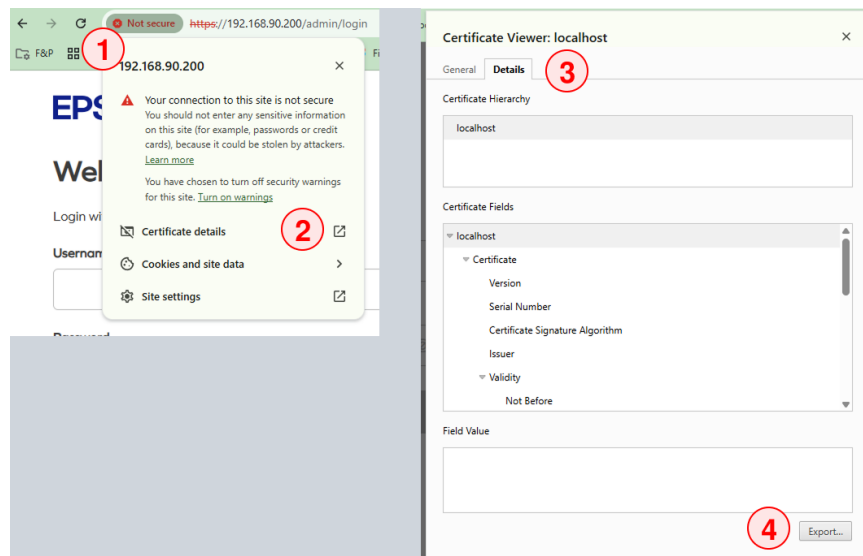


Figure 2.6.1: Exporting the SSL Certificate

1. View Site Information, 2. Go to Certificate Details, 3. Go to Details in Certificate Viewer, 4. Export Certificate

2.6.2 User Login

In the first step, you need to send a login request to the built-in authentication server, which is reachable via **TCP** at **port 5001**. To do this, you need to know the **IP address** of the robot, your login credentials (**username** and **password**), and you need to use the same **SSL certificate** as the robot uses. Here is a Python example that sends the login request via TCP socket and returns the access token. You may use other programming languages to do this.

Note: As of now, login is only possible through TCP. If you want to control your robot through one of the connection modules other than TCP (e.g. ROS/REST/Modbus), you still need to use TCP to acquire the access token. Alternatively, you may disable the authentication for these connection modules in the settings of AX Portal. Beware: If you disable the authentication, any client is able to control the robot via ROS/REST, without prior authentication!

```
import json, socket, ssl

# define the robot ip address, login credentials, and path to SSL certificate
ip_address = "10.0.0.203"
username = "user_xyz"
password = "my_fancy_password"
session_id = "my_session_id"
ssl_cert = "/home/user/robot-ssl/cert.pem"

# create request to send
request = {"request": "authentication", "action": "login", "data":
           {"username": username, "password": password, "session_id": session_id}}

# prepare SSL context to use with the connection
ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
ssl_context.load_verify_locations(ssl_cert)
ssl_context.check_hostname = False
ssl_context.verify_mode = ssl.CERT_NONE

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    with ssl_context.wrap_socket(sock, server_hostname=ip_address) as ssock:
        # connect to the authentication socket
        ssock.connect((ip_address, 5001))
        # send request (and end character)
        message = json.dumps(request) + "\x03"
        ssock.send(message.encode("UTF-8"))
        # receive answer (before end character)
        message = ssock.recv(1024).decode("UTF-8")
        answer = json.loads(message.split("\x03")[0])

token = answer["response"]["data"]["token"]
session_id = answer["response"]["data"]["session_id"]
```

2.6.3 Connection Registration

Once you have a **valid access token** at hand, you may use it in the “token_login” function, whose usage is described in detail in the chapters dedicated for the [connection module](#).

3. USB Export and Recovery Update

3.1 Overview

If the [Admin Interface](#) cannot be reached even though the controller is running, you can attempt to recover the controller using the recovery update method.

3.2 USB Export

The system log and AXPortal database can be exported without accessing the web interface by using a USB stick:

1. Format a USB stick with exFAT, FAT32 or NTFS and make sure it does NOT carry the file system label LOG or RECOVERY
2. Plug it into the controller and wait for a few minutes.
3. Remove the USB stick. You should find in the root of the stick a zip file containing the database and the relevant system logs.

3.3 Run Recovery Update

To run the recovery software update, please follow these steps:

1. Get the **update file**, which is usually in RBIN format. If you do not have an update file, please contact the supplier.
2. Make sure you have a **USB stick** at hand that you can use for the update. It should be formatted with either exFAT, FAT32 or ntfs and the file system of the first partition has to be labeled "RECOVERY".
3. Load the update file onto the USB stick, into its root directory.
4. Power down and reboot the robot. Wait **at least 1 minute** for the robot to boot properly.
5. Plug the prepared USB stick into any USB port of the controller.
6. Wait for the update to finish. Note that this may take **up to 15 minutes**. Try to access the [Admin Interface](#) and inspect whether the new version has been installed.
7. **Important:** Unmount the USB stick from the control box. If you forget this step, the update will run again after the reboot, which you do not want.

4. Desktop Interface (GUI)

4.1 Introduction

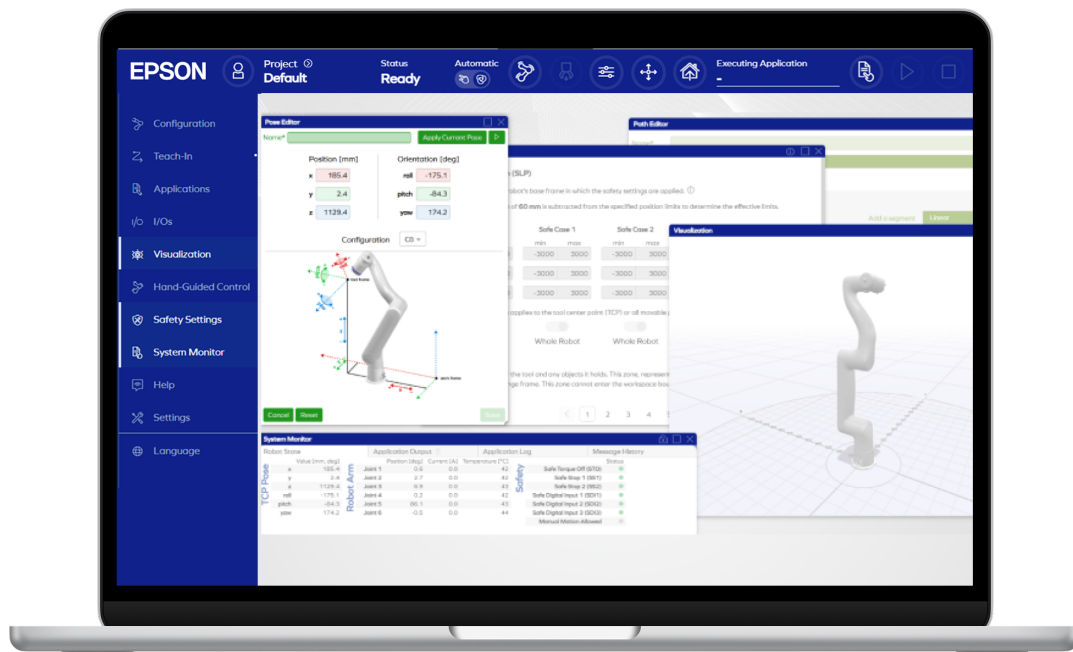


Figure 4.1.a: Example Screenshot of the Desktop Interface

The Desktop Interface of AX Portal is the main Graphical User Interface (GUI) that allows the user to control the robot, program applications, manipulate database elements, and visualize the current robot status. This GUI has been designed to be intuitive and mostly self-explanatory, even to users without much experience in robot control. You can access this interface via the [Admin Interface](#), by logging in and using the “Go to Desktop Interface” button.

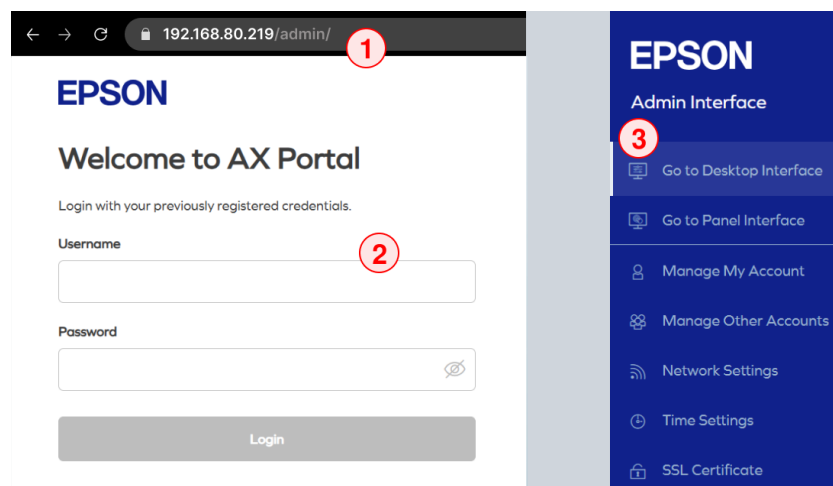


Figure 4.1.b: Accessing the Desktop Interface

1. Enter IP address in browser URL, 2. Enter credentials to log in, 3. Navigate to Desktop Interface

The Desktop Interface contains all available control functionalities for the robot and additional features to observe and modify its state. To simply execute preprogrammed applications, without the need to modify any applications or settings, we recommend to use the [Panel Interface](#) instead. In this chapter, all windows, menus, and elements of the Desktop Interface are explained in detail.

In some browsers (especially on Windows), the display scaling/zoom is set to a high number (e.g. 150%) by default, which may cause parts of the interface (such as cockpit or logo) to appear cut-off. For the best experience, we recommend setting the display scaling/zoom to 100% (or even lower).

4.2 Main Window

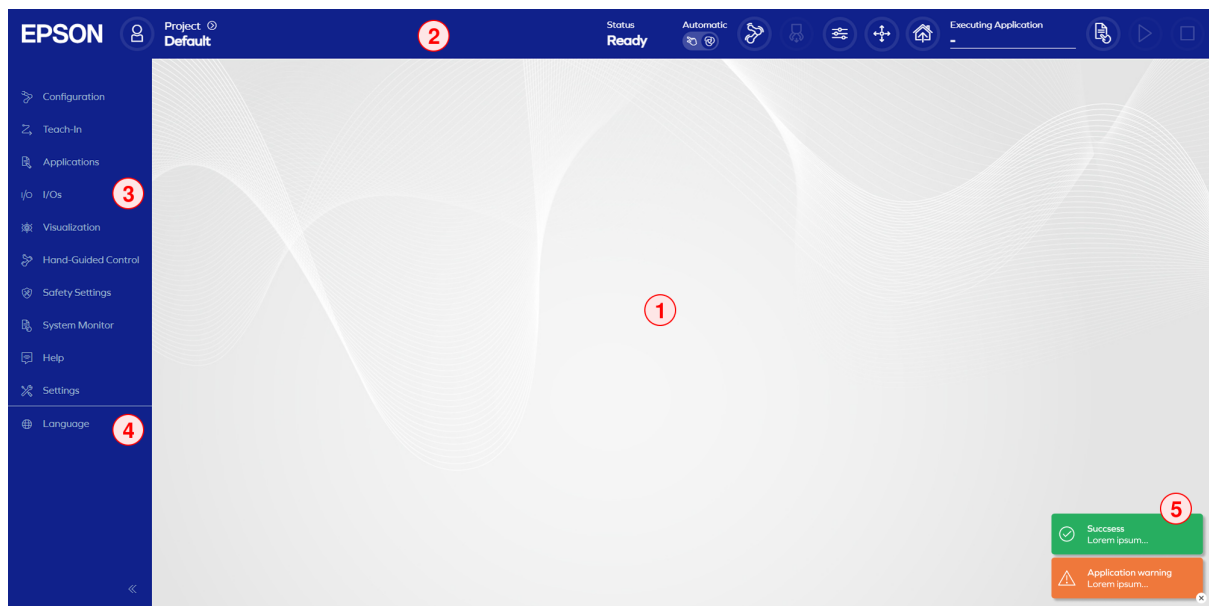


Figure 4.2: Elements of the Desktop Interface

1. Working Area, 2. Cockpit, 3. Menus, 4. Switch Language, 5. Pop-Up Messages

Once you successfully connect your computer or device to the Desktop Interface, the main window appears in your web browser.

This main window contains the following items:

- a cockpit on the very top of the window, showing the current program status, messages, and quick access to the robot controls,
- the menu buttons on the left side of the window, which you can click to navigate through the available menus (which will all be explained further down in this chapter),
- space for message pop-ups on the right side of the screen, and
- the main working area, in which the user can rearrange menus in a window explorer kind of style.

4.2.1 Menu Overview

On the left side of the main window, you will find buttons to enter the different menus of AX Portal. The exact purpose and detailed information of each available menu is described in the following sections. Here is a quick overview over the available menus:

	The Configuration menu lets the user configure and power-up the robot.
	The Teach-In menu lets the user teach poses, paths, and grids to the robot.
	The Application menu lets the user write, modify, and test custom applications.
	The I/Os menu lets the user observe I/Os and change output signals.
	The Visualization menu displays the current motion of the robot.
	The Hand-Guided Control menu lets the user move the robot joints by hand.
	The Safety Settings menu lets the safety officer modify the safety setup.
	The System Monitor menu lets the user observe system outputs.
	The Help menu contains helpful information about the program and system.
	The Settings menu lets the user choose some basic settings.
	The Language button lets the user switch the display language of the interfaces.

4.2.2 Cockpit

The cockpit of AX Portal is always visible on top of the GUI and contains the most important information and controls. Let's explain all of the cockpit elements in detail from left to right:

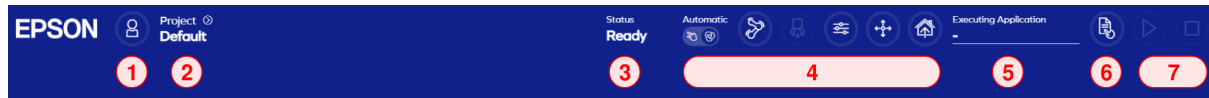


Figure 4.2.2: Cockpit

1. User Account Options, 2. Project Selection, 3. Robot Status, 4. Robot Controls, 5. Currently Running or Selected Applications, 6. Application Selection, 7. Main Controls



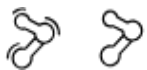
Press the **User** button to change your account settings or to log out. These options will navigate you to the [Admin Interface](#) one way or another.

Press the **Project** button to enter the [Projects Menu](#), where you can switch projects and see the project overview. Below this button you see the name of the active project.

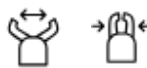
The **Request Control** button only shows if this interface is not the active control station of the robot. Read more about this safety feature in the [Active Control](#) section.

The **Status** message bar shows the current [Program Status](#). The program status basically defines whether a command or an action such as a button press is executable or not.

The **Operation Mode** slider switches the robot between manual and automatic operation modes. Read more about this safety feature in the [Operation Modes](#) section.



The **Enable/Disable HGC** button lets the user switch to the hand-guided control mode. While enabled, the status switches to “HGC” and the user may guide the robot manually. The robot can only switch to HGC mode if the program status is ready. Disabling HGC will switch the status back to “ready”.



The **Tool** button toggles between the “pick” and “place” functions of the mounted tool. In the [Tool Editor](#) the user may link these functions to digital inputs. The tool can only be triggered if the behavior is defined and if the program status is ready or HGC.



The **Jogging** button opens up the [Jogging Controls](#). Here the user may change individual robot coordinates or joint angles.



The **Motion** button opens up the [Motion Controls](#). Here the user may manipulate the robot velocity, change the global velocity and acceleration limits and change coordinate frames.



The **Homing** button lets the user move the robot to its homing position. This is meant as a shortcut to e.g. move to a starting position before running an application or move to a sleep position before powering down. The homing position can be changed in the [Settings Menu](#).

The **Currently Selected** area shows which element (application, path, pose, etc.) is currently linked to the control buttons (play, pause, resume, stop). This information changes whenever you click on a play button or select an application.



The **Application Selection** menu lets the user select an application that was previously created using the [Application Editor](#).



The **Play** button executes the selected application. The program status needs to be “ready” to play an application. The **Pause** button pauses a running application. If an application is paused during a movement of the robot, the robot will smoothly come to a stop. The **Resume** button resumes a paused application. If the application has been paused during a movement, this movement will smoothly continue. The **Stop/Abort** button stops a running or pausing application. If an application is stopped during a movement, the robot will immediately stop.



and hold its position. This button triggers an automatic recovery routine that switches the status back to “ready”.



4.2.3 Active Control

To make sure that the robot can only be controlled by a single control station at any time, the interfaces are protected with the active control feature. If you are the first user to log into the robot, you automatically get control. Let's call this the [Active Control Station](#). If someone else (or yourself) then tries to use a different connection channel (may also be a different browser window), this connection will not have control, and only has access to monitoring and observing the robot (read-only access). Let's call this the [Passive Observer](#).

4.2.3.1 Active Control Station

The active control station has the possibility to release its control. In the interface this is done by clicking the **User** button and selecting **Release Control Access**. If you tend to work on a robot with multiple users or interfaces, it is recommended to always release the control access and/or close the connection channel.

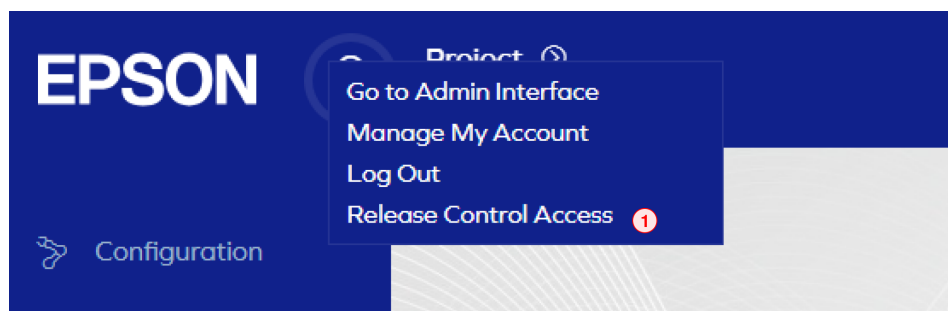


Figure 4.2.3.1 Releasing Control Access
1. Release Control Access

4.2.3.2 Passive Observer

The passive observer has the possibility to request control. In the interface this is done by clicking the **User** button and selecting **Request Control Access**, or simply by clicking on the big **Request Control** button that appears in the cockpit. After requesting control, a dialog pops up at both the active control station and the passive observer asking for access. While the passive observer can only wait until control access was granted, the active control station can either accept or reject the request.



Figure 4.2.3.2 Requesting Control Access
1. Request Control Access, 2. Request Control Dialog

4.2.4 Program Status

The current program and/or robot status is visible in the cockpit. The status gives feedback about what AX Portal and/or the robot is currently doing, and it basically dictates which actions or commands can currently be performed. Each status has its own properties and some functionalities are only available in specific program statuses. Here's a list of the different statuses and their explanations:

Unpowered	The motors are not powered and the devices are not connected. This state is used to configure the robot before powering up. Using the power-up slider makes the status change to Powering Up . Other actions may change the status to Loading , especially if an action should not be interrupted and takes some time to complete.
Loading	The robot is currently processing some data, e.g. loading a database or some settings. After properly loading the data, the status changes back to Unpowered .
Powering Up	The robot is currently powering up its motors and connecting its devices. After successful connection, the robot starts actively controlling its motors and the status changes to Ready .
Ready	The program is ready to receive commands and/or performing actions such as executing an application, picking and placing objects, etc. This is the idle state of the robot.
Running	An application/motion is currently running. This application/motion can be paused or aborted using the corresponding Pause or Stop buttons, respectively. After the application reaches the end of its code or the running motion finishes, the status goes back to Ready .
Paused	An application/motion is currently paused. Press the Resume button to continue, which sets the status back to Running , or press the Stop button to abort, which sets the status back to Ready .
Hand-Guided Control	The hand-guided control (HGC) is enabled for one or multiple joints of the robot. In this mode, you may guide the robot manually to any pose you want by applying force on its joints. Meanwhile, the robot holds its own weight by actively compensating the gravitational force. Note that you need to press the HGC button or activate the enabling device to move the robot. Disable the Hand-Guided Control again to bring the status back to Ready .

Processing	A time consuming action like saving/loading data or calculating a trajectory is currently on-going. After this action has been processed successfully, the status will change back to its previous value.
Stopping	Usually, the Stopping status automatically recovers to Ready in no time. If that should not be the case, some part of AX Portal could not be properly finalized. This unfortunately can only be recovered by restarting the robot.
Emergency Stop Normal Stop Protective Stop Collision Safeguard Position Limit	These states correspond to different interrupts. If an interrupt is triggered, the corresponding state is displayed and the Interrupt Window opens up for handling the interrupt. Once the interrupt has been resolved, the status usually changes to Proceed .
Proceed	After an interrupt was resolved, the status changes to Proceed. This requires the user to manually acknowledge that it's safe to proceed with the robot application or behavior.
Error	An error has occurred that could not automatically be resolved or that requires special attention from the user. Have a look at the error message and try to solve the error if possible.
Powering Down	The robot is currently powering down its motors, disconnecting its devices, and finalizing all running modules. The status thereafter changes to Unpowered .

4.2.5 Operation Modes

The AX series robot comes with two operation modes: Manual vs Automatic. The operation mode can be toggled using the dedicated slider in the cockpit.

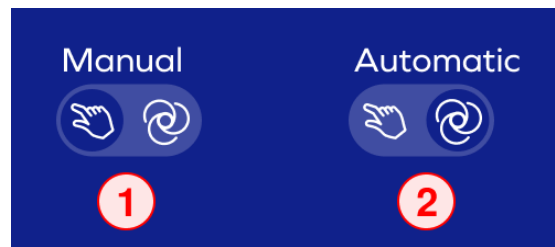


Figure 4.2.5 Operation Mode Toggle Button

 WARNING	Automatic mode must only be used if the system is equipped with the necessary safety functions, has been tested, and its safety has been confirmed by a risk assessment. Using automatic mode in other situations is very dangerous. For further information, please refer to the “AX6 / RC-A101 Safety Manual”.
---	--

4.2.5.1 Automatic

The automatic operation mode is usually used on normal operation, meaning when the applications have been verified to run properly. This mode allows the full speed range of the robot.

4.2.5.2 Manual

The manual operation mode is a safety feature and is usually used when modifying the robot's applications, especially when teaching new poses or testing changes on applications. In manual mode, the robot moves with reduced speed (max. 250 mm/s) and only moves when the **Enabling Device** is being triggered. An enabling device is a 3-position safety switch. If the enabling device is not triggered, the user is informed by a warning that states that movements are not allowed.



Figure 4.2.5.2 Enabling Device Example

4.2.6 Jogging Controls

Press the jogging button in the cockpit to open up the Jogging Controls. Using jogging, the robot can be fine-tuned up to 0.1 [mm]/[deg] precision. This window comes with two tabs. The first tab lets the user modify the current Euclidean coordinates of the tool center point (TCP), which is the same as the current tool frame expressed in the work frame. The second tab lets the user modify the current set of joint angles of the robot.

These tabs come with the following features:

Input Fields	Entering a number into the input field changes a single coordinate or joint value.
Slider	The slider defines both the Increment Step that defines how far the robot moves on short-clicking a control button and the Maximum Motion Velocity that defines how fast the robot moves using these controls.
Control Buttons	A short press on a control button applies the selected increment in the direction depicted by the button. A long press on a control button on the other hand applies a <u>continuous motion in the direction depicted by the button</u> .
Configuration	One specific TCP can be reached by up to eight sets of joint angles. These sets of joint angles are called the configurations of this pose. By changing the Configuration value in the drop-down list the user may select a specific configuration for the current pose. And by pressing the Next Configuration button the user may iterate through the available configurations of the current pose. Note that changing the configuration is not a fine-tuning step, this may move all joints of the robot.
Save as Pose	By pressing the Save as Pose button, this pose is saved under a new unique name. All available poses are listed in the Teach-In menu.

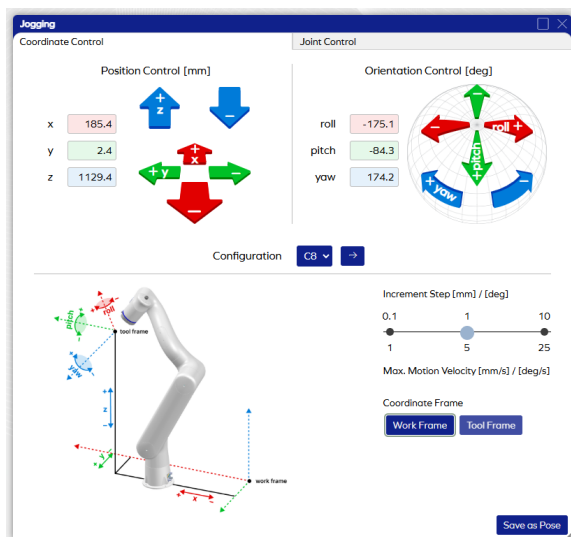


Figure 4.2.6.a: Jogging – Coordinate Control

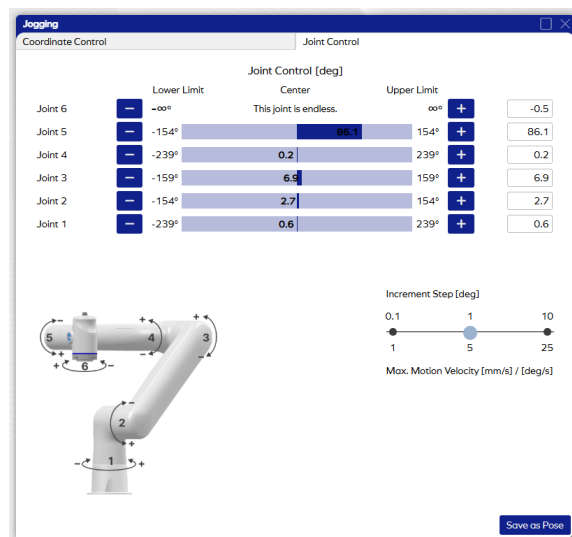


Figure 4.2.6.b: Jogging – Joint Control

4.2.7 Motion Controls

Press the motion controls button in the cockpit to open up the Motion Controls. This window comes with the following three tabs:

4.2.7.1 Motion Limits

The motion limits tab contains settings to limit the maximum robot velocity, and to scale down the velocity of all robot movements by a scaling factor. These controls are useful during setup of robot applications, e.g. for limiting the velocity while validating a new application or changing movements in existing applications.

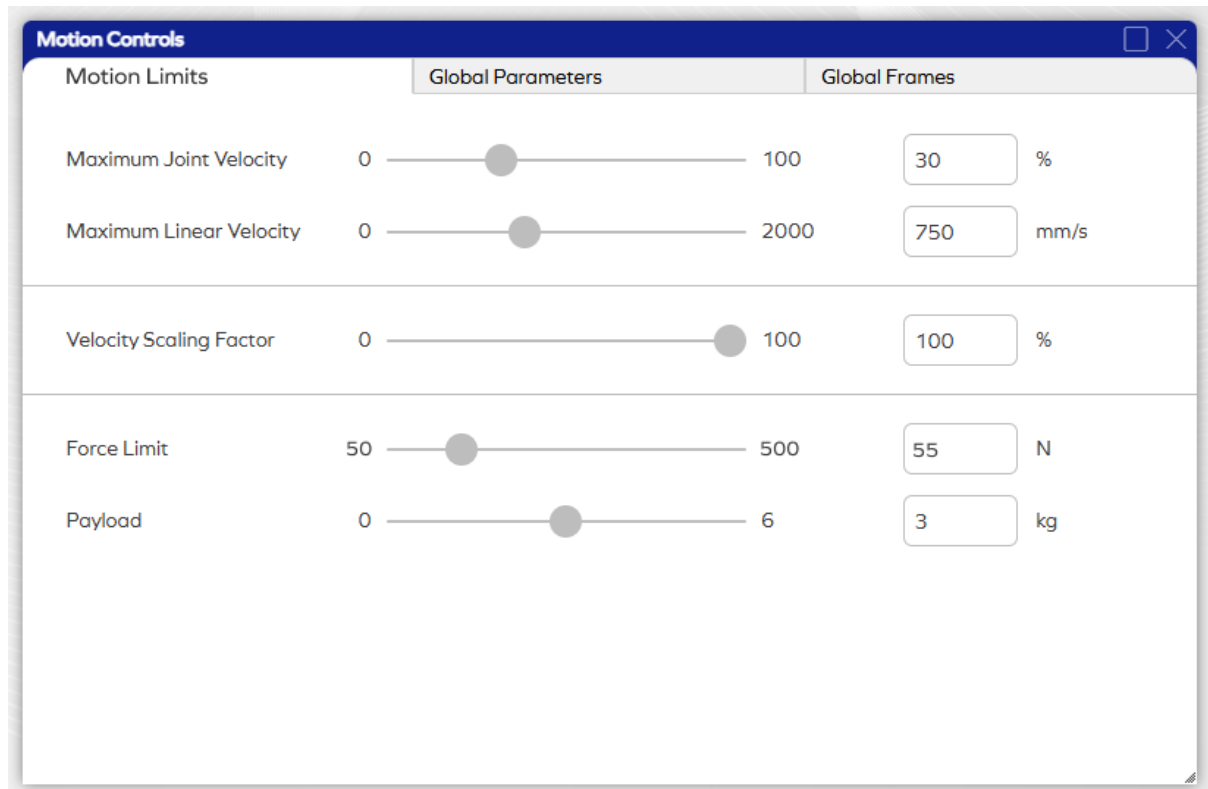


Figure 4.2.7.1: Motion Controls – Motion Limits Tab

The force limit and payload on the other hand are used to tune the collision sensitivity of the robot. The force limit defines an upper limit on allowed external forces on the tool, and the payload defines the total weight the robot carries (including its tool).

4.2.7.2 Global Parameters

The global parameters tab contains settings to define the default velocity and acceleration for various movement commands. Movement commands are divided into these categories:

- Commands for moving the robot in a time-optimal way use the “joint_velocity” and “joint_acceleration” arguments, whose default values are set here as **Global Joint Velocity** and **Global Joint Acceleration**.
- Commands for moving the robot in a specific direction or shape (linear, circular) use the “linear_velocity” and “linear_acceleration” arguments, whose default values are set here as **Global Linear Velocity** and **Global Linear Acceleration**.
- Commands for moving the robot such that the tool center point (TCP) stays at the same position, only changing its orientation, use the “angular_velocity” and “angular_acceleration” arguments, whose default values are set here as **Global Angular Velocity** and **Global Angular Acceleration**.
- Commands for moving along a path that contains multiple motion segments use the “blend_radius” argument, whose default value is set here as **Global Blend Radius**. This parameter is used to smoothen the interpolation between path segments.

Motion Controls		
Motion Limits	Global Parameters	Global Frames
Global Joint Velocity	0 — [Slider] — 100	20 %
Global Joint Acceleration	0 — [Slider] — 100	40 %
Global Linear Velocity	0 — [Slider] — 2000	250 mm/s
Global Linear Acceleration	0 — [Slider] — 8000	500 mm/s ²
Global Angular Velocity	0 — [Slider] — 1000	150 deg/s
Global Angular Acceleration	0 — [Slider] — 4000	1000 deg/s ²
Global Blend Radius	0 — [Slider] — 200	5 mm

Figure 4.2.7.2: Motion Controls – Global Parameters Tab

4.2.7.3 Global Frames

The global frames tab contains settings to define the default coordinate frames of the robot. While the Base Frame and the Flange Frame are fix attached to the robot, the Work Frame and Tool Frame can be defined relative to these fixed frames:

- The **Base Frame** is fixed to the center of the robot's bottom mounting surface, such that the z-axis points into the robot arm, and the x-axis points away from the connected cables.
- The **Flange Frame** is fixed to the center of the robot's top mounting plate. Its orientation is aligned with the Base Frame if all joints of the robot are at zero degrees.
- The **Work Frame** is the reference frame in which the user expresses the coordinates of the tool center point (TCP) throughout applications. By default, the Work Frame matches the Base Frame of the robot. Many movement commands use the "work_frame" argument, whose default value is set here as **Global Work Frame**.
- The **Tool Frame** defines the position and orientation of the tool center point (TCP) expressed in the Flange Frame. By default, the Tool Frame matches the Flange Frame of the robot. Many movement commands use the "tool_frame" argument, whose default value is set here as **Global Tool Frame**.

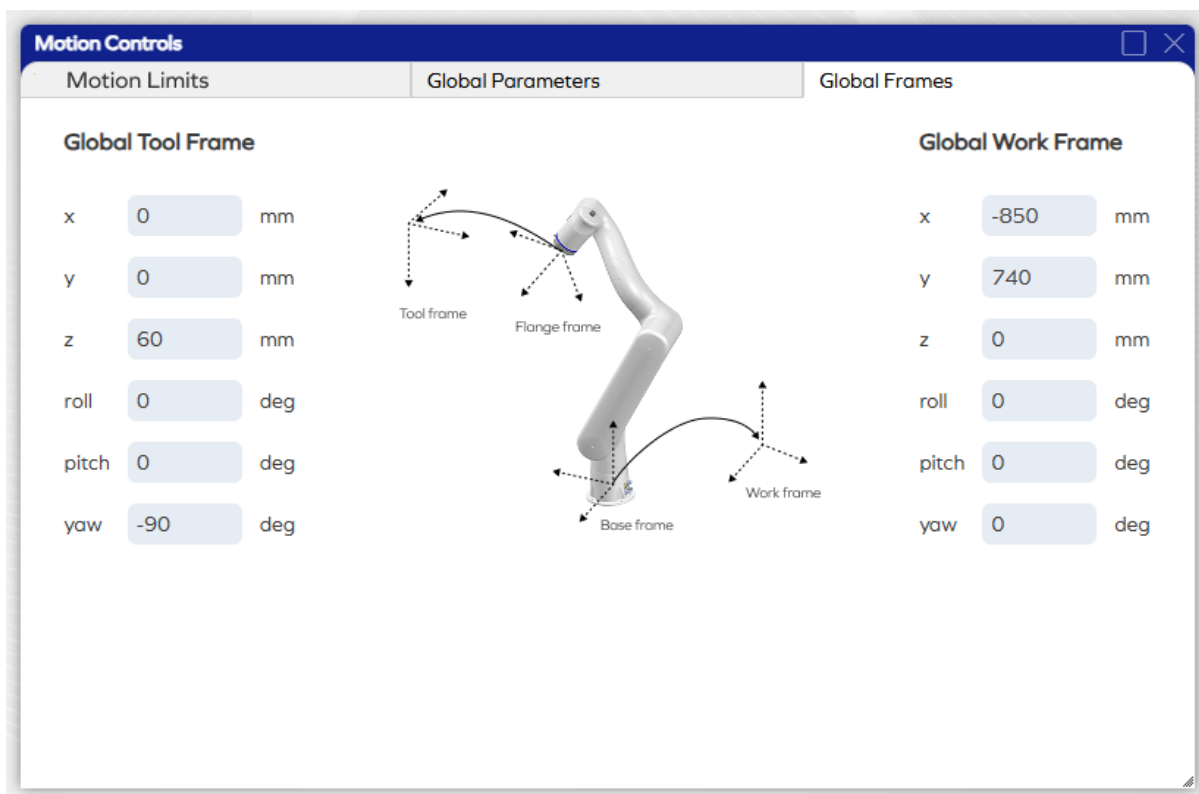


Figure 4.2.7.3: Motion Controls – Global Frames Tab

4.3 Building Blocks & Database

In order to achieve easy and intuitive application programming, AX Portal comes with an object oriented structure containing different building blocks. These building blocks (further called “elements”) can be created individually in different editors which are accessible via the Desktop Interface. AX Portal then allows the user to smartly interlink these elements to create applications in an object oriented way, resulting in application code that is short, easily readable, and usually does not contain many numbers and coordinates.

Here's a list of elements provides:

	Applications contain mainly the application code (and/or blocks) and application specific settings. The program code supports Python 3.13 and besides basic Python functions it also includes many functions from the internal robot control API (application programming interface). Applications are created and modified using the Application Editor .
	Poses are positions of the Tool Center Point (TCP) with specific orientations, uniquely defined by a set of joint angles. Poses are created and modified using the Pose Editor .
	Paths are sequences of poses describing smooth movements of the TCP and therefore movements of the whole robot. Paths are created and modified using the Path Editor .
	Grids are customizable, up to three-dimensional arrays of points that define e.g. locations of predefined trays. Grid points are extracted and used as iterators in applications. Grids are created and modified using the Grid Editor .
	I/Os are separated into digital and analog signals. They can be configured using the I/O Configuration Menu , and the value of the I/Os is visible in the I/O Menu . I/Os are used to connect and communicate with external devices.

All of these elements are stored in the database of AX Portal. You can download the database of your robot at any time (to have a back-up of your complete set-up), or upload a new database (overwriting the current database and all of its elements) via the [Configuration Menu](#). Also, the [Project Menu](#) provides a nice overview over all projects and elements stored in your database.

4.4 Project Menu

Click on the name of the currently selected **Project** in the cockpit to access the project menu. This menu contains an overview over the available elements, like poses, paths, grids, and applications. These elements are grouped into several projects. Only one project is selected at any time and only the elements of the selected project are visible, editable, and usable in the other menus of AX Portal.

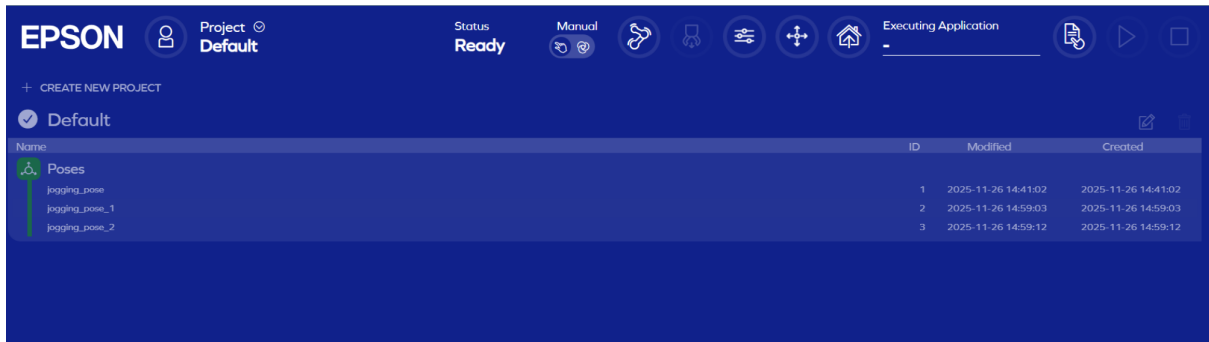


Figure 4.4: Project Menu

-
-  Press the **Create New Project** button to create a new, empty project.
 -  Press on the circle to the left of the project name to select that project.
 -  Press the **Copy** button of a project element to create an exact copy of that element within the same project. The name of the copied element is slightly modified and ends with “(copy)”.
 -  Press the **Edit** button to the right of the project name to change the name of that project. Press the **Edit** button of a project element to open up the appropriate editor for inspection or modification.
 -  Press the **Delete** button to the right of the project name to remove that project incl. all of its elements. Press the **Delete** button of a project element to remove that particular element from a project.
-
- Press on an element symbol (e.g. Poses, Paths, etc.) to show or hide the list of elements in that particular section.
-
- Move** an element from one project to another by dragging it from the source project with the mouse and dropping it in the target project. Press the **CTRL** key while dragging the element to create a **copy** of that element in the target project.
-

4.5 Configuration Menu

Press the **Configuration** button on the main window to access the configuration menu. In this menu, you may configure the database and the hardware setup (I/Os and tools). If you operate a static robot for a specific application, you will only need to set up the configuration once, and then you'll use this window only to power-up the robot. And you may even automate the steps of powering up and running an application using the appropriate settings in the [Settings Menu](#). If you use your robot for multiple purposes though, you may come back to this menu more often, and use different tools or even different databases for different projects and purposes.

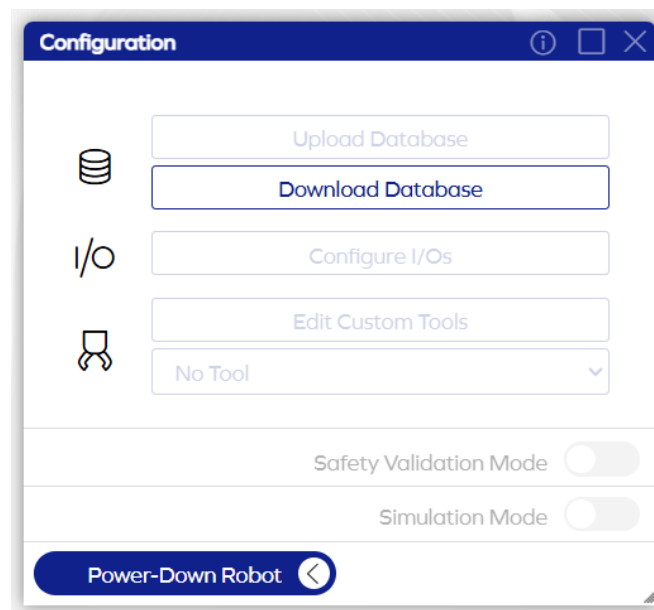


Figure 4.5: Configuration Menu

4.5.1 Database (Download & Upload)

In the database section, you may download the current version of your application database, or upload (and overwrite) the database with another compatible version of the database. **Downloading the database** will create a local copy of the database ZIP file on your device. It is recommended to **regularly backup your database**, especially if you have lots of applications and lots of time invested in writing and fine-tuning your applications. By **uploading a database**, your browser will ask you to select a ZIP file from your device. Selecting the database file will **override the current database** and replace it with the uploaded one.

By downloading a database on one robot and uploading it to another robot, you can copy all applications and elements across robots. It is recommended to regularly back-up the database.

Also note that databases are versioned, and not every database is compatible with every version of AX Portal. Older databases can be imported in newer versions of the software, but newer versions of the database cannot be loaded in older versions of the software. In other words, databases are not backwards compatible.

4.5.2 I/O Configuration

Press the **Configure I/Os** button to open the **I/O Configuration** window. Here you can configure all I/Os, assign predefined functions to digital inputs, and specify custom names for your I/Os. Custom names may be used in application scripts to increase the readability of your script. There you can simply replace the integer I/O identifier (int) by the I/O name (str).

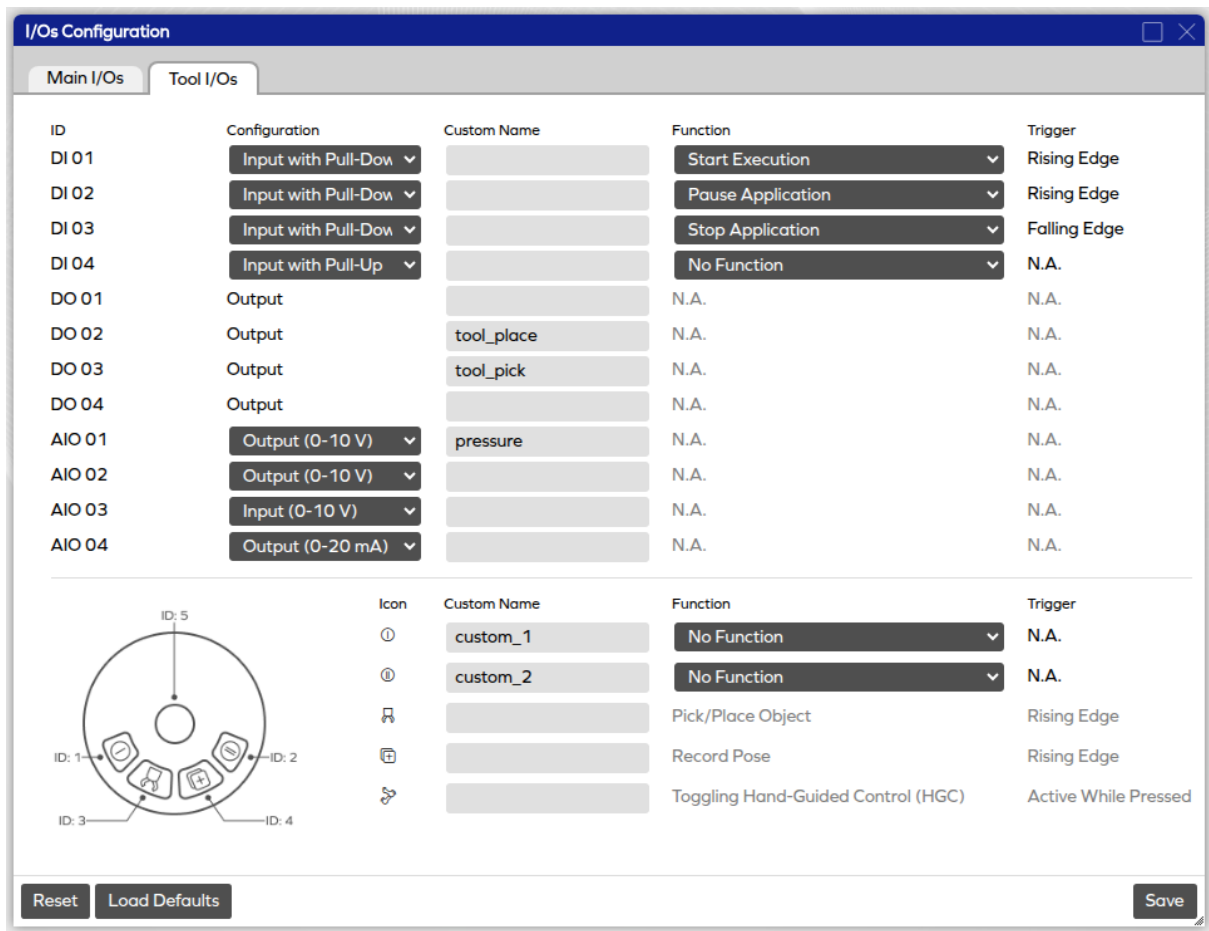


Figure 4.5.2.a: I/O Configuration Menu

The AX series comes with a **Button Interface** attached to the uppermost joint. Functions can be assigned to the two custom buttons exactly the same way as functions can be assigned to other digital inputs in this menu.

Note that triggering any function by digital input, the active control is withdrawn from the current control station.

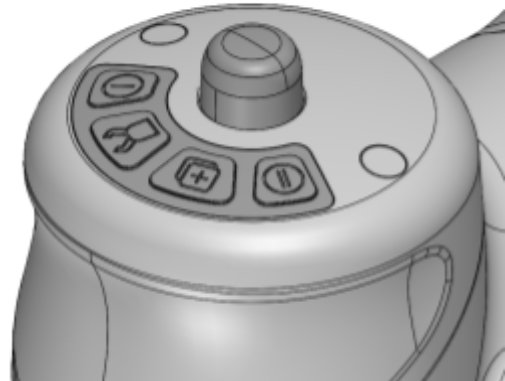


Figure 4.5.2.b: AX Button Interface

The following predefined functions for digital inputs are available, which you can set to trigger on rising edge (default) or falling edge:

No Function	Nothing triggered. This is the default.
Pause Application	Pausing the currently running application. This is similar to pressing the “pause” cockpit button.
Stop Application	Stopping the currently running or paused application. This is similar to pressing the “stop” cockpit button.
Start Execution	Running the currently selected application. This is similar to pressing the “run” cockpit button. If the robot recovers from an interrupt, this function moves the robot back to the pose where the robot was interrupted, and continues the previous motion. This is similar to pressing “proceed” and “resume” buttons in the dialog that pops up in the GUI after an interrupt occurs.
Reset Safety	Resetting the safety after an interrupt. This is similar to resetting the safety via the dialog that pops up in the GUI after an interrupt occurred.
Enable HGC	Enabling hand-guided control (HGC) for all joints (while status is ready).
Disable HGC	Disabling hand-guided control (HGC) for all joints (while status is hand-guided control).
Enable/Disable HGC	Toggling between enabling and disabling HGC for all joints. The triggered function depends on the current program status.
Pick Object	Triggering the tool pick function (predefined or linked to digital outputs). This is similar to pressing the “pick” cockpit button.
Place Object	Triggering the tool place function (predefined or linked to digital outputs). This is similar to pressing the “place” cockpit button.
Pick/Place Object	Toggling between picking and placing an object using the tool. Depending on the current tool status.
Record Pose	Record the current robot pose and save it as “button_pose” (or similar).
Manual Mode	Switching the robot to manual operation mode. This is similar to switching the mode using the switch in the cockpit.
Automatic Mode	Switching the robot to automatic operation mode. This is similar to switching the mode using the switch in the cockpit.

4.5.3 Tool Configuration

In the tool section, you can configure the robot's tool. If you are about to integrate a custom tool, e.g. a tool from a third party company or your own special tool for some specific application, you may use the **Edit Custom Tools** option. This will open up the list of available custom tools, with the following options:

	Press the Create New Tool button to open up the tool editor for defining the kinematic and dynamic properties as well as pick and place triggers of your tool.
	Press the Search button to filter the list of custom tools by name. Entering three or more characters enables the search filter.
	Press the Edit button (or double-click anywhere on the element) to open the tool editor in a separate window for inspection or modification.
	Press the Delete button to irreversibly remove a tool from your database.

4.5.4 Tool Editor

The tool editor is split into four sections (see figure below). To fill in these values, you may need to study the datasheet of the tool you are about to integrate into AX Portal, where physical and electrical properties of the tool may be specified in detail.

In the first section, you need to define the **Tool Center Point (TCP)**, which is the [pose](#) in space where the tool interacts. For a tool that picks objects, this means that the TCP defines the location where the tool is considered to be located while it is being picked. The TCP is defined in the [coordinate frame](#) that is attached to the very center of the top connector plate, which is oriented the same way as the base frame (considered all joints are at zero degrees). This frame is the **Default Tool Frame** in case no TCP is defined (x, y, z, roll, pitch, and yaw all at zero). From there you may translate and rotate the TCP such that it matches the point of interaction of your custom tool.

In the next step, it is necessary to adapt the **Dynamic Properties** of your custom tool. The quality of features like gravity compensation and collision detection depend on the dynamic model of the robot to be as precise as possible, so it is worthwhile to measure and define these properties of your tool. You need to define the total mass of the tool, as well as the center of mass, which (same as the TCP) is defined in the **Default Tool Frame** of the robot. The [rotational moment of inertia](#) may either be manually entered (if known), or automatically approximated (AX Portal does not know the shape and mass distribution of your tool).

Tool Editor : pneumatic_gripper

Name*

pneumatic_gripper

1. Defining the Global Tool Frame

Translation [mm]

x

0

y

0

z

36

Rotation [deg]

roll

0

pitch

0

yaw

90

2. Defining the Dynamic Properties

Mass [kg]

0.662

Center of Mass [mm]

x

2

y

-1.5

z

14

Rotational Moment of Inertia [kg m²]

Automatic

☒

Manual

0.0004649

0

0

0

0.0004649

0

0

0

0.0005880

3. Defining the Picking Behaviour

Mode

Dual Output Mode

Select Digital Output for Picking

tool DO 03 (tool_pick)

Select Digital Output for Placing

tool DO 02 (tool_place)

Output State for Picking & Placing

High

Delay Between Output Triggers T_1 [ms]

20

Pick

Place

Pick Output

Place Output

T_1

T_1

Cancel

Reset

Save & Close

Figure 4.5.4: Tool Editor

Finally, if your tool comes with a mechanism to pick up an object, here you can define which digital outputs are triggered by the **Tool Pick** and **Tool Place** functions which are accessible both via the cockpit and script functions. The following **Picking Behaviors** are available:

4.5.4.1 Single Output Mode

This is the simplest of the available picking behaviors. You simply define one digital output which is used for toggling between picking and placing, and you can define whether picking is triggered on a HIGH or LOW signal.



Figure 4.5.4.1: Example of Single Output Mode with HIGH for picking (and LOW for placing)

4.5.4.2 Dual Output Mode

This picking behavior allows you to define separate digital outputs for picking and placing. Toggling the tool state will then trigger both outputs consecutively with a (minimum) time delay T_1 (in milliseconds). In addition, you can define whether the active state refers to a HIGH or LOW signal.

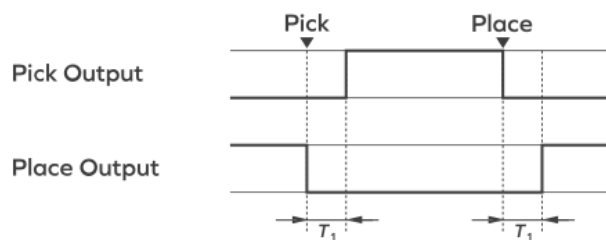


Figure 4.5.4.2: Example of Dual Output Mode with HIGH as active trigger state

4.5.4.3 Separate Trigger Mode

In the separate trigger mode, you can define two digital outputs, one that defines the direction (whether to pick or place) and one that defines the trigger (whether to trigger the selected direction). For the direction output you can choose whether a HIGH or LOW signal is used for picking (the other one is placing). For the trigger output you can also choose whether a HIGH or LOW signal is used for triggering. In addition, you may define the (minimum) time delay T_1 (in milliseconds) between choosing the direction and triggering the selected direction, and the (minimum) time duration T_2 (in milliseconds) that defines how long the trigger will be active.

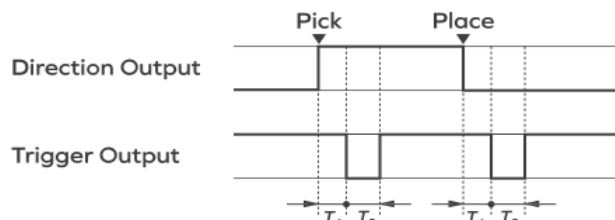


Figure 4.5.4.3: Example of Separate Trigger Mode with HIGH as picking direction and LOW as trigger

4.5.5 Robot Power-Up

After loading the database and configuring the I/Os and the tool, it is time to power-up the robot. Usually you are not changing the initial setup of the robot, and you will directly go ahead to drag the **Power-Up Robot** slider fully to the right side, which triggers the initialization routine.

If you do not want to power-up the hardware, but instead only modify database content (applications, poses, etc.) without using any hardware (e.g. not moving the robots, not reading or writing I/Os), you may enable the **Simulation Mode** toggle first, and then drag the **Power-Up Robot** slider to the right side. Note that in simulation mode all movements are simulated in real time.

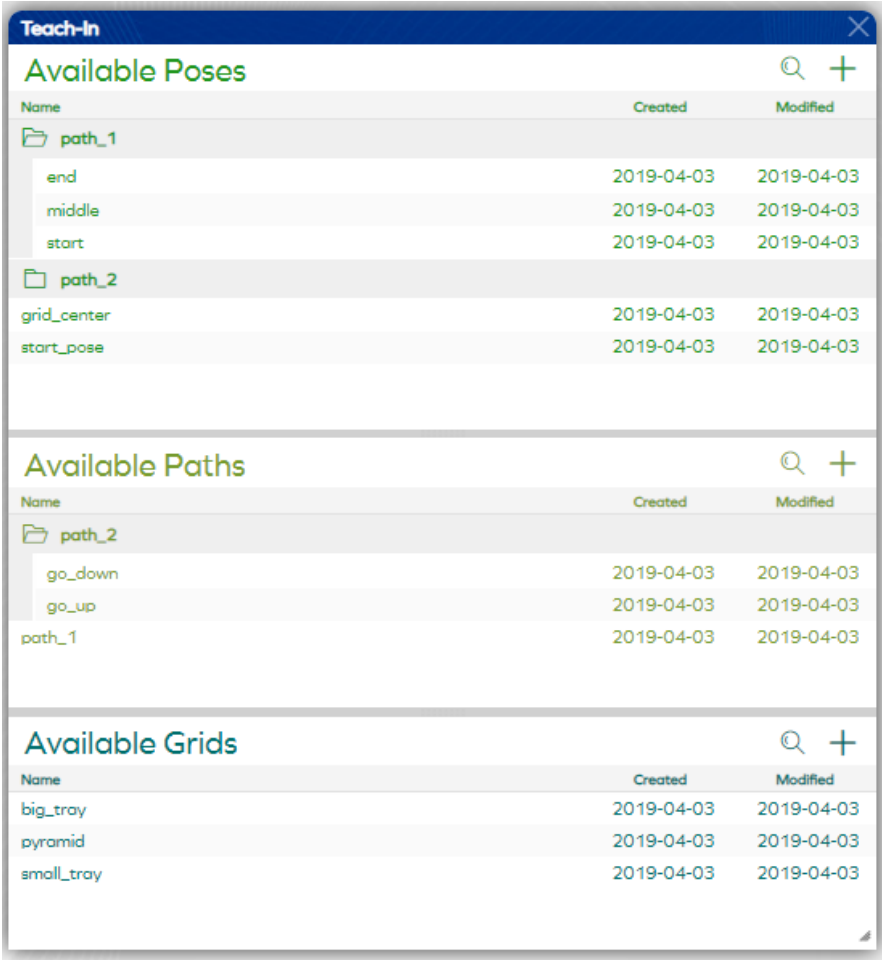
While the robot is powered up, you will not be able to change any of the initial configurations of the robot. You can only download the database or choose to **Power-Down Robot**, which is recommended if the robot is idle or on standby.

If you are logged in as a safety officer, you will have access to the **Safety Validation Mode** toggle here. Enabling this mode tells the software to ignore position, velocity, and force limits, which can be used to verify that the safety device actually triggers in case of a limit violation. Note that using one of the [Motion Controls](#) in this mode still overrides the limits.

4.6 Teach-In Menu

Press the **Teach-In** button on the main window to access the teach-in menu. This menu contains an overview over all available poses, paths, and grids of the currently selected project.

Poses and paths can be created, edited, copied, deleted, and even directly executed. Grids on the other hand are helpful tools to keep repeating and equally spaced coordinates organized. Please note that grids are not directly used for moving a robot, but they bundle, and store sets of coordinate values in order to keep your application code structured and to prevent you from entering coordinates. A list of grid points can be extracted from a grid for diverse purposes, usually though grids are set up such that their points directly represent TCP (tool center point) coordinates, which then are used as input values for movement functions.



The screenshot shows the 'Teach-In' window with three sections: 'Available Poses', 'Available Paths', and 'Available Grids'. Each section has a search icon and a plus icon for adding new items. The 'Available Poses' section lists poses under two folders: 'path_1' (end, middle, start) and 'path_2' (grid_center, start_pose). The 'Available Paths' section lists paths under 'path_2' (go_down, go_up) and 'path_1'. The 'Available Grids' section lists grids: big_tray, pyramid, and small_tray. All items have creation and modification dates of 2019-04-03.

Available Poses		
Name	Created	Modified
path_1		
end	2019-04-03	2019-04-03
middle	2019-04-03	2019-04-03
start	2019-04-03	2019-04-03
path_2		
grid_center	2019-04-03	2019-04-03
start_pose	2019-04-03	2019-04-03

Available Paths		
Name	Created	Modified
path_2		
go_down	2019-04-03	2019-04-03
go_up	2019-04-03	2019-04-03
path_1	2019-04-03	2019-04-03

Available Grids		
Name	Created	Modified
big_tray	2019-04-03	2019-04-03
pyramid	2019-04-03	2019-04-03
small_tray	2019-04-03	2019-04-03

Figure 4.6: Teach-In Menu

	Press the Plus/Add button to open a pose, path, or grid editor in a new window. Further down in this section, all of these editors are explained in detail.
	Press the Search button to filter the list of poses, paths, or grids by name. Entering three or more characters enables the search filter.
	Press the Play button to move the robot to a pose or to play a path. Note that playing a path will first move the robot to the starting pose of that path, if it is not yet at this particular pose.
	Press the Copy button to create an exact copy of this pose or path. The name of the copied element is slightly modified and ends with "(copy)".
	Press the Edit button (or double-click anywhere on the element) to open this pose, path, or grid in a separate window for inspection or modification.
	Press the Delete button to irreversibly remove a pose, path, or grid from your database.
	Press on a Folder to show or hide the poses, paths, or grids within that folder. Elements only show in folders (and subfolders) if their names contain one or more "/" characters, e.g. "folder/subfolder/name".

4.6.1 Pose Editor

The pose editor allows users to create a new [Pose](#) or modify an existing one. Note that changing the coordinates or configuration of a pose does not automatically move the robot. If this is desired, the [Logging Controls](#) should be used instead.

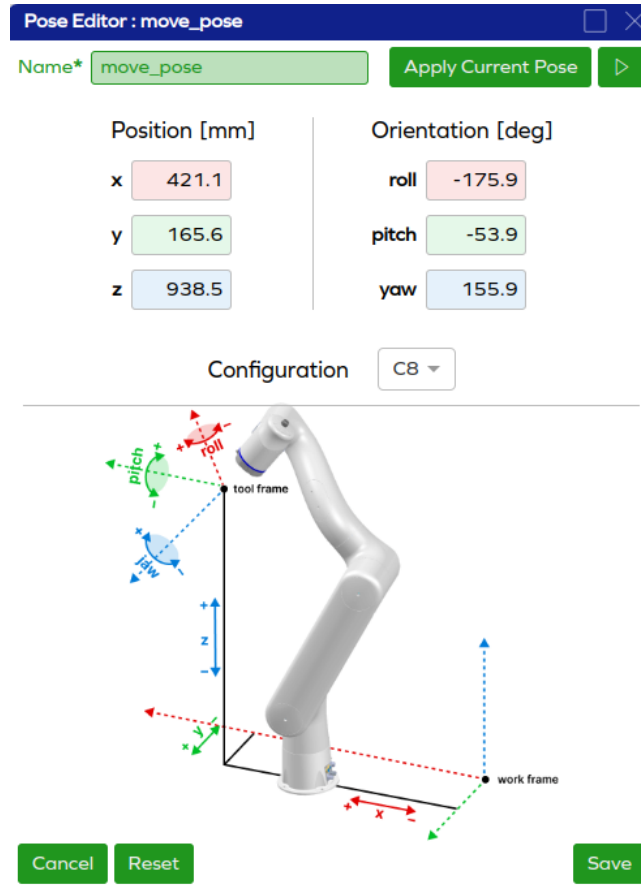


Figure 4.6.1: Pose Editor

Name:

Specifies the name under which the pose is stored in the database. Use “/” characters in the name to group your poses in a folder-like structure.

Position:

Represents the Tool Center Point (TCP) position in millimeters, defined relative to the global work frame. Note that invalid position values, e.g. ones that cannot be reached by the robot arm, can be entered. However, they will be highlighted in red as pressing the play button will result in an error. This is permitted because the pose can be valid when used with different coordinate frames and offsets.

Orientation:

Represents the TCP orientation in deg, defined relative to the global work frame. The orientation is expressed using [nautical angles](#). As with position values, invalid orientation values are highlighted in red.

Configuration:

Specifies the robot arm [configuration](#). The user can select from all available configurations (C1–C8) via the dropdown menu. Note that modifying the configuration does not alter the TCP position or orientation, it only affects the joint angles used to achieve the specified pose.

Additional Actions in the Pose Editor

Save	Clicking the Save button stores the current pose under the specified name. A pose cannot be saved without a valid name.
Cancel	Clicking the Cancel button discards all unsaved changes made in the editor and closes the window.
Reset	Clicking the Reset button reverts all unsaved changes and restores the pose to its last saved state.
 Play	Clicking the Play button at the top of the window commands the robot to move to the defined pose.
Apply Current Pose	Clicking the Apply Current Pose button updates all fields to reflect the robot's current pose values.

4.6.2 Path Editor

The path editor allows users to create a new [path](#) or modify an existing one. It is divided into three editable sections: **Name**, **Initial Pose** and **Segments of this path**.

The screenshot shows the 'Path Editor : path_1' window. At the top, there's a 'Name*' field with 'path_1' and a play button. Below it is the 'Initial Pose' dropdown set to 'path_1/initial_pose'. The main section is 'Segments of this path'. It contains two segment cards. The first is a 'Linear' segment with 'Target Pose' 'path_1/pose_1', 'Velocity' 500 mm/s, and 'Acceleration' 1000 mm/s². Below it is a 'Blend Radius' field set to 100 mm. The second is a 'Circular' segment with 'Intermediate Pose' 'path_1/pose_2', 'Target Pose' 'path_1/pose_3', 'Velocity' 250 mm/s, and 'Acceleration' 500 mm/s². At the bottom right of the segments list is an 'Add a segment' button with a 'Circular' dropdown and a plus icon. At the very bottom are 'Cancel', 'Reset', and 'Save' buttons.

Figure 4.6.2: Path Editor

Name:

Specifies the name under which the path is stored in the database. Use “/” characters in the name to group your paths in a folder-like structure.

Initial Pose:

Defines the starting pose of the path. The robot first moves to this pose using the default velocity and acceleration settings before executing the defined segments sequentially.

Segments of this Path:

Lists and defines the individual segments that make up the path. The following segment types are available:

- Linear Segment
- Circular Segment
- Coordinates Segment
- Pose Segment
- Orientation Segment

A more detailed description of each segment can be found in the [segments](#) section.

To **add a new segment**, select the desired segment type from the dropdown menu in the lower-right corner and click the plus (+) button. Once segments are added, their **order** can be freely adjusted using the drag-and-drop functionality.

Each segment requires the specification of a **target pose**, which defines the final pose for that segment. Target poses must be pre-defined in the [Pose Editor](#) and can then be selected via the dropdown menu. The Circular Segment uniquely requires an additional **intermediate pose**, which must also be selected from the list of pre-defined poses.

For each segment, **velocity** and **acceleration** parameters can be set. If left unspecified, the global default values will be used during execution.

The blend radius can be configured to control the smoothness and efficiency of transitions between segments. It defines the maximum allowable deviation in mm from the end pose of the previous segment corresponding to the start pose of the current segment.

Additional Actions in the Path Editor

The Path Editor also provides additional features to manage, edit, and organize the path effectively.

Save	Clicking the Save button stores the current path under the specified name. A path cannot be saved without a valid name.
Cancel	Clicking the Cancel button discards all unsaved changes made in the editor and closes the window.
Reset	Clicking the Reset button reverts all unsaved changes and restores the path to its last saved state.
 Play	Clicking the Play button at the top of the window commands the robot to execute the defined path. If the robot's current pose does not match the path's specified initial pose, it will first move to that initial pose before starting execution.
 Delete	Clicking the Delete button next to a segment removes that segment from the path.

4.6.3 Grid Editor

When creating a new grid, the grid editor opens. This editor guides the user step-by-step through all the available grid options. Use the buttons **Next** and **Back** to navigate through this wizard, until the desired ordered list of grid points is defined as desired. By editing already existing grids, the grid editor jumps directly to the manual point editing.

In a first step, the user must choose the **Grid Dimension**. Usually, a grid is interpreted as a two-dimensional array, though it may as well be a simple list of points without depending on a second dimension (e.g. for picking objects from a conveyor belt), or even a three-dimensional structure (e.g. stacking objects in boxes). The grid is initialized with perfectly aligned points, though note that single grid points may always be modified, added, or removed later on.

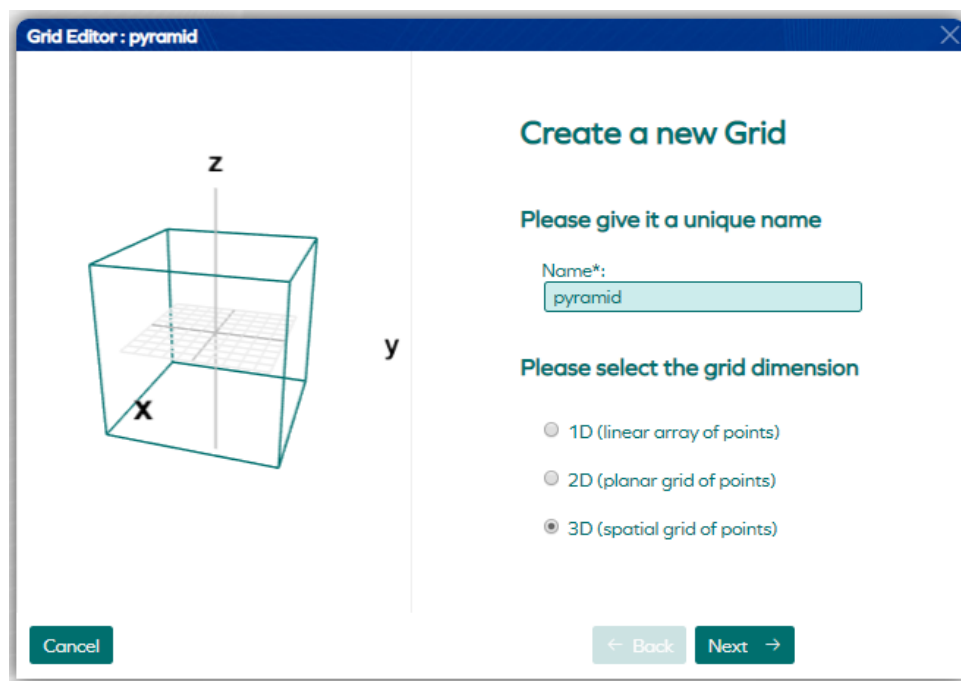


Figure 4.6.3.a: Grid Editor (1st step)

In a second step, the user must choose the **Corner Points** of the selected grid. Grids of different dimensions have a different number of corners, and the corners are labelled from “A” to e.g. “D” for a two-dimensional grid. The user may enter the corner coordinate values (given in the base frame of the robot) directly in the input fields, or may simply guide the TCP of the end-effector to the corners of the grid and press the appropriate “teach” button to save the coordinates.

Once enough corners are taught to define the grid (e.g. at least 3 points for the two-dimensional grid), the user may press the **Evaluate Values** button to evaluate the given corners and auto-fill the missing values to fit them to a parallelogram (two-dimensional) or a parallelepiped (three-dimensional). With the additional **Auto-Alignment** flag activated, the evaluation procedure takes into account horizontal and vertical coordinate axes and planes and tries to align the resulting grid along these axes and planes (if close enough).

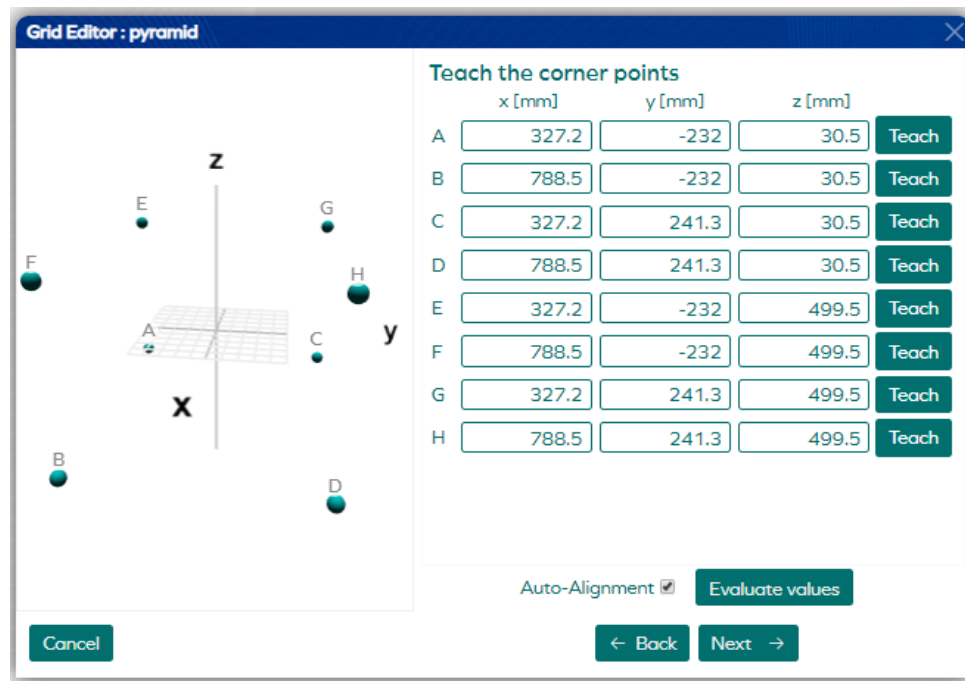


Figure 4.6.3.b: Grid Editor (2nd step)

In a third step, the user is asked to choose the number of points, special extension options and iteration directions. Let's have a look at these available options in detail:

- The **Number of Points** can be set for each dimension individually. If you find it hard to distinguish between the dimensions (e.g. rows and columns), refer to the labels of the previously taught corner points.
- For two-dimensional grids, there exists an **Extension Option** for adding intermediate points between each set of four neighboring points. For three-dimensional grids this option exists for each grid plane as well as for the set of eight neighboring points in space.
- The **Iteration Direction**, meaning the direction in which the points are returned and used as iterators in programming loops, is given by a starting point and a vector in each direction. The last direction is always given by the previously selected options and is simply shown for reference and visual verification. The letters in the drop-down menus correspond with the labels of the corner points of that grid.

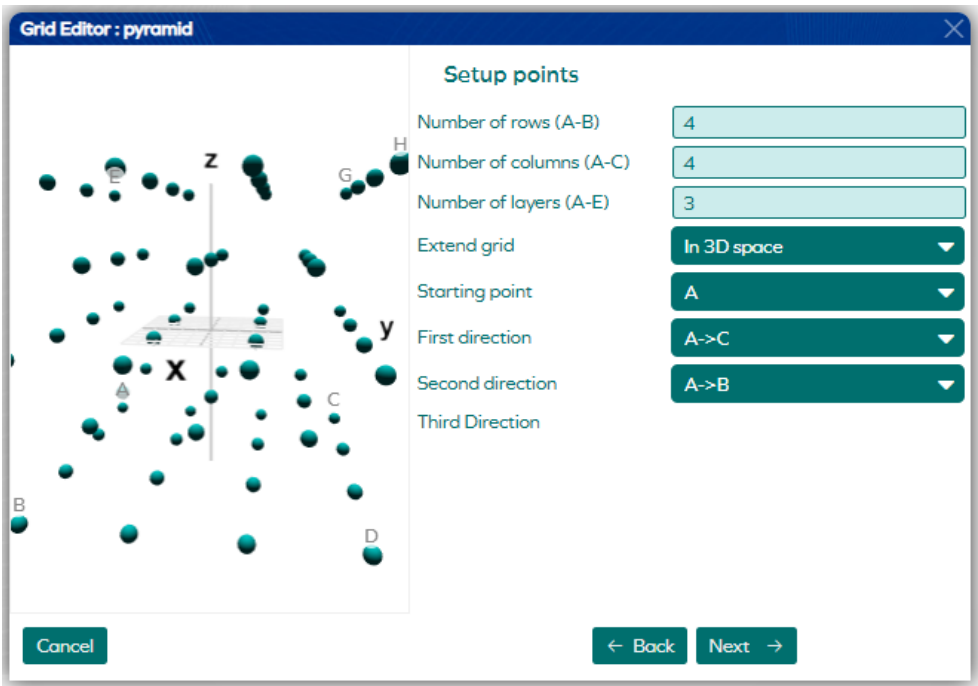


Figure 4.6.3.c: Grid Editor (3rd step)

Finally, the **List of Grid Points** is ready for manual editing. Hereby, single points can be copied, deleted, moved, and coordinate values may be manually changed. Meanwhile, the visualization represents the point order and the relative coordinates of the grid points.

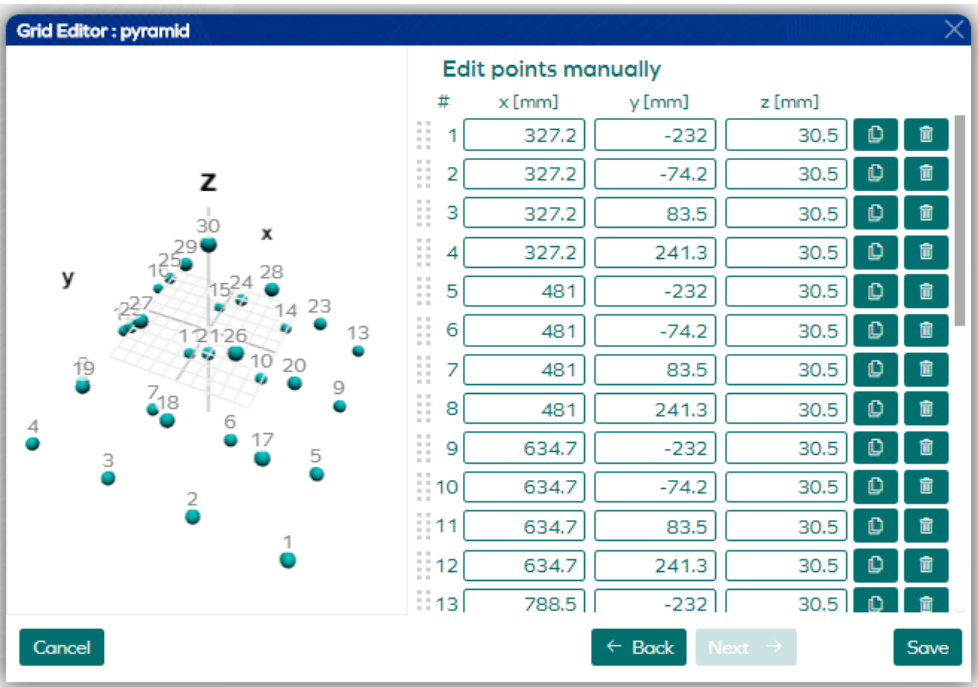


Figure 4.6.3.d: Grid Editor (4th step)

While editing, you may always go back to inspect previously set options and values. You may also change those options, however this will reset all changes in the following windows.

4.7 Application Menu

Press the **Application** button on the main window to access the applications menu. This menu contains an overview of all saved applications as well as the output of the last executed application. It also allows the user to access the **Application Editor** window to create new or edit existing applications.

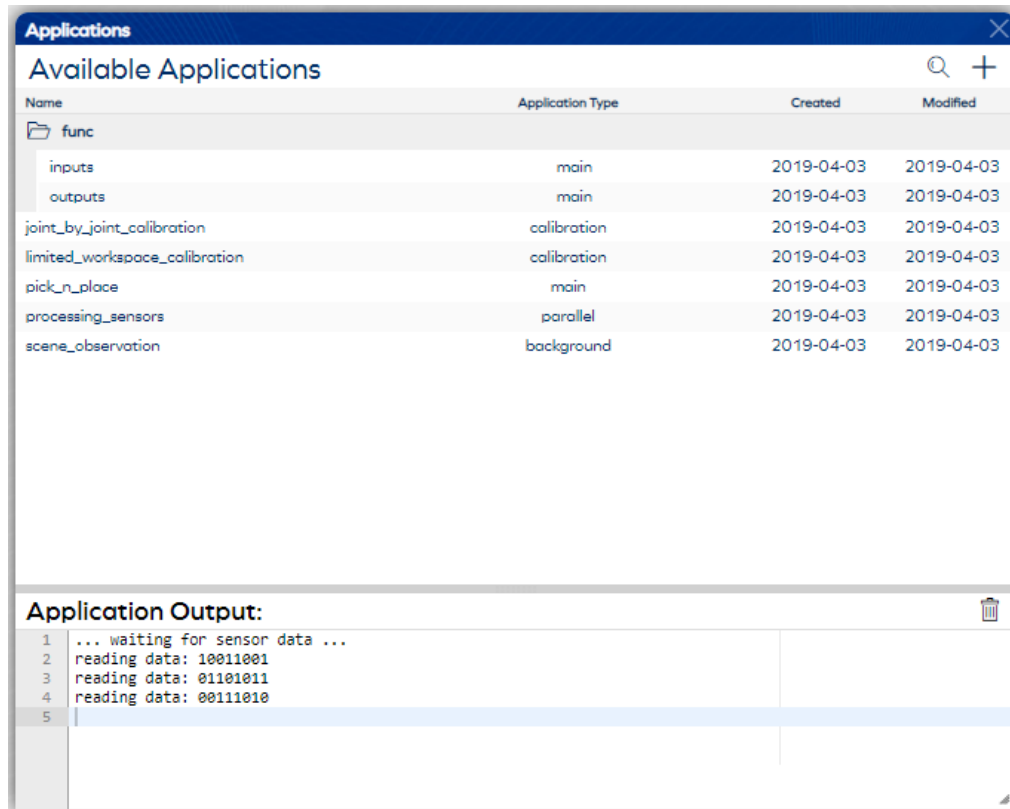


Figure 4.7: Applications Menu



Press the **Plus/Add** button to create a new application. This opens up the application editor, whose features are explained in detail further down in this section.



Press the **Search** button to filter the list of applications by name. Entering three or more characters enables the search filter.



Press the **Play** button to run an application. Note that only “main” applications are executable this way, as only “main” applications are selectable in the cockpit.



Press the **Copy** button to create an exact copy of this application. The name of the copied element is slightly modified and ends with “(copy)”.



Press the **Edit** button (or double-click anywhere on the element) to open this application in a separate window for inspection or modification.



Press the **Delete** button to irreversibly remove an application from your database. This removes the program code as well as the application specific settings.



Press on a **Folder** to show or hide the applications within that folder. Elements only show in folders (and subfolders) if their names contain one or more “/” characters, e.g. “folder/subfolder/name”. You may click the appearing **Edit** button to enter the [Folder Editor](#).

4.7.1 Application Editor

The **Available Applications** section of the **Applications** menu lists all previously saved applications. You may modify an existing application by clicking the **Edit** button or create a new application by pressing the **Create New Application** button. In both cases the **Application Editor** window will pop-up. Its functionality is similar for both cases. Let's quickly have a look at the main buttons of the application editor, before going into detail:

Save	By pressing the Save button, the application is saved using the given name. An application cannot be saved without a name. Use "/" characters in the name to group your applications in a folder-like structure.
Cancel	By pressing the Cancel button, all unsaved changes made in this editor are rejected and the editor closes.
 Test	By pressing the Test Application button on top of the window, the application runs in test mode. Test mode means that the script code executes but does not prevent you from continuing editing that application. Changes in the application that have been added after testing an application are not executed.

4.7.2 Application Code

The application **Code** tab is the main one of the three sub-windows of the application editor. This section shows the application code and optionally the library with AX Portal functions and elements to insert in the application code. The integrated programming language is **Python 3.13**, with most of its basic functionality, including some pre-installed Python packages. In addition to that, AX Portal comes with a big library of functions designed to simplify the writing of a robotic application (e.g. moving the robot, reading sensor data, communicating with the user, etc.). For details about these functions, please refer to the [Code Documentation](#).

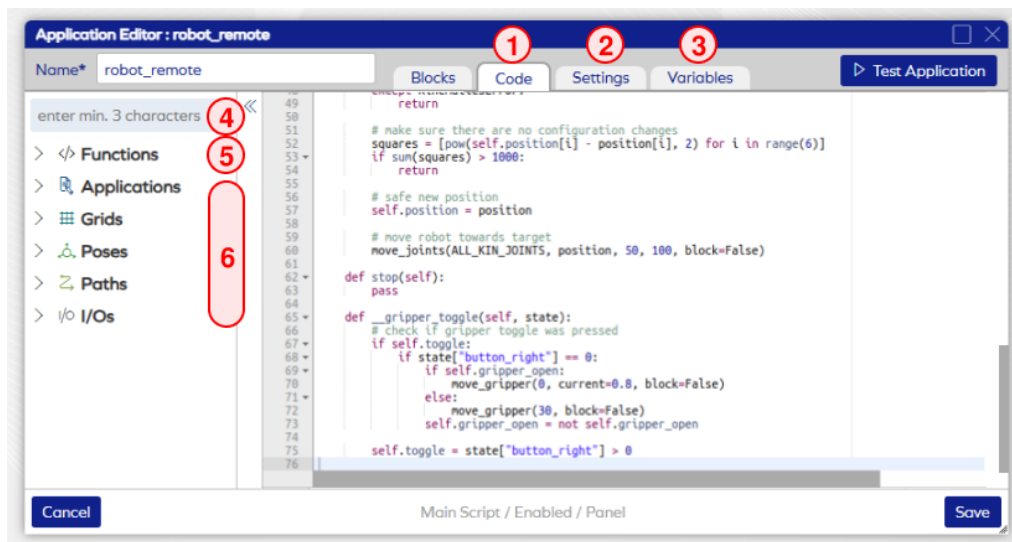


Figure 4.7.2: Application Editor (Code Section)

1. Write Your Python Program Code Here
2. Modify Application Specific Settings
3. Define and Modify Application Variables
4. Search Filter
5. Function Library
6. Available Elements to Use

Press on one of the **Function Symbols** or one of the **Element Symbols** on the left side of the editor to open up the element **Palette**. Here you will find a full list of available functions. Also, you have access to all elements that you can integrate in your code by double-clicking on them, so you don't have to remember the names of your elements.

Hint: Increase the readability of Python code by removing the optional arguments that are inserted into your code by double-clicking on a function. You may e.g. write

```
move_joints(joints, joint_position)
```

instead of

```
move_joints(joints, joint_position, joint_velocity=None,  
joint_acceleration=None, relative=False, block=True)
```

4.7.3 Application Settings

The application **Settings** tab contains some additional settings for the application, including

- an **Executable** flag that can be disabled e.g. if your application is still under construction and should not be executed,
- a flag to **Show** this application in the [Panel Interface](#),
- functionality to upload and replace the **Picture** shown in the Panel Interface,
- application **Labels** as a way to translate the application name into the currently selected interface language for the Panel Interface,
- and most importantly the **Application Type**, which is explained in the following section:

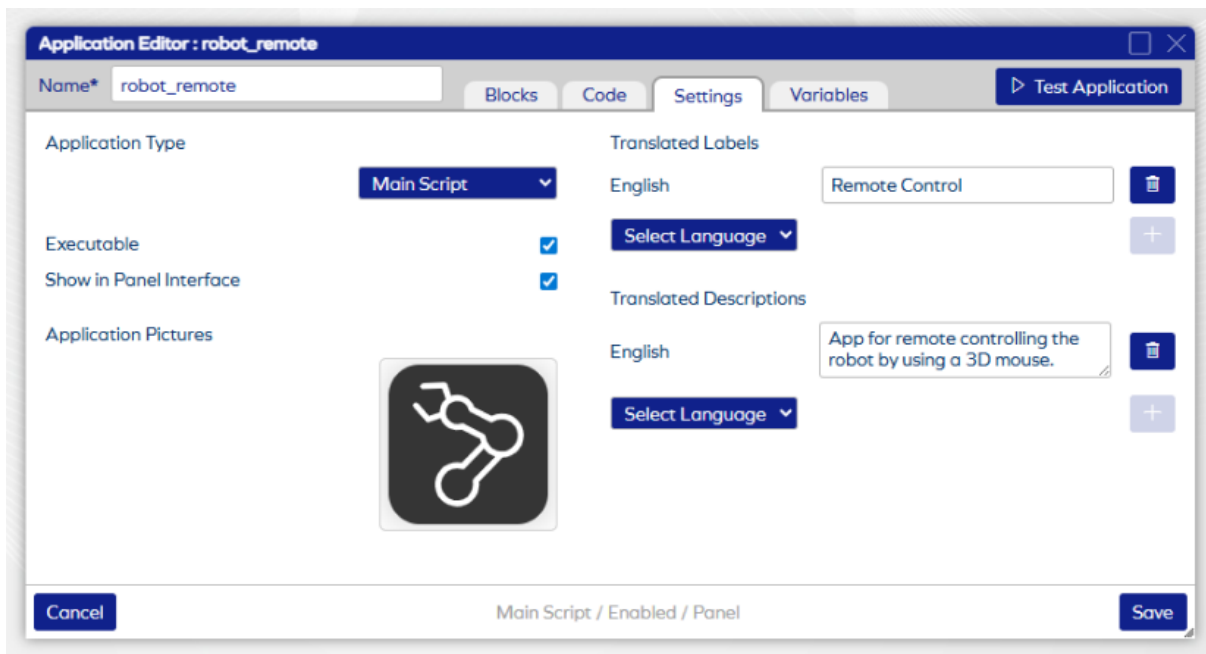


Figure 4.7.3.a: Application Editor (Settings Section)

The **Application Type** specifies in which context the application is supposed to be run, and therefore dictates how an application can be executed as well as which functions can be used within that application:

- **Main Scripts** contain the main applications to be executed. They are designed to use all the available script functions and to directly control the robot as well as its external devices. Usually, a simple application can be written in a single script, but for long or complicated applications, or to improve readability, it can be useful to split up an application into multiple subscripts that run sequentially to one another. Multiple main scripts never run in parallel to one another though, for this purpose “Parallel Scripts” are needed. You may also use the “import” statement to import functions across Main Scripts. Note that only Main Scripts are selectable from the cockpit.
- **Parallel Scripts** on the other hand run in separate threads and therefore are designed to run in parallel to a main application. This can be very useful if it comes to e.g. continuously reading sensor data, using the idle time of the robot moving to compute power consuming calculations. Parallel scripts are not allowed to use all script functions, like e.g. controlling the robot, since this could seriously mess up the program flow or invoke unwanted movement behavior if not paying close attention. All parallel scripts pause while the main application is paused, and they terminate once the main application stops running.
- **Background Scripts** are scripts that run in the background of AX Portal, independently of its status, and are therefore useful for e.g. constantly observing, checking, or logging data, or to define initial global variables for a project. Each project can contain only one background script. If this one is defined, it will automatically execute once the appropriate project is loaded (e.g. when starting up or when changing a project). The background script does not listen to the status restrictions and – if containing a loop – may run until the robot powers down. Whenever a background script starts and finishes, an appropriate notification is displayed. You may start, restart, or stop the background script from the cockpit.



Figure 4.7.3.b: Background Script Controls

4.7.4 Application Variables

The application **Variables** tab contains a drag and drop interface to add different types of state variables to this application. State variables can be used only in the main script. They are used in the [Panel Interface](#), where they are very useful to e.g. monitor the state and progress of an application or to specify certain input values to an application. Use the functions **set_state_variable(...)** and **get_state_variable(...)** to write and read state variables, respectively.

Here's an overview of the variable types and how they are displayed in the Panel Interface:

boolean	Booleans are represented as simple on-off sliders that are either enabled or disabled. They are read and written as Python type bool , where enabled matches True and disabled matches False.
integer	Integers are represented as text fields that allow numerical inputs, including +/- buttons to increase and decrease their values in edit mode. They are read and written as Python type int .
float	Floats are represented as text fields that allow numerical inputs in edit mode. They are read and written as Python type float .
slider	Percent variables are represented as sliders in edit mode, with a mandatory minimum and maximum value. They are read and written as Python type float .
text	Text variables are represented as text fields in edit mode. They are read and written as Python type str .
dropdown	Dropdown variables are represented as dropdown lists in edit mode. They are read and written as Python type str .

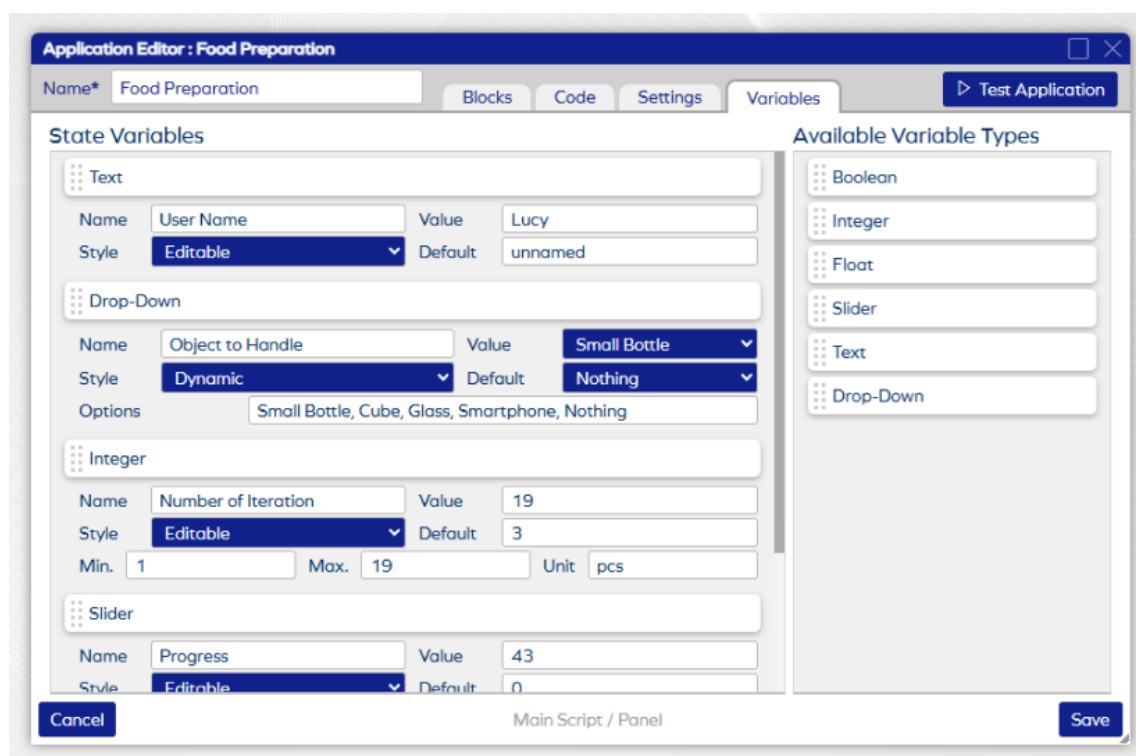


Figure 4.7.4: Application Editor (Variables Section)

Here's a quick overview over the variable options, which vary depending on variable type:

Name	Give your variable a suitable name, it will later be accessible by that name in string form.
Value	The value of a variable contains the currently set value for that variable, that will be read and written in the Python equivalent type explained above.
Default	The default of a variable is the value that it takes when resetting the variables using the according button in the Panel Interface.
Style	The style of a variable defines if, when, and how this variable is editable.
Min	Numerical value types allow for setting a minimum boundary on that number. If you enter values lower than the minimum, the minimum value will be applied. Boundaries are mandatory for percent variables and optional for integers/floats.
Max	Numerical value types allow for setting a maximum boundary on that number. If you enter values higher than the maximum, the maximum value will be applied. Boundaries are mandatory for percent variables and optional for integers/floats.
Unit	For numerical value types, you may add an optional unit string to the number.
Options	The options for dropdown variables can be entered in a comma separated string.

Here's a quick overview over the variable styles, which define how a variable is used:

- **Editable** variables allow for changing the value before running the application, but not during run-time.
- **Protected** variables are editable, and as a protection layer you have to press on an edit button to modify the value and on a save button to accept the value change.
- **Dynamic** variables are editable, and you may even dynamically change their values on run-time.
- **Dynamic & Protected** variables share the features of protected and dynamic variables and are therefore editable on run-time by pressing on an edit button.
- **Read Only** variables are not editable, their value can only be changed by the application. They are usually used to monitor the state or progress of an application.
- **Hidden** variables are not shown in the Panel Interface, but can be written and read the same way as other state variables.

4.7.5 Shared & Global Variables

In addition to the application specific state variables explained above, shared and global variables can be defined to share data between multiple scripts running in parallel or multiple sessions of the same application:

- **Shared Variables** exist only during run-time of a single application. Use the functions `set_shared_variable(...)` and `get_shared_variable(...)` to share values amongst multiple scripts running in parallel or sequence in that application.
- **Global Variables** share values not only between the scripts of an application, but between all applications and still exist even after rebooting the robot. Use the functions `set_global_variable(...)` and `get_global_variable(...)` therefore.

4.7.6 Folder Editor

If your application names contain the character “/”, the application is automatically sorted into a folder structure. You may edit a folder by clicking on the edit button appearing when hovering over that folder. This opens up the folder editor. This editor simply contains labels as a way to translate the folder name (similar to the application editor). The folder labels as well as the application labels appear in the [Panel Interface](#), this way the Panel Interface can easily be set up to be usable in multiple languages.

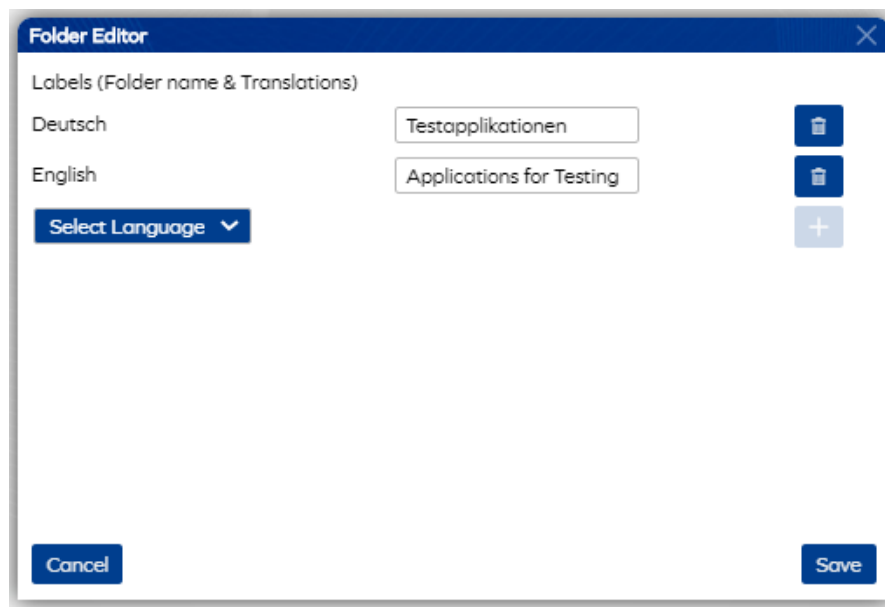


Figure 4.7.6: Folder Editor

4.7.7 Block Programming

The **Blocks** tab of the application editor contains a graphical programming interface. This interface allows the user to create simple straight-forward program flows by dragging and dropping programming blocks into the program flow area.

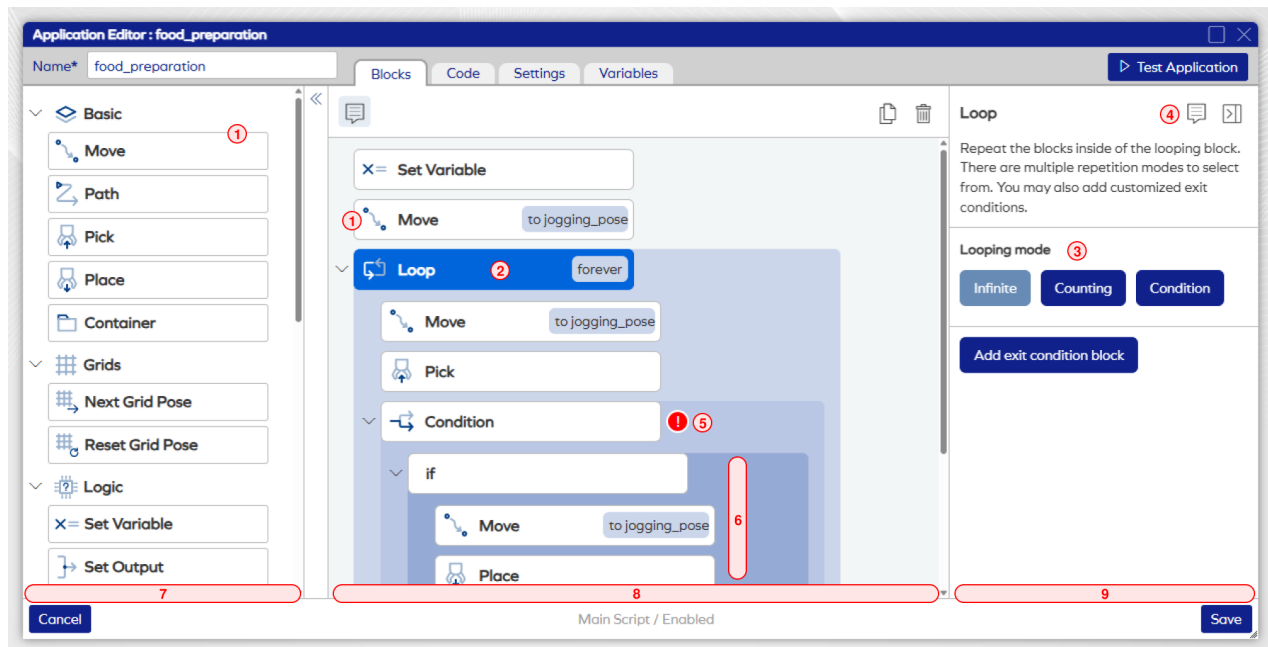


Figure 4.7.7: Application Editor (Block Programming)

1. Drag Blocks Into Program Area, 2. Select Block to Modify, 3. Modify Block Details, 4. Comment, 5. Error Indicator, 6. Block Containers, 7. Available Blocks, 8. Program Flow, 9. Block Details

Select a block in the programming flow to see its details. Some block details are mandatory to be filled in to make the program executable (e.g. for a movement block, you need to provide information about where the robot should move to). Other block details are optional (e.g. using default values if not provided) or can be used to improve the readability of the program (e.g. comments). If a block is not properly defined (e.g. mandatory parameter not defined, or invalid value provided) it will show an error indicator in the program flow, and the error reason in the block details when selected.

Some blocks create colored containers in the program flow when added. This is a graphical help to indicate that these blocks are grouped together and always executed in order (top down). Loops and conditions are examples of such blocks, which are responsible for e.g. executing a group of blocks multiple times, or executing a group only if a specific condition applies.

Furthermore, a selected block can be deleted, or copied (and dragged to the desired location in the program).

4.7.7.1 Using Application Variables and Expressions

Before explaining all programming blocks in detail, note that it is possible to use mathematical and logical expressions in most of the text fields inside the block details. These fields also support inserting values of predefined [application variables](#) as well as values of [digital and analog inputs](#).

This is easiest explained in an example. Let's assume we defined the following variables:

The 'State Variables' window displays two variable configurations:

- Boolean Variable:**
 - Name: flag
 - Style: Editable (dropdown)
 - Value: 0
 - Default: 0
- Integer Variable:**
 - Name: counter
 - Style: Editable (dropdown)
 - Value: 0
 - Default: 0
 - Min.: 0
 - Max.: 1000
 - Unit: (empty)

Figure 4.7.7.1.a: Example Variables

Examples for valid **numerical expressions**:

Examples of numerical expressions:

- value (number): 42
- value (number): -7 * (8.2 - 3)
- value (number): {counter} + 1
- value (number): (Tool AIO 02)

Examples for valid **logical expressions**:

Examples of logical expressions:

- value (bool): FALSE
- value (bool): NOT {flag}
- value (bool): {flag} AND ({counter} > 5)
- value (bool): {Main DIO 09}

Figure 4.7.7.1.b: Example Numerical & Logical Expressions

Values of variables and inputs can also be injected into regular text fields:

Example of text expression: The counter is set to {counter}.

Figure 4.7.7.1.c: Example Text Expression

4.7.7.2 Move Block

The **Move** block runs a movement function that moves the robot from its current position to a target. The user must select the movement style ...

- **Point to Point** → move to the target in a time optimal way, no restriction on the trajectory
- **Linear** → move to the target along a linear trajectory, the orientation is interpolated linearly
- **Circular** → move to the target along a circular trajectory, the orientation is interpolated linearly, and the user must specify an **Intermediate Position** on the circular trajectory

... and the target format.

- **Pose name** → choose the name of the (previously saved) target pose
- **Grid name** → move to the points of a (previously saved) grid, initially it will move to the first point of this grid, and then use the **Next Grid Pose** block to increase the grid point counter
- **Joint angles** → specify the joint angles of the target pose
- **Coordinates** → specify the coordinates of the target pose

Additionally the user may modify the following optional settings. Note that if not specified, default/global values will be used.

- **Speed Settings**
 - **Joint Velocity/Acceleration** → specify the speed for point-to-point movements
 - **Linear Velocity/Acceleration** → specify the speed for linear/circular movements
 - **Angular Velocity/Acceleration** → specify the speed for movements which only change the orientation of the TCP (the position stays the same)
- **Coordinate Frame Settings**
 - **Tool Offset** → shift the TCP with respect to the flange frame (top of the robot arm)
 - **Tool Frame** → alter the TCP with respect to the flange frame (top of the robot arm)
 - **Work Offset** → shift the reference frame of coordinates with respect to the base frame (bottom of the robot arm)
 - **Work Frame** → alter the reference frame of coordinates with respect to the base frame (bottom of the robot arm)
- **Other Settings**
 - **Relative Coordinates** → if enabled, coordinate values are added to the current coordinates
 - **Blocking Motion** → if enabled, the program flow only continues after the movement has completed

4.7.7.3 Path Block

The **Path** block triggers a previously defined path. The user selects the path to run. Running this block then first moves the robot to the initial pose of the path, and then runs along the path segments in one smooth motion.

4.7.7.4 Pick Block

The **Pick** block sends a pick command to the connected tool. For angular or parallel grippers this means closing the gripper to hold the object inside. For pneumatic grippers this means enabling the vacuum pump to hold the object in place. This block does not come with parameters.

4.7.7.5 Place Block

The **Place** block sends a place command to the connected tool. For angular or parallel grippers this means opening the gripper to let go of the object inside. For pneumatic grippers this means disabling the vacuum pump to drop the object in place. This block does not come with parameters.

4.7.7.6 Container Block

The **Container** block is simply a way of grouping together multiple blocks. It has no special functionality assigned to it.

4.7.7.7 Next Grid Pose Block

The **Next Grid Pose** block increases the internal pose counter for a specific grid. The affected grid needs to be selected. Whenever using a **Move** block to move towards a grid, it targets the pose currently pointed to by this counter. For actually moving to all points of a grid, you may use both the **Move** and the **Next Grid Pose** block inside a **Loop** structure as follows:

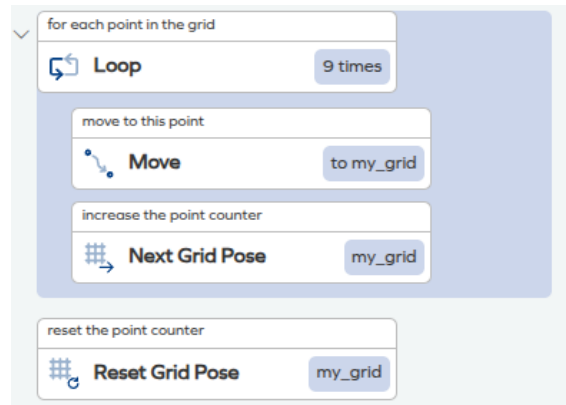


Figure 4.7.7.7: Using Grid Blocks Example

4.7.7.8 Reset Grid Pose Block

The **Reset Grid Pose** block resets the internal pose counter for a specific grid back to zero (initial value). The affected grid needs to be selected.

4.7.7.9 Set Variable Block

The **Set Variable** block sets the value of a predefined [application variable](#). It's mandatory to choose a variable first, and choose a value thereafter. The use of numerical and/or logical expressions is allowed.

4.7.7.10 Set Output Block

The **Set Output** block sets the value of a digital or analog output. The user first needs to choose the output to write, and depending on whether it's a digital or analog output, the value to set is a boolean or number, respectively.

4.7.7.11 Wait Block

The **Wait** block blocks the program flow until a certain event occurs. There are three different waiting modes to choose from:

- **Time** → wait for a specific amount of time to pass (numeric expression allowed)
- **Motion** → wait for a running movement command to stop
- **Input** → wait until the value of a digital input is set to a specific value, or the value of an analog input exceeds a certain threshold

4.7.7.12 Loop Block

The **Loop** block creates a group of blocks that is being called repeatedly. There are three different looping modes to choose from:

- **Infinite** → loop forever
- **Counting** → loop for a specific number of times (numeric expression allowed)
- **Condition** → loop until a specific condition is met (logic expression allowed)

The user may add an **Exit If** block to the loop, see next section.

4.7.7.13 Exit If Block

The **Exit If** block is bound to its **Loop** block and breaks out of this loop if the specified condition applies (logic expression allowed).

4.7.7.14 Condition Block

The **Condition** block creates a container with two groups “if” and “else” inside. The condition (logic expression allowed) is used to fork the program flow into either the “if” group (if the condition applies), or into the “else” group (if the condition does not apply).

4.7.7.15 Comment Block

The **Comment** block simply adds an informative comment into the program flow. Note that each other block comes with the possibility of adding a comment as well, which is then visible in the program area. Sometimes it makes sense though to create a separate comment that is not tied to some specific block.

4.7.7.16 Quit Block

The **Quit** block aborts the currently running program. Remaining programming blocks will not be executed.

4.7.7.17 Dialog Block

The **Dialog** block creates a container with multiple groups inside. On execution, the program flow interrupts and a dialog box pops up. The user must answer the dialog for the program flow to continue. The groups inside (by default “OK” and “Cancel”) match the buttons in the dialog. Depending on the pressed button, the program flow forks and continues with the chosen group. The dialog message (variable injection allowed), button names, and number of buttons can be modified in the block details.

4.7.7.18 Print Block

The **Print** block prints a custom message into the application output.

4.7.7.19 Log Block

The **Log** block logs a custom message into a custom log file. Log files can be downloaded via the Desktop Interface, e.g. using the shortcut CTRL + SHIFT + L.

4.7.7.20 Notify Block

The **Notify** block sends a custom notification message to the GUI. The notification does not block the program flow.

4.7.7.21 Warning Block

The **Warning** block sends a custom warning message to the GUI. The warning does not block the program flow.

4.7.7.22 Error Block

The **Error** block sends a custom error message to the GUI and aborts the currently running program.

4.7.7.23 Python Code Block

The **Python Code** block is an advanced block that lets the user call a few lines of standalone Python code. You may want to use this block to call functions that are otherwise not available via block programming. The complete set of application functions can be used within.

4.8 I/Os Menu

Press the **I/Os** button on the main window to access the I/Os menu. This menu shows all digital and analog I/Os, and their properties that were previously defined in the [I/O Configuration](#) Window (their configuration, assigned functions and custom names), as well as the values of all I/Os. The inputs are read-only, hence these values can simply be visually observed here. The outputs however can also be set via this window.

Digital I/Os represent boolean values, where **True or 1 or “green”** means **“powered”** and **False or 0 or “grey”** means **“not powered”**. Digital inputs show this value in a circular indicator, while digital outputs can be changed by pressing the square indicator button to change their states.

Analog I/Os are represented as floating point **values** in steps of 0.1. The unit of an analog I/O is defined by the configuration of this I/O (V for voltage, mA for current). While analog inputs are represented as an observable number, analog outputs can be changed using the text fields besides the current value of this output.

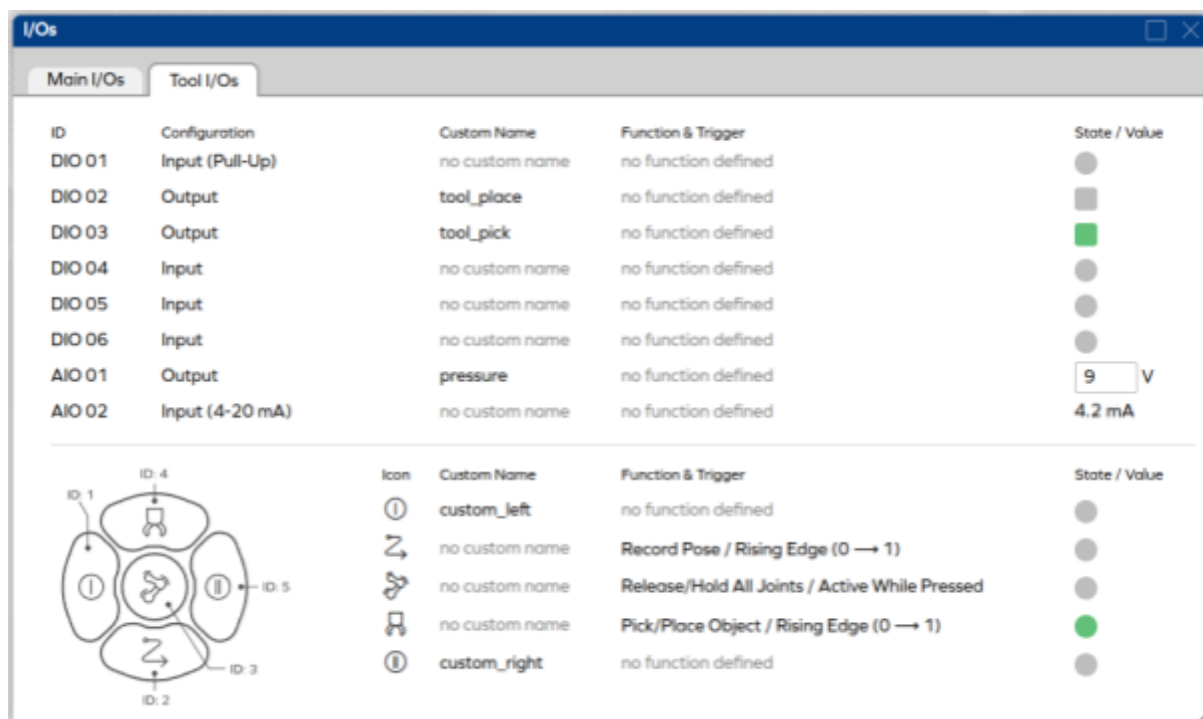


Figure 4.8: I/Os Menu

4.9 Visualization Menu

Press the **Visualization** button on the main window to access the visualization window. This window simply shows the current robot pose, which is e.g. useful if the robot is operated remotely without direct visual feedback. You can also see the previously saved poses in the 3D visualization. The visualization is interactive, and allows for a few controls:

- **Rotate** the image by clicking inside the visualization with your mouse (left click) and drag the image sideways as well as up and down.
- **Zoom** in and out by turning the wheel on your mouse upwards and downwards, respectively.
- **Shift** the camera sideways as well as up and down by using the arrow keys on your keyboard, or right click using a mouse and drag left and right.

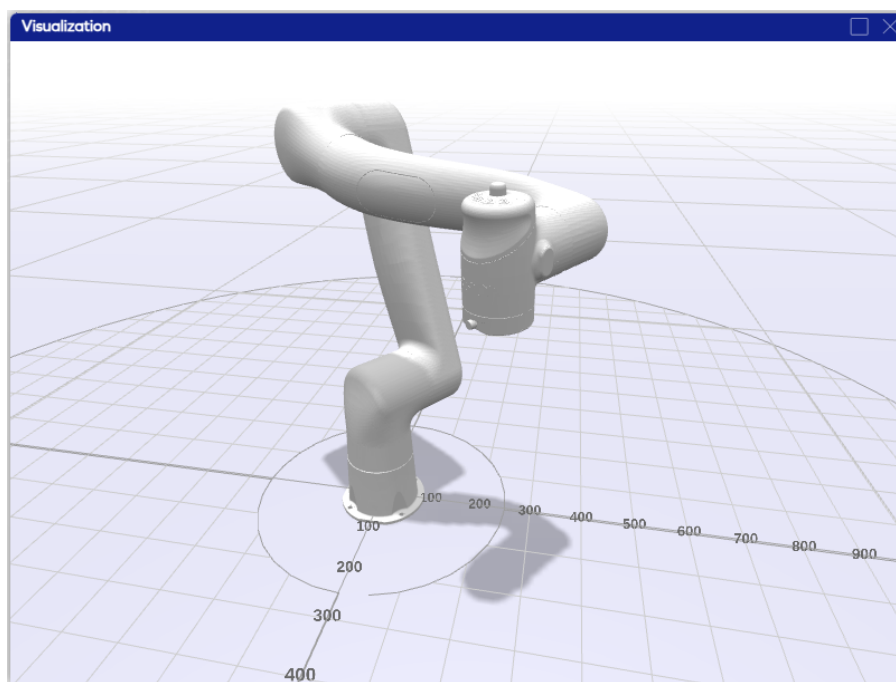


Figure 4.9: Visualization Window

4.10 Hand-Guided Control

Press the **Hand-Guided Control** button on the main window to access the hand-guided control menu. This menu lets the user specify which joints of the robot are allowed to move during HGC mode. By default, pressing the robot's HGC button (and the related button in the [Cockpit](#)) allows the user to move all joints. By clicking on the individual joint indicators in this menu, the user may activate and deactivate specific joints from being affected by HGC.

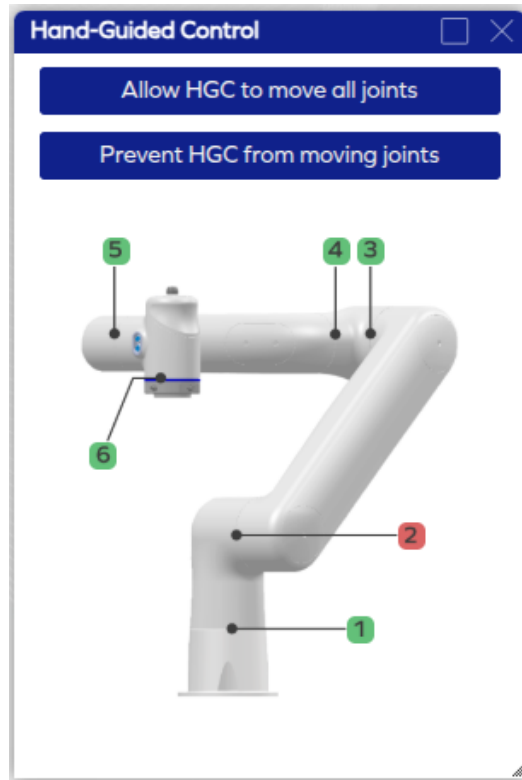


Figure 4.10: Hand-Guided Control Menu

4.11 System Monitor

Press the **System Monitor** button on the main window to access the system monitor menu. This menu contains the following system outputs for the user to observe:

4.11.1 Robot State

The **Robot State** tab shows the current state of the robot. This includes the coordinates of the tool center point (TCP) with respect to the base frame of the robot as well as the positions, currents, and temperature values of all motors. Also, the states of all safety inputs are visible here.

4.11.2 Application Output

Whenever an application returns text using the **print(...)** function, this text appears in the **Application Output** tab of this window. Note that if multiple applications are running in parallel, all of them may print text into the application output.

4.11.3 Application Log

Whenever an application logs data using the **log_message(...)** function, this data appears in the **Application Log** tab of this window. Here logged data can be filtered and searched. These custom logs will be downloaded together with the built-in log files if you use the keyboard shortcut **CTRL + SHIFT + L**.

4.11.4 Message History

Whenever a message pops up in the interface, it is also listed in the **Message History** tab of this window. The messages are separated into four types, which are used in different contexts:



Information messages provide useful messages, informative messages or hints to the user.



Success messages provide feedback to let the user know that a command or action was successfully executed.



Warnings provide important information to the user that must be read with caution before continuing. Warnings are never critical and do not interrupt the program flow.



There's a wide range of **errors**, from minor usability errors (e.g. wrong actions or function arguments) to critical errors caused by e.g. a damaged robot.

4.12 Help Menu

The **Help Menu** contains helpful information about the currently running system and modules, about keyboard shortcuts available to the user, and about contact possibilities in case of questions or problems.

4.12.1 Robot Information

The **Robot Information** tab of the help menu contains detailed information about the AX Portal environment and the connected hardware components (if connected). In the **Modules** section of the robot information you will find a list of the installed modules, and whether they are loaded properly. Only if the status shows “OK” you can use the functions and menus provided by this module. In case of support requests, please always mention the detailed robot information of AX Portal and the connected hardware. You can simply press on the “copy to clipboard” button and paste it (optimally in a [monospaced font](#)) along with your support request.

4.12.2 Keyboard Shortcuts

The **Keyboard Shortcuts** tab contains an overview of the available shortcuts. The shortcuts are ordered into general shortcuts, which are e.g. used to operate the control buttons in the cockpit, and application editor shortcuts, which are useful for programming AX Portal applications. Note that on macOS, the keyboard layout is significantly different compared to other operating systems (Windows, Linux), and therefore the list of shortcuts is different as well.

4.12.3 Code Documentation

The **Code Documentation** tab contains the documentation of all functions available for programming the applications. The documentation is separated into core functions that are always available, and add-on functions that refer to specific additionally installed modules. Using the **Download** button in the tab header, the code documentation can be downloaded as a ZIP file. To see the code documentation offline, open its “index.html” file in your browser.

4.12.4 Open Source License

The **Open Source License** tab contains all necessary information regarding open-source libraries and modules used by AX Portal.

4.12.5 Contact Information

The **Contact** tab shows support contact information. Please always consult the manuals first before posting support requests. For support requests, make sure to download the log files and send them with your request.

4.13 Settings Menu

In the **Settings Menu**, the user permanently changes some of the basic settings of AX Portal. Press **Save** to accept all changes, press **Cancel** to reject them, or press **Reset** to reset all settings to the latest saved values. In the following, all settings categories and options are explained:

The Settings Menu is organized into the following sections:

- GUI Settings**
 - Live Auto-Completion: ☒
 - Show Optional Function Arguments: ☒
 - Message Fading Level: Warning (dropdown)
 - Message Fading Time: 5 (slider)
- Customize Cockpit Buttons**
 - Show 'Hand-Guided Control' button: ☒
 - Show 'Gripper' button: ☒
 - Show 'Jogging' button: ☒
 - Show 'Motion Controls' button: ☒
 - Show 'Homing' button: ☒
 - Homing Position
 - Joint 1: 0
 - Joint 2: 0
 - Joint 3: 0
 - Joint 4: 0
 - Joint 5: 0
 - Joint 6: 0
- Start-Up Settings**
 - Start-Up Project: Last Selected (dropdown)
 - Start-Up Application: Last Selected (dropdown)
 - Triggering Events: No Trigger (dropdown)
- Motion Behaviors**
 - Interrupt Behavior: Pause and Approve (dropdown)
 - Collision Resume Attempts: 3 (slider)
 - Collision Resume Timeout: 20 (slider)
 - Temperature Speed Reduction: ☒
- Authentication Settings**
 - Disable Authentication for ROS: ☒
 - Disable Authentication for TCP: ☒
 - Disable Authentication for REST: ☒
 - Disable Authentication for Modbus: ☒
- Default TCP Communication Settings**
 - Encoding Standard: UTF-8 (text input)
 - End Character: \x03 (text input)
 - Send Heartbeat: ☒
- ROS Function Settings**
 - Discovery Mode: Peer to Peer (dropdown)
 - IP Adresses of ROS 2 Peers: 127.0.0.1 (text input, with + and - buttons)
 - ROS Domain ID: 10 (dropdown)

Buttons at the bottom: Cancel, Reset, Save.

Figure 4.13: Settings Menu

4.13.1 GUI Settings

This category contains several user interface related settings.

Live Auto-Completion	Enable this option to see hints for auto-completing text live while typing code in the application script editor. If disabled, you may still auto-complete the current word using the shortcut CTRL+SPACE.
Show Optional Function Arguments	Choose whether inserting function snippets into the code shows or hides the optional function arguments. In order to keep the application code clean of unnecessary function arguments we recommend disabling this option. If you are not yet familiar with the built-in functions however, we recommend enabling this option.
Message Fading Level	Choose up to which level the appearing messages are fading. To make all messages fade, choose the highest possible level. To make all messages stick, choose the lowest possible level.
Message Fading Time	Specify the fading time of fading messages in seconds (between 1 and 60).

4.13.2 Customize Cockpit Buttons

These settings simply allow for the user to enable and disable the control buttons appearing in the cockpit of the graphical user interface both for the [Desktop Interface](#) as well as for the [Panel Interface](#). The detailed functionality of these buttons is explained in the [Cockpit](#) section. It is convenient to disable a button if the user either doesn't want to or can't use it (e.g. you can't use the gripper button if you don't have an interactive tool attached) or if the user doesn't want an operator to use it via the Panel Interface.

Show * Button	Enable to show this button, disable to hide it.
Homing Position	Specify the angles of the homing position, which is the position linked to the Homing Button accessible in the cockpit. Enter one value for each joint angle.

4.13.3 Start-Up Settings

In this category, all settings refer to the start-up procedure of AX Portal. Modifying these settings helps to avoid repetitive actions upon start-up. Choose the list entry **Last Selected** (default entry) for some start-up settings to remember and reload its last session.

Start-Up Project	Select the project that loads at start-up. All defined projects appear in this list.
Start-Up Application	In case a project is selected, choose the application that loads start-up. All defined main applications of that project appear in this list.
Triggering Events	Choose, if you want the robot to automatically power-up or even execute an application at start-up. By default, this option is disabled. In order to select a trigger, some of the start-up settings listed above have to be selected first, to prevent the user from accidentally executing unintended applications.

4.13.4 Motion Behaviors

In this category, all settings refer to behaviors during robot movements.

Interrupt Behavior	Select the motion behavior on interrupt. For more details about the different interrupt behaviors and how to resolve interrupts, refer to the following sub-sections “Interrupt Behavior” and “Interrupt Dialog”.
Collision Resume Attempts & Timeout	Set the maximum number of attempts of resuming after a collision occurred. If too many collisions occur within the defined timeout, the robot will abort resuming.
Temperature Speed Reduction	Choose whether the robot speed is automatically reduced in case of high drive temperatures. This setting is enabled by default, which has the effect that above 70 °C movements automatically slow down to prevent overheating. If temperatures above 90 °C are detected, a critical error is raised to avoid damage due to overheating. More details about this feature are described in the “Performance Limitations” section of the “Collaborative Robot: 6-Axis Robots AX6 Manual”.

4.13.4.1 Interrupt Behavior

Interrupts can occur due to different reasons like:

- an SS1 stop event was triggered
- an SS2 stop event was triggered
- a collision of the robot with a person, an object, or the environment happened
- a safeguard interrupt was triggered

A collision is detected whenever the currents of the robot joints deviate from the robot's dynamic model beyond a defined threshold. Collision detection is a safety feature that provides an additional layer of protection; it must not be used intentionally as part of an application, as repeated collisions place increased stress on the robot's joints and accelerate wear.

Note that for collision detection to work properly, the programmer is required to adapt the robot's payload to the correct value whenever an object is being picked or placed.

In addition to physical collisions, external safety devices such as light barriers, or similar safeguards can be connected to the robot. These triggers act preventively, before physical contact occurs, but are

handled in the same way as collision events. The Interrupt Behavior defines how both collision- and safeguard-triggered interrupts are handled by the system.

Right now, this is the suggested flow on Interrupt Behaviors:

Pause and Approve	causes the running application to pause and waits for approval by the user (dialog window) to continue or stop the application.
--------------------------	---

4.13.4.2 Interrupt Dialog

The **Interrupt Dialog** is displayed whenever an interrupt condition occurs during robot operation. Typical interrupt sources include detected collisions, activation of an emergency stop, safeguard triggers, or other safety-relevant events. The dialog guides the user through resolving the condition in a safe and controlled manner.

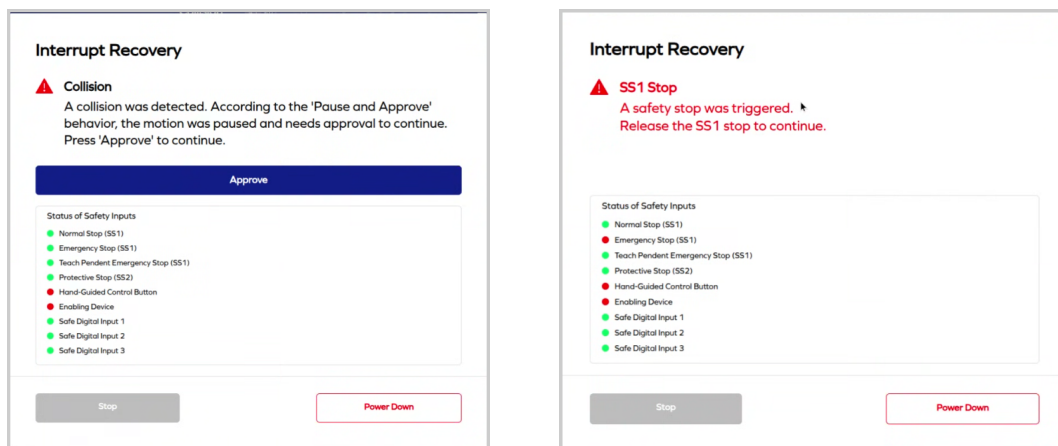


Figure 4.13.4.2: Interrupt Recovery

The dialog is divided into two main sections:

Upper section – Interrupt information and resolution: This area describes the currently detected interrupt condition and provides clear instructions on how to resolve it. Examples include:

- Clearing the robot's path after a collision.
- Releasing an emergency stop or other safety devices.
- Resetting a safety condition when required.

Depending on the interrupt type, an action button such as **Approve**, **Proceed**, or **Reset** is shown. The user must follow the instructions and press the indicated button to continue operation.

Lower section – Status of safety inputs: The lower half of the dialog displays the real-time status of all relevant safety inputs. This allows the user to quickly identify which safety device is currently active, for example:

- Emergency stop (including teach pendant emergency stop)
- Protective stop
- Enabling device

- Hand-guided control button
- Safe digital inputs

Green indicators signal a normal state, while red indicators show an active or blocking condition (or the button not being pressed).

Stop Option

The **Stop** button is available whenever the user needs to choose between continuing the previous application/motion or stopping it. Pressing the button stops the robot and returns it to the "Ready" state, where it is ready for new actions.

Power Down Option

The **Power Down** button is always available in the Interrupt Dialog. It allows the robot to be powered down at any time. Note that powering down does **not** automatically resolve the interrupt condition; the underlying cause (e.g. a pressed emergency stop) may still need to be cleared before normal operation can resume after power-up.

Typical Workflow

1. An interrupt occurs and the dialog is shown automatically.
2. Read the description of the detected condition in the upper section.
3. Check the safety input status in the lower section to identify the active cause.
4. Physically resolve the condition (e.g. release emergency stop, remove obstacle).
5. Press the indicated action button (**Approve**, **Proceed**, or **Reset**) to continue.
6. If required, repeat the action until the robot resumes normal operation.

SLP Violation

One special case for interrupt recovery is the SLP (safe limited position) violation, which occurs if the robot attempts to move outside of the position limits specified in the [safety settings](#). If this happens, the interrupt dialog guides the user to switch to manual operation mode, in which the user will be able to jog out of the position limit. Except for jogging, movements are not allowed when the robot is in a SLP violation. If the robot is powered down, or it starts up with an active SLP violation, then do the following:

1. Switch to manual operation mode
2. Power-up the robot
3. Navigate to the [jogging menu](#)
4. Jog the robot out of its position limits

4.14 Safety Settings Menu

The Safety Settings Menu is the gateway to the safety system of the AX series robot. This screen is designed for **Safety Officers**. Other users may only read and extract the safety parameters, but they may not change them.

4.14.1 User Interface

Upon entering the **Safety Settings Menu**, this menu shows all of the safety parameters that are currently set in the safety system of the robot. Every user can see these settings, however, only a **Safety Officer** can change them. This menu is separated into sub-menus, which you can navigate through using the page numbers or forward and backward button at the bottom of the window.

Export	By pressing the Export button, the safety settings are exported as a *.yaml file. This file can be stored to back-up the safety settings, and can also be imported on other robots to duplicate the safety setup.
Edit	The Safety Officer may click the Edit button (and enter his user password) to enter edit mode.

Once in edit mode, the settings of all sub-menus become editable. The safety officer can now change the safety limitations according to the robot's mounting setup and use-case. For better understanding of which safety settings have been changed, changed fields are marked in blue color.

Import	By pressing the Import button, a previously exported safety settings file can be loaded. This overwrites the manually changed values of all sub-menus.
Apply	By pressing the Apply button (and entering the user password), all safety settings are sent to the safety system, which verifies the flags and limitations. If the verification does not succeed, invalid entries appear in red color in the menu, so you will easily find them. If the verification is successful, the menu switches to verification mode. Note that the values are saved but not verified at this point. Verifying the settings is necessary to power-up the robot.
Cancel	By pressing the Cancel button, the safety settings are not applied and all changes are rejected. The menu then switches back to (default) observation mode.

Once the safety officer is done with editing the safety settings, and all values have been successfully applied, this menu is in verification mode. The previously applied but not yet validated settings are marked in yellow color.

Verify	By pressing the Verify Page button, the safety officer officially marks the settings of the currently visible sub-menu as accepted, and the next sub-menu is selected. Pressing the Finish Verification button of the last sub-menu is the final step towards saving the settings.
---------------	--

After verifying the new values, it is mandatory to validate the behavior of the safety system. This can be done by powering up in [Safety Validation Mode](#), which allows the software to ignore all safety limits during the validation process to test that the safety system triggers as expected.

Safety Settings

Safely Limited Position (SLP)

Defining the areas in the robot's base frame in which the safety settings are applied. ⓘ

Note that a safety margin of 60 mm is subtracted from the specified position limits to determine the effective limits.

	Default		Safe Case 1		Safe Case 2		Safe Case 3		
	min	max	min	max	min	max	min	max	
x	-500	3000	-3000	3000	-3000	3000	-3000	3000	mm
y	-800	800	-3000	3000	-3000	3000	-3000	3000	mm
z	-3000	3000	100	-100	-3000	3000	-3000	3000	mm

Defines whether the limiting area applies to the tool center point (TCP) or all movable parts of the robot.

Limit only the TCP of the robot ☒ Limit all parts of the robot

Tool Safety Zone

Defining a safety zone for the tool and any objects it holds. This zone, represented as a sphere with a specified radius, is offset from the center of the flange frame. This zone cannot enter the workspace boundaries. ⓘ

Tool Safety Offset mm

Tool Safety Radius mm

Cancel

< 1 2 3 4 5 6 >

Import

Apply

Figure 4.14.1: Safety Settings Menu while editing settings

83

4.14.2 Safety Settings

Most of the safety settings (across the sub-menus) are separated in 4 columns. These columns define the **Safe Cases** of the robot. A safe case is triggered when the respective **Safe Digital Input** (SDI 1-3) is being triggered. The **Default** parameters are always active, regardless of the Safe Digital Inputs.

Safe Digital Outputs (SDO) are not configurable and their electrical specification is described in the “Robot Controller RC-A101 Manual”.

For some settings a safety margin is hardcoded. This value specifies the acceptable range for the software to trigger earlier than the safety system, in order to prevent a safety violation.

4.14.2.1 Safely Limited Position (SLP)

The first sub-menu of the safety settings menu contains the Safely Limited Position. Here, the position of the robot can be limited in the x-, y-, and z-directions, expressed in the robot's base frame. If the coordinate frames are unclear, a click on the information sign shows an image with the coordinates. You may either limit the position of all parts of the robot (incl. base and links), or you may just limit the position of the tool center point (TCP) of the robot.

Safety Settings

Safely Limited Position (SLP)

Defining the areas in the robot's base frame in which the safety settings are applied. ⓘ

Note that a safety margin of **60 mm** is subtracted from the specified position limits to determine the effective limits.

	Default		Safe Case 1		Safe Case 2		Safe Case 3		
	min	max	min	max	min	max	min	max	
x	-3000	3000	-3000	3000	-3000	3000	-3000	3000	mm
y	-3000	3000	-3000	3000	-3000	3000	-3000	3000	mm
z	-3000	3000	-3000	3000	-3000	3000	-3000	3000	mm

Defines whether the limiting area applies to the tool center point (TCP) or all movable parts of the robot.

Limit only the TCP of the robot ☐ Limit all parts of the robot

Tool Safety Zone

Defining a safety zone for the tool and any objects it holds. This zone, represented as a sphere with a specified radius, is offset from the center of the flange frame. This zone cannot enter the workspace boundaries. ⓘ

Tool Safety Offset mm Tool Safety Radius mm

Export < 1 2 3 4 5 6 > **Edit** **Start Verification**

Figure 4.14.2.1: Example Settings for SLP

4.14.2.2 Safely Limited Joint Angles

The second sub-menu of the safety settings menu contains the Safely Limited Joint Angles. In the default state, Safely Limited Joint Angles are disabled. By enabling a safe case, angular position of the joints can be monitored.

However, in a safe case, the monitoring range for joint angles is limited to a range of $[-180^\circ, 180^\circ]$. As a result, the robot's operating range may be restricted compared to the actual physical limits of the robot. To avoid this restriction, monitoring can be disabled individually for joints 1, 4, and 6.

Safely Limited Joint Angles are only active when both the lower and upper limits are defined. If both limits are left undefined, monitoring for that joint is disabled.

Safety Settings

Safely Limited Joint Angles

Defining the position limits of each joint of the robot. ⓘ

The default position limits are determined by the physical constraints of the joints and cannot be modified.

Note that some joints can move beyond **180 deg**, but safe cases do not support these ranges. Disabling the limits allows the full joint range to be used.

Note that a safety margin of **5 deg** is subtracted from the specified position limits to determine the effective limits.

	Default		Safe Case 1		Safe Case 2		Safe Case 3		
	min	max	min	max	min	max	min	max	
Joint 6	Disabled		Dis.	Dis.	Dis.	Dis.	Dis.	Dis.	deg
Joint 5	Disabled		-160	160	-160	160	-160	160	deg
Joint 4	Disabled		Dis.	Dis.	-100	100	Dis.	Dis.	deg
Joint 3	Disabled		-165	165	-165	165	-165	165	deg
Joint 2	Disabled		-160	160	-160	160	-160	160	deg
Joint 1	Disabled		Dis.	Dis.	Dis.	Dis.	-179.9	179.9	deg

Export < 1 2 3 4 5 6 > Edit Start Verification

Figure 4.14.2.2: Example Settings for Safely Limited Joint Angles

4.14.2.3 Safely Limited Speed (SLS)

The third sub-menu of the safety settings menu contains the safely limited speed. Here, the speed of each of the robot's joints can be limited individually, as well as the motion speed of all parts of the robot, including the tool, base and links.

Safety Settings

Safely Limited Speed (SLS)

Defining the speed limits of each joint and all movable parts of the robot.

Note that a safety margin of **7 deg/s** or **100 mm/s** is subtracted from the specified speed limits to determine the effective limits.

	Default	Safe Case 1	Safe Case 2	Safe Case 3	
Joint Velocity 6	247	247	247	247	deg/s
Joint Velocity 5	247	247	247	247	deg/s
Joint Velocity 4	247	247	247	247	deg/s
Joint Velocity 3	187	187	187	187	deg/s
Joint Velocity 2	187	187	187	187	deg/s
Joint Velocity 1	187	187	187	187	deg/s
Speed	2000	2000	2000	2000	mm/s

Export < 1 2 3 4 5 6 > Edit Start Verification

Figure 4.14.2.3: Example Settings for SLS

4.14.2.4 Safe Power and Force Limit

The fourth sub-menu of the safety settings menu contains power and force related settings:

- For each safe case the user may define a different force limit that defines how sensitive the robot is to external forces like collisions with people or objects. The force limit is applied at the tool center point (TCP).
- Setting the average payload (and proper payload values during operation) which is defined as the tool weight plus items grabbed is important to make the force limitations more sensitive to your specific application.
- The gravity vector of the robot must match the robot's mounting. If the robot is mounted on the floor, the gravity vector is $[0, 0, -9.81]$. If the robot is mounted on a wall, sloped surface, or on the ceiling however, the gravity vector must be calculated first, and properly entered here.

The screenshot shows a software window titled "Safety Settings" with a blue header bar. Inside, the "Safe Power and Force Limit (PFL)" section is active, displaying a description and four input fields for Force Limit (Default, Safe Case 1, Safe Case 2, Safe Case 3) all set to 250 N. Below this is the "Average Payload" section with a single input field set to 0 kg. The "Gravity Vector" section shows three input fields for x, y, and z components, with x and y set to 0 and z set to -9.81 m/s². At the bottom, there is a navigation bar with buttons for "Export", a set of numbered tabs (1-6) with tab 4 selected, and "Edit" and "Start Verification" buttons.

	Default	Safe Case 1	Safe Case 2	Safe Case 3	
Force Limit	250	250	250	250	N

Average Payload: 0 kg

	x	y	z	
Gravity Vector	0	0	-9.81	m/s ²

Figure 4.14.2.4: Safe Power and Force Limit Settings

4.14.2.5 Safety I/Os and Operator Limitations

The fifth sub-menu of the safety settings menu allows configuration of safety-related I/O behavior and operator permissions. Users can define how protective stops are handled, whether inputs are observed in manual mode, reset conditions, mode-switching permissions, and password requirements for operation changes.

The screenshot shows a web-based configuration window titled "Safety Settings". It contains two main sections: "Safe I/O Configuration" and "Operator Limitations".

Safe I/O Configuration

- Defining whether a protective stop input results in a SS1 or SS2 stop.
 - Results in a SS2 Stop: ☐
 - Results in a SS1 Stop: ☒
- Defining whether a protective stop input is always being normally observed, or if it is to be ignored during manual mode.
 - Ignore in manual mode: ☐
 - Observe in manual mode: ☒
- Defining whether resetting interrupts can only be triggered via digital inputs.
 - Trigger always: ☐
 - Digital inputs only: ☒

Operator Limitations

- Defining whether operators are able to switch the operation mode.
 - Allow switching mode: ☐
 - Don't allow switching mode: ☒
- Defining whether operators need to enter their password to switch the operation mode.
 - No password required: ☐
 - Password required: ☒

At the bottom of the window, there is a navigation bar with an "Export" button, a series of numbered tabs (1-6) with tab 5 selected, an "Edit" button, and a "Start Verification" button.

Figure 4.14.2.5: Safe I/Os Configuration Settings

4.14.2.6 Verification Data

The last sub-menu of the safety settings contains information about the safety device:

- The checksums read from the safety device (used for troubleshooting)
- The transfer counter (marks how often the safety parameters have been written)
- When the safety parameters have been last updated
- Whether the safety parameters have been verified
- The serial number and firmware version of the safety device

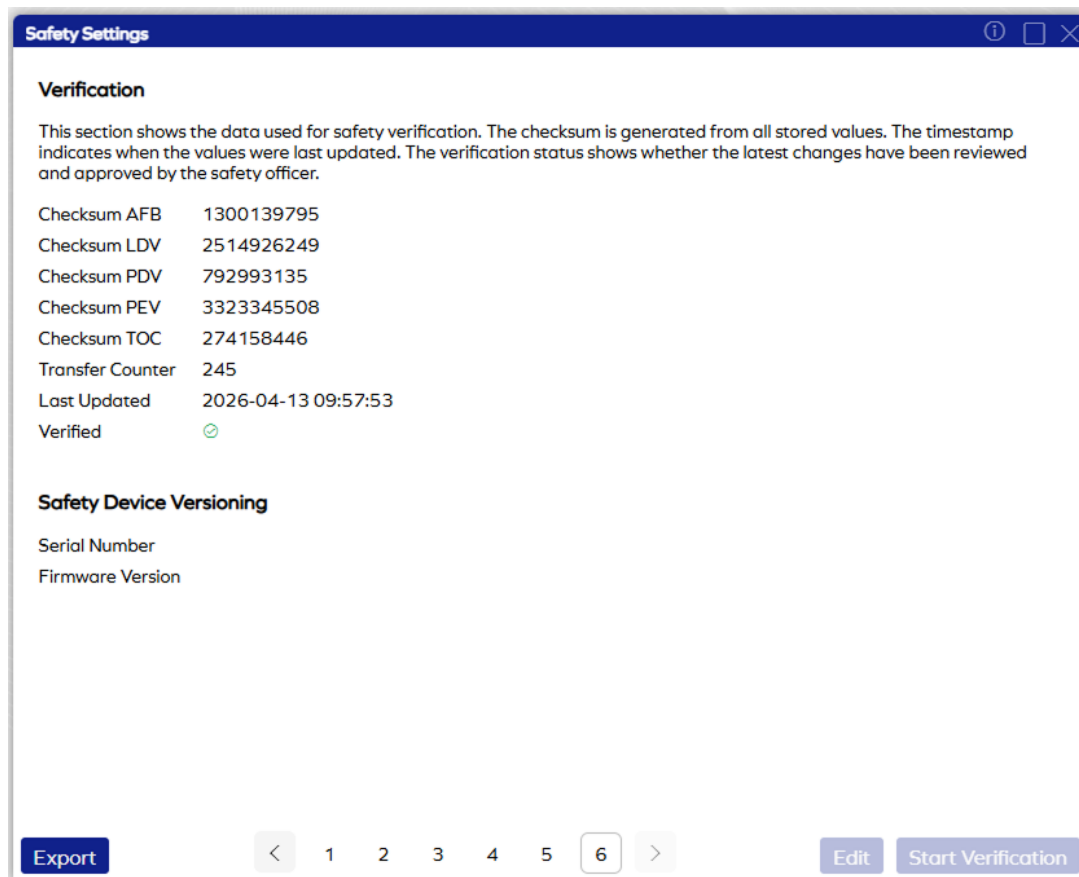


Figure 4.14.2.6: Verification Settings

4.14.2.7 Safety Output Functions

The logic states of specific safety functions are connected with the safety outputs of the robot controller. For detailed information consult the "Robot Controller RC-A101 Manual".

4.14.2.8 Connecting Optional Devices

For connecting optional devices such as Tablet Holder or Enabling Device please consult the "Robot Controller RC-A101 Manual".

5. Panel Interface (GUI)

5.1 Introduction

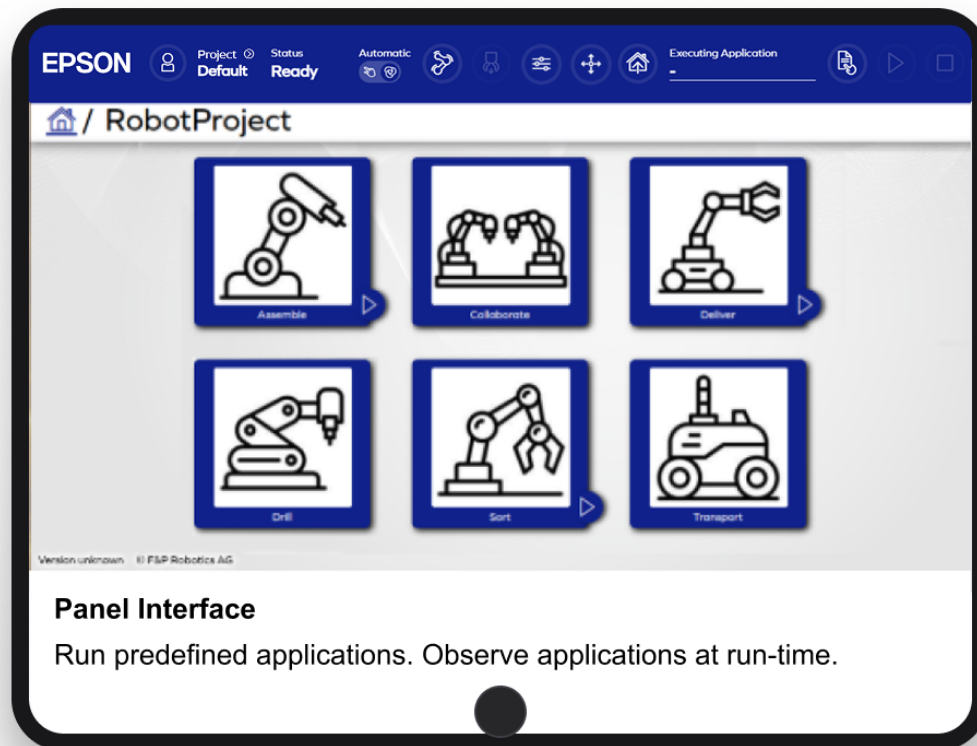


Figure 5.1.a: Example Screenshot of the Panel Interface

The panel interface is an alternative to using the [Desktop Interface](#) to quickly access your main applications. This interface has been designed to be fully adaptable to customer needs and especially to only display the applications (and information about their state). The panel interface is touch screen compatible and its design resembles the well-known app-like structure of state-of-the-art mobile devices, making it intuitive even for users without technical background. Also, using the panel interface, the user does not have to care about programming or the code behind the application. Just click on the appropriate app and the application starts running. This interface can be accessed via the [Admin Interface](#), by logging in and using the “Go to Panel Interface” button.

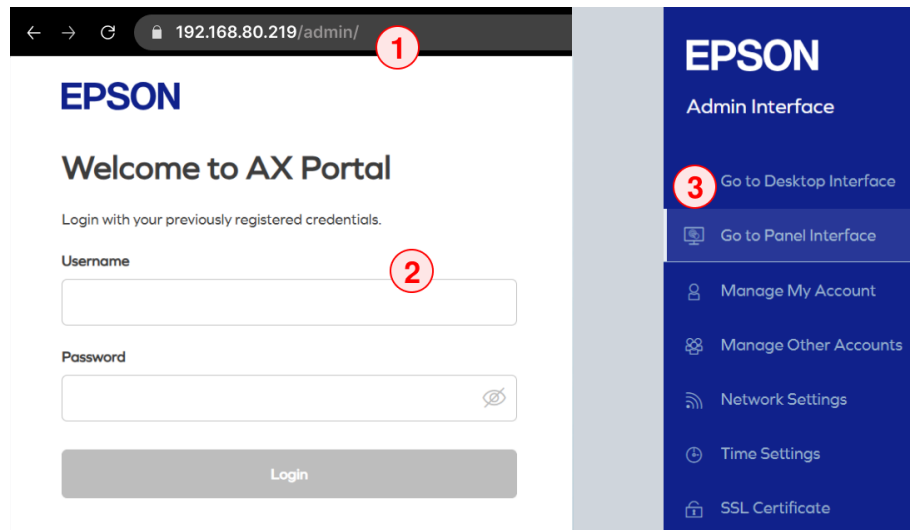


Figure 5.1.b: Accessing the Panel Interface

In some browsers (especially on Windows), the display scaling/zoom is set to a high number (e.g. 150%) by default, which may cause parts of the interface (such as cockpit or logo) to appear cut-off. For the best experience, we recommend setting the display scaling/zoom to 100% (or even lower).

5.2 Configure Applications for Panel Interface

By default, the Panel Interface does not show any applications. The user first needs to specify which application to show there for quick execution by the operator. In order to configure a panel application, follow these steps:

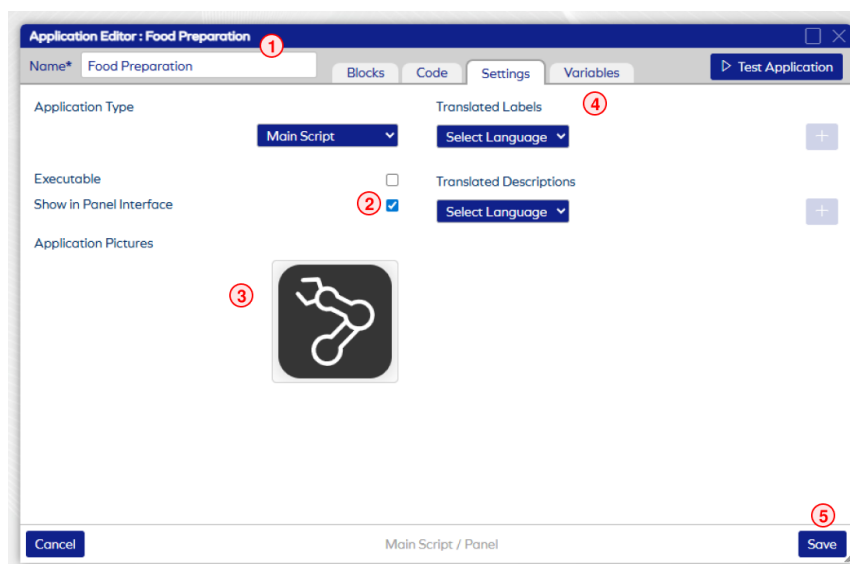


Figure 5.2.a: Panel Interface Relevant Settings in the Application Editor (Desktop Interface)
Settings Relevant for Panel Interface: 2., 3.,4.

1. Navigate to the Desktop Interface, edit the application, go to the [Application Settings](#),
2. enable the **Show in Panel Interface** flag,
3. optionally upload an appropriate **Application Picture** (the optimal size ratio is 5:4), which later shows together with the name of the application in the main menu of the Panel Interface,
4. optionally navigate to the [Application Variables](#) tab to add **State Variables**, which allows you to monitor and/or modify variables via the Panel Interface,
5. and finally **Save** the application.

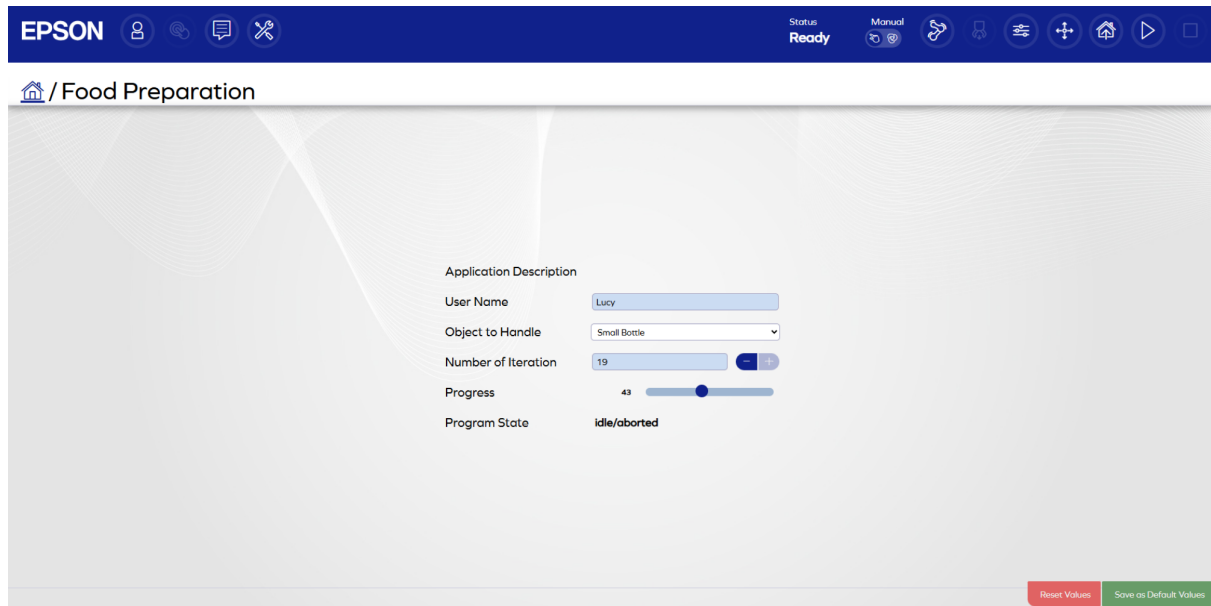


Figure 5.2.b: Panel Application with State Variables

5.3 Cockpit and Settings

The cockpit of the Panel Interface shows more or less the same buttons as the cockpit of the [Desktop Interface](#). Please refer to the section [Cockpit](#) for a detailed explanation of all the elements. In addition to these elements, you will see some different buttons on the left side:



The **Panel** button opens up the main panel page showing the applications to select and run.



The **Message** button opens the message history where all past messages including warnings and errors are visible in chronological order.



The **Settings** button leads you to the settings window, which contains only the most important settings for quick-access, namely

- selecting the connected tool,
 - powering up and down the robot,
 - downloading the log files.
-

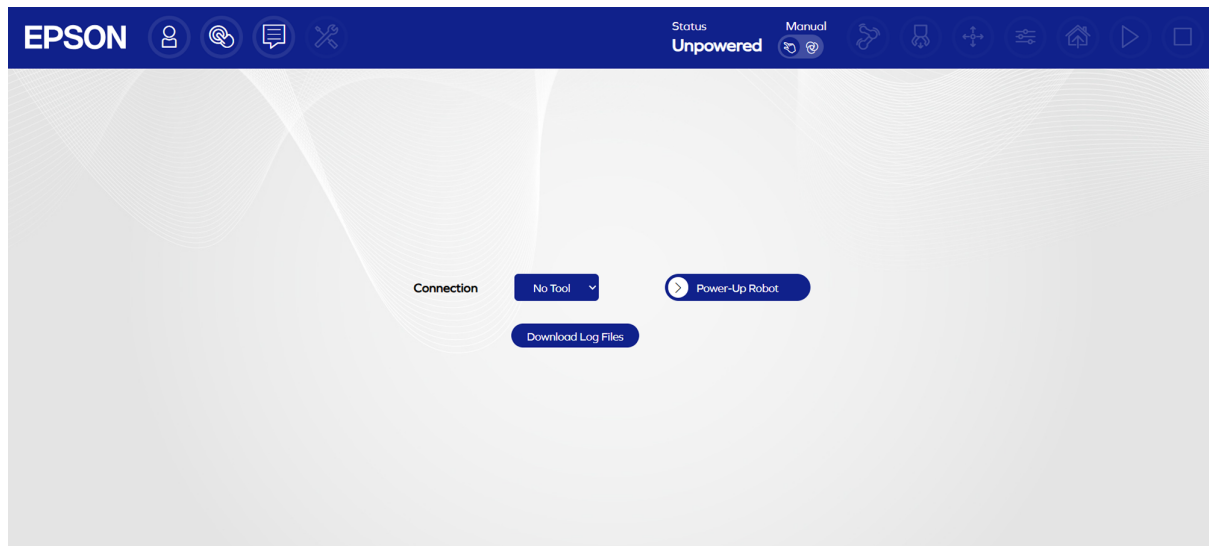


Figure 5.3: Panel Settings Window

5.4 Controlling Applications

In the main window of the Panel Interface all applications are listed in an app-like structure, showing their name and application picture. The panel differentiates between two kind of applications:

- Applications **without state variables** show a play button. Press on the application button to directly run this application.
- Applications **containing state variables** show without a play button. Press on the application button to enter the application detail view showing all state variables. In this window you can change the editable state variables and press the play button to execute this application. Press the back button to return to the main panel window.

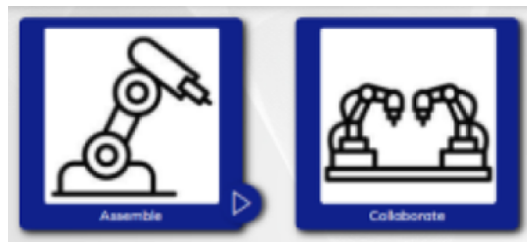


Figure 5.4: Application Buttons

6. Kinematic Definitions

Robot kinematics describes how the robot moves and how its movements can be represented. To start working with kinematics, the user must understand the different definitions of the kinematic parameters and objects available:

6.1 Coordinates

Coordinates define an absolute robot position and orientation. The position is a vector with three values $[x, y, z]$ given in mm and the orientation is also a vector with three values $[\text{roll}, \text{pitch}, \text{yaw}]$ (nautical angles) given in deg.

Initialization can be done as follows:

```
coord = Coordinates([600, 0, 300, 180, -90, 0])
# ... this is similar to ...
coord = Coordinates({"x": 600, "y": 0, "z": 300, "roll": 180, "pitch": -90, "yaw": 0})
```

Accessing elements can be done as follows:

```
coord = Coordinates([600, 0, 300, 180, -90, 0])

# accessing individual elements (x, y, z, roll, pitch, yaw)
x = coord.x
x = coord["x"]
x = coord[0]

# accessing position or orientation
position = coord.position
orientation = coord.orientation
```

6.1.1 Nautical Angles

Nautical angles are one of many ways to define the orientation part (roll, pitch, yaw) of the coordinates. Nautical angles are commonly used for the orientation of airplanes. Using nautical angles, a three-dimensional rotation is defined as

- 1st a rotation around the negative z-axis in yaw (ψ) direction,
- 2nd a rotation around the negative y-axis in pitch (θ) direction and
- 3rd a rotation around the positive x-axis in roll (ϕ) direction.

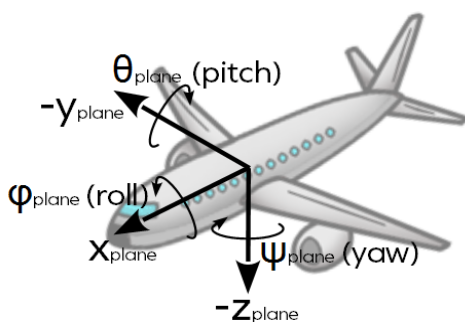


Figure 6.1.1.a: Nautical Coordinate System

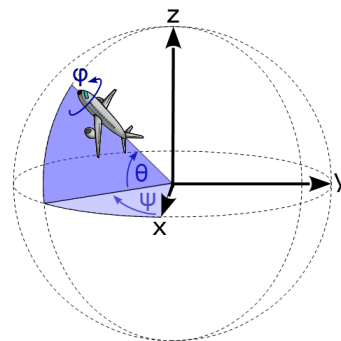


Figure 6.1.1.b: Nautical Order of Rotations

Hint: Whenever a function requires you to deal with three dimensional orientations of the end-effector, it helps to imagine a plane being attached to the end-effector, such that the coordinate system of both the end-effector and the plane coincide. This way it is much easier to extract the angles (roll, pitch, and yaw) that lead to this very orientation.

6.1.2 Coordinate Systems

For the manipulation of AX series robots, different coordinate systems are important to the user:

- The **Base Frame** of a robot is defined such that its z-axis points away from the base plate and the x-axis is parallel to the base plate and points away from the connectors at the back panel of the base. The well-known right-hand rule then defines the direction of the y-axis.
- The **Work Frame** of a robot is a user defined frame relative to the base frame and is equal to the base frame unless otherwise specified. The coordinates will be given in this frame. The user can use this frame to define a custom workspace for the robot.
- The **Flange Frame** of a robot differs for each type of tool connected to the robot:
 - o If no tool is mounted, the origin of this frame is located in the center of the flange (connection plate) on top of the robot. Its orientation matches the orientation of the base frame if all joint angles are at zero.
 - o If a custom tool is mounted (and defined via tool editor), the frame is shifted by the position and orientation specified in this tool.
- The **Tool Frame** of a robot is a user defined frame relative to the flange frame and is equal to flange frame unless otherwise specified. The coordinates will be reached in the origin of this frame. The user can use this frame to define the tool center point.

For an example of these coordinate systems, please refer to the following figure:

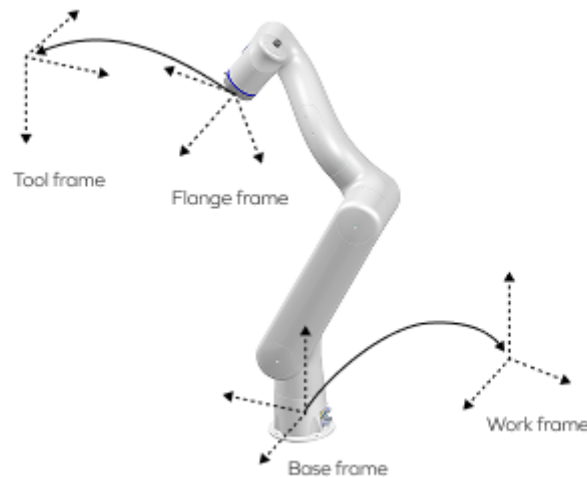


Figure 6.1.2: Robot Coordinate Frames

6.2 Configuration

For a given robot position and orientation, the inverse kinematics calculation can produce several different joint configurations. These configurations differ in how the joints bend or rotate. The configuration setting is used to select one specific joint configuration from these possible options. For a typical six-axis robot, different combinations of the shoulder, elbow, and wrist directions can result in up to eight different configurations, even when the end-effector position and orientation are the same. To identify these configurations, they are named from “c1” to “c8”.

Note that configurations are not suitable to compare two poses or to check whether a path is seamless. A change in configuration does not necessarily make a path non-seamless. Also a non-seamless path must not necessarily include configuration changes.

6.3 Pose

A pose defines the coordinates and configuration of the robot. It can be defined by the ID or name of a predefined pose in the database. Alternatively, it can be created by specifying the coordinates and configuration and using the create pose function or by specifying the joint angles and using the forward kinematics function.

Initialization can be done as follows:

```
# read predefined pose from database by index
pose = Pose(1)
# read predefined pose from database by name
pose = Pose("fancy_pose")
```

6.4 Point

A point is a discrete element of a path. In addition to the robot pose, it also specifies the robot velocity and the time interval which has passed since the previous point. This is called PVT (position, velocity, time) and is usually saved as a vector of joint angles, vector of joint velocities and time interval float. Note that an extended version of a point might also detail the robot acceleration.

6.5 Path

A path defines how the robot moves between two or more poses. A continuous path is defined only in tool space by multiple segments which can be linear or circular. The path is then discretized and converted into joint space. A path is called seamless, if the discrete solutions to the path do not require the robot to reposition itself (configuration change). A path is called continuous, if the acceleration is continuous along the path.

A path starts at a specified pose (as opposed to coordinates). Segments of the path usually define coordinates, as the configuration of the path is defined by the initial pose. By using the closest solution of the inverse kinematic function to each previous discrete point, the path will automatically be seamless if possible. Note that we use the words path and trajectory interchangeably even though a path does technically not contain velocity information.

Initialization can be done as follows:

```
# read predefined path from database by name
path = Path("fancy_path")
```

6.6 Segment

A segment represents an individual part of a path that defines how the TCP moves towards the target.

The available segment types are:

- **Linear:** The TCP moves in a straight line to the target coordinates defined by the segment.
- **Circular:** The TCP follows a circular path from its current pose, through a defined intermediate position to the target coordinates of the segment. Note that the TCP orientation during this motion is determined automatically by the algorithm.
- **Coordinates:** The TCP moves to the target coordinates using the most efficient path. This corresponds to a linear movement in joint space.
- **Pose:** The TCP moves to the target pose using the most efficient path. This corresponds to a linear movement in joint space.
- **Orientation:** The TCP performs a pure rotation movement to reach the specified target orientation while maintaining its current position.

For each segment, velocity and acceleration parameters can be defined:

- For **Orientation** segments, these correspond to the angular velocity in [deg/s] and angular acceleration in [deg/s²]
- For all other segment types, they correspond to the linear velocity in [mm/s] and linear acceleration in [mm/s²]

Additionally, a blend radius can be specified, which defines the maximum allowable deviation in mm from the end pose of the previous segment corresponding to the start pose of the current segment during the transition. A blend radius of 0 mm forces the TCP to pass exactly through the defined pose, whereas a higher value enables smoother, more natural transitions by allowing the path to curve between segments.

6.7 Grid

Grids are customizable, up to three-dimensional arrays of points that define e.g. locations of predefined trays and can be used to bundle, and store sets of coordinate values. Usually, a grid is interpreted as a two-dimensional array, though it may as well be a simple list of points without depending on a second dimension (e.g. for picking objects from a conveyor belt), or even a three-dimensional structure (e.g. stacking objects in boxes).

7. Connection Modules

In addition to the user-friendly and intuitive GUIs for controlling and observing the robot, AX Portal also comes with a set of connection modules that allow the robot to be controlled and observed using different communication methods like TCP, REST, ROS, and Modbus. Each of these communication methods is suited to different use cases:

- **TCP** is ideal for real-time communication where fast and continuous data exchange is required. It is commonly used for direct control applications, custom client implementations, or when integrating with systems that require persistent socket connections.
- **REST** is best suited for integration with web-based services or applications. It is typically used when ease of use, scalability, and compatibility with standard HTTP-based systems are more important than real-time performance.
- **ROS** (Robot Operating System) is designed for robotics research and development. It is particularly useful when integrating the robot into complex robotic systems, enabling features such as sensor fusion, autonomous behavior, and interaction with other ROS-enabled devices.
- **Modbus** is commonly used in industrial automation environments. It is well-suited for communication with PLCs (Programmable Logic Controllers) and other industrial equipment where reliability and standardization are critical.

The usage of these modules is explained in this chapter.

7.1 Connections, Authentication, and Active Control

Before using any connection module, it is important that you understand how these modules connect to clients, how the authentication works, and how the principle of active control works in AX Portal.

7.1.1 Connection Channels

The connection between a device and the robot always goes via a specific connection channel. These connection channels are implemented a bit differently for the different connection modules.

The **TCP module** supports **multiple channels**. This means that each device that wants to connect to the robot creates its own connection channel, and logs in separately. Closing one of these channels closes the connection and also releases active control (see below).

 WARNING	The REST, ROS, and Modbus modules only come with a single channel. This means that once the connection to one of these modules is established and authenticated, any device can send commands to the robot. These channels are created on start-up of AX Portal and cannot be closed by connecting devices.
---	--

7.1.2 Authentication

AX Portal uses a secure two-step authentication. First, please follow the remote login procedure described in the [Secure Authentication](#) section to get a valid access token. This step must always be done via a **separate TCP connection**. Once you successfully logged in and created an access token, you can register this token via the chosen connection channel to authenticate your user using the **token_login** command.

**CAUTION**

The authentication step may be skipped if you disable the authentication via TCP in the settings menu.

Note that the Modbus Module does not come with a possibility to send a **token_login** command, hence the authentication of this module must be disabled in the [Settings Menu](#) to be able to use it.

7.1.3 Active Control Station

AX Portal comes with protection of the active control station, such that other channels cannot control the robot at the same time. Not having this protection could potentially lead to dangerous situations.

The command **request_active_control** is used by a connection channel to become the actively controlling station. Once this command runs through successfully, this control station has permanent control, until the channel is closed or the robot loses power. If another control station has control, this command blocks until that control station accepts the control request.

The command **release_active_control** is used by the active control station to give away the active control. It is recommended to always release control once the channel is no longer actively used. Otherwise this channel will block the active control permanently.

7.2 TCP Connection Module

7.2.1 Establishing a Connection

The server side of the TCP connection module is [Python](#) based and was realized by using the built-in [socket library](#). By default, the server can be reached using the **port 38152**. The message is encoded with the **UTF-8** encoding standard. Messages are separated by the **End of Text** character 0x03 (“\x03” in Python). Use the following example script to connect to your robot and print all incoming answers. The meaning of the incoming messages are explained further down in the [Receiving Commands](#) section.

```
import socket
import time

# define the robot ip address
ip_address = "10.0.0.203"

# create the tcp socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    # connect to the server
    client_socket.connect((ip_address, 38152))

    # prepare message buffer
    message_buffer = ""
    while True:
        # read data and add it to message buffer
        try:
            message_buffer += client_socket.recv(1024).decode("UTF-8")
        except socket.timeout:
            time.sleep(0.1)
```

```
        continue

# check for complete messages in buffer
if "\x03" in message_buffer:
    # extract complete messages
    message_split = message_buffer.split("\x03")
    for message in message_split[:-1]:
        print(message)
    # incomplete message (if any) remains in buffer
    message_buffer = message_split[-1]
```

7.2.2 Sending Commands

Before sending commands to the robot, make sure that you understood the principles explained in the section [Connections, Authentication, and Active Control](#). The following example sends the initial **token_login** and **request_active_control** commands.

```
import json
import socket
import time

# define the robot ip address, access token, and session identifier
ip_address = "10.0.0.203"
token = "MY_ACCESS_TOKEN"
session_id = "MY_SESSION_ID"

# create the tcp socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    # connect to the server
    client_socket.connect((ip_address, 38152))

    # use the authentication token to log in (needed if authentication is not disabled)
    request = {"action": "token_login", "bridge": "core",
               "arguments": {"token": token, "session_id": session_id}}
    message = json.dumps(request) + "\x03"
    client_socket.sendall(message.encode("UTF-8"))

    time.sleep(0.5)

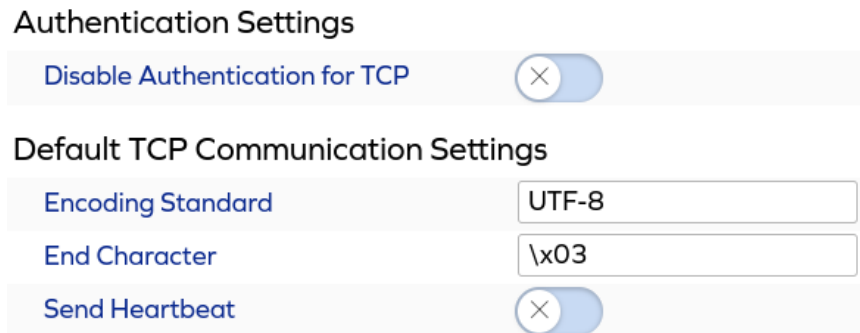
    # make sure to acquire active control, which makes the TCP connection the active
    # control station of the robot (only one control station can be active at any time)
    request = {"action": "request_active_control", "bridge": "core",
               "arguments": {}}
    message = json.dumps(request) + "\x03"
    client_socket.sendall(message.encode("UTF-8"))

    time.sleep(0.5)

# add more robot control commands here
```

7.2.3 Settings

The examples above use the following default TCP settings, which can be modified in the [Settings Menu](#) of the Desktop Interface. Here you can permanently disable the authentication for TCP, and change the encoding and end character for TCP messages. Also, you may enable a heartbeat for continuously checking whether the connection is still alive.



The screenshot shows the 'Authentication Settings' section with a toggle for 'Disable Authentication for TCP' set to 'on'. Below this is the 'Default TCP Communication Settings' section, which includes three fields: 'Encoding Standard' set to 'UTF-8', 'End Character' set to '\x03', and 'Send Heartbeat' set to 'on'.

Authentication Settings	
Disable Authentication for TCP	☒

Default TCP Communication Settings	
Encoding Standard	UTF-8
End Character	\x03
Send Heartbeat	☒

Figure 7.2.3: TCP Module Settings

7.2.3.1 Encoding Standard

Here you may define the encoding of the messages sent via the TCP connection. The usual [Python encoding standards](#) are accepted. Regularly used standards are UTF-8 which is the default or ASCII.

7.2.3.2 End Character

Here you may change the end character of the messages sent via the TCP connection. The end character signals the end of the message. The end character is automatically cut from the message, when a message is received.

7.2.3.3 Send Heartbeat

Here you may choose whether the TCP server automatically sends a heartbeat. A heartbeat will be sent every 50 ms and consists of an empty message (the end character is sent).

7.2.4 Message Protocol - Receiving Commands

All commands that are sent and received are [JSON](#) strings. As stated above, messages are encoded with the **UTF-8** encoding standard and are separated by the **End of Text** character 0x03 (“\x03” in Python). A received command is a JSON encoded dictionary that contains a key and a value. The key defines the type of the message. The value contains the information of the message. You can find a code example that receives these messages from the robot in the [Establishing a Connection](#) section.

Please note that this list is not complete, but contains the most important commands.

7.2.4.1 Status Info

The status of the robot is sent whenever it changes.

Key	Value	Type
status_info	The current robot status according to the following list: ● 0 → Unpowered ● 1 → Powering Up ● 2 → Loading ● 3 → Ready ● 4 → Processing ● 5 → Calibrating ● 6 → Running ● 7 → Autonomous ● 8 → Paused ● 9 → Proceed ● 10 → Protective Stop ● 11 → Safeguard ● 12 → Position Limit ● 13 → Hand-Guided Control ● 14 → Collision ● 15 → Emergency Stop ● 16 → Normal Stop ● 17 → Error ● 18 → Powering Down ● 19 → Stopping	Integer

```
{"status_info": 0}
```

7.2.4.2 Action Return Value

After executing an action, the return value is sent containing the command of the action. If the action does not have a return value, it is sent as null. Success is only true if there was no error during the execution of the function. Finally, the sequence number is an optional parameter that is only added if the user specified a sequence when they sent the command.

Key	Value	Type
action_return_value	The name of the returning action, the return value and whether the action was successful. If an error occurred, the action was not successful.	Dictionary

```
{"action_return_value": {"action": "release", "value": null,
                        "sequence": 100, "success": true}}
```

7.2.4.3 Error

If an error occurs, an error message is sent containing more detailed information.

Key	Value	Type
error	The error information.	Dictionary

```
{"error": {"text": "The argument 'variable_value' is missing for the
              'set_state_variable' action.",
          "title": "Minor Error",
          "color": 3,
          "code": "e0010035",
          "time": 1554907874.0436678}}
```

7.2.4.4 Message History

List of past messages sent to the user interface.

Key	Value	Type
message_history	The messages contain text, title, code and time stamps.	List

```
{"message_history": [{"text": "The specified bridge does not exist.",
                          "title": "Minor Error", "color": 3,
                          "code": "e0010019",
                          "time": 1554897027.5446029}]}}
```

7.2.4.5 State Group Define

Defines positional data of the robot. A state group contains an ID that specifies what the contained data is about. The definitions can be used to identify the state group data.

Key	Value	Type
state_group_define	State group definition.	Dictionary

```
{"state_group_define": {"id": "pose",
                        "label": "tcp_pose",
                        "members": [{"id": "x", "label": "x"},
                                   {"id": "y", "label": "y"},
                                   {"id": "z", "label": "z"},
                                   {"id": "roll", "label": "roll"},
                                   {"id": "pitch", "label": "pitch"},
                                   {"id": "yaw", "label": "yaw"}],
                        "cols": [{"id": "value", "unit": "mm, deg"}]}}
```

7.2.4.6 State Group Data

State group data contains TCP or motor data. Use “request_data” to receive this data.

Key	Value	Type
state_group_data	The value contains either the TCP of the robot (id = “pose”), the motor data (id = “robot”), or the safety flags (id = “safety”).	Dictionary

```
{"state_group_data": {"id": "pose",
                     "data": {"x": {"value": "0.0"},
                              "y": {"value": "0.0"},
                              "z": {"value": "1378.5"},
                              "roll": {"value": "0.0"},
                              "pitch": {"value": "90.0"},
                              "yaw": {"value": "180.0"}}}}}
```

7.2.4.7 Open Dialog

When a dialog in the UI opens, the “open_dialog” command is received. The dialog can then be answered using the “answer_dialog” action.

Key	Value	Type
open_dialog	The dialog takes the following values: • id → unique dialog ID (used to answer that dialog) • kind ○ 0 → confirm dialog (buttons to confirm/reject) ○ 1 → input dialog (text input) ○ 2 → file dialog (file upload) ○ 3 → dropdown dialog (requires list of options) ○ 4 → choice dialog (requires list of options) • default → return value if it does not receive an answer • color ○ 0 → information ○ 1 → success ○ 2 → warning ○ 3 → error • title → title to display • text → text to display • options → list of options for dropdown/choice dialogs	Dictionary

```
{"open_dialog": {"id": "5cee85a8-9ab2-4de8-b1ea-52ded8cc68c0",  
  "kind": 0, "default": true, "color": 2,  
  "title": "Movement Warning",  
  "text": "In order to Calibrate, the robot will move.  
  Do you want to proceed?"}}
```

7.2.4.8 Script Send

Receive a message that was sent by the script.

Key	Value	Type
script_send	Value that was sent by the script	Various

```
{"script_send": 123}
```

7.2.5 Message Protocol – Sending Commands

All commands that are sent and received are [JSON](#) strings. As stated above, messages are encoded with the **UTF-8** encoding standard and are separated by the **End of Text** character 0x03 (“\x03” in Python). A sent command must contain a JSON encoded dictionary with three mandatory (and one optional) keys. You can find a code example that sends messages to the robot in the [Sending Commands](#) section.

action	defines which command to run	str
bridge	define the module that contains the command, usually “core”	str
arguments	define the command arguments, read detailed descriptions on arguments below	dict
sequence	<optional> command identifier that is returned in the action_result_value at the end	int

Please note that this list is not complete, but contains the most important commands.

7.2.5.1 Token Login

The first command to send must always be a login call to use the active connection. Please follow the remote login procedure described in the [Secure Authentication](#) section to get a valid access token.

Action	token_login	
Bridge	core	
Argument Keys	token	session_id
Argument Value	the personalized login token provided by the login command	the unique session ID provided by the login command
Argument Type	String	String

```
{"action": "token_login",  
  "bridge": "core",  
  "arguments": {"token": "MY_ACCESS_TOKEN",  
                "session_id": "MY_SESSION_ID"}}
```

7.2.5.2 Request Active Control

Acquiring active control, such that the TCP connection becomes the active control station of the robot. Note that this sends a request to the active control station that can be accepted or rejected.

Action	request_active_control
Bridge	core

```
{"action": "request_active_control",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.3 Release Active Control

Giving away active control such that the TCP connection is no longer the active control station of the robot.

Action	release_active_control
Bridge	core

```
{"action": "release_active_control",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.4 Request Data

This command can be used as a heartbeat. Receives different kinds of data.

Action	request_data
Bridge	core

```
{"action": "request_data",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.5 Power Up

Make the robot power up its motors.

Action	power_up
Bridge	core
Argument Keys	tool_id
Argument Value	ID of a customized tool , or 0 to auto-detect the tool
Argument Type	Integer or String

```
{"action": "power_up",  
  "bridge": "core",  
  "arguments": {"tool_id": 0}}
```

7.2.5.6 Power Down

Finalize the connection, make the robot power down its motors.

Action	power_down
Bridge	core

```
{"action": "power_down",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.7 Pause

Pause all running movement functions and scripts.

Action	pause
Bridge	core

```
{"action": "pause",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.8 Resume

Resume the robot if it is Paused.

Action	resume
Bridge	core

```
{"action": "resume",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.9 Enabling Hand-Guided Control (HGC)

Release robot joints so that they are in gravity compensation mode and can be moved by hand. The status needs to be Ready and will switch to Hand-Guided Control.

Action	release
Bridge	core
Argument Keys	joint_ids
Argument Value	<optional> The list of joint IDs for which to enable hand-guided control. Enable hand-guided control for all joints by default.
Argument Type	List

```
{"action": "release",  
  "bridge": "core",  
  "arguments": {"joint_ids": ["1", "2", "5", "6"]}}
```

7.2.5.10 Disabling Hand-Guided Control (HGC)

Hold robot joints so that they are in position control mode and can no longer be moved by hand. The status needs to be Hand-Guided Control and will switch to Ready.

Action	hold
Bridge	core
Argument Keys	joint_ids
Argument Value	<optional> The list of joint IDs to hold. Hold all joints by default.
Argument Type	List

```
{"action": "hold",  
  "bridge": "core",  
  "arguments": {"joint_ids": ["1", "2", "5", "6"]}}
```

7.2.5.11 Stop

Interrupt all running movement functions and scripts.

Action	stop
Bridge	core

```
{"action": "stop",  
  "bridge": "core",  
  "arguments": {}}
```

7.2.5.12 Change Project

Change the current database project to the project with a specific ID. The status needs to be Ready.

Action	change_project
Bridge	core
Argument Keys	project_id
Argument Value	ID of the project to change to. The default project has ID 1.
Argument Type	Integer

```
{"action": "change_project",  
  "bridge": "core",  
  "arguments": {"project_id": 1}}
```

7.2.5.13 Execute Script

Run an application given by its name or ID. The status needs to be Ready and will switch to Running. While a script is running the robot can be paused. Once the script is finished, the status will switch back to Ready.

Action	execute_script	
Bridge	core	
Argument Keys	script_name	script_id
Argument Value	The name of the script to run.	The ID of the script to run.
Argument Type	String	Integer

```
{"action": "execute_script",  
  "bridge": "core",  
  "arguments": {"script_id": 3}}  
  
{"action": "execute_script",  
  "bridge": "core",  
  "arguments": {"script_name": "my_application"}}
```

7.2.5.14 Test Script

Run the Python script of an application. The status needs to be Ready and will switch to Running. While a script is running the robot can be paused. Once the script is finished, the status will switch back to Ready.

Action	test_script		
Bridge	core		
Argument Keys	script_code	script_type	script_id
Argument Value	The Python code of the application to test.	The type of the script: • “main” • “parallel” • “background”	The id of the script to test. If the script does not exist in the database, the id is 0.
Argument Type	String	String	Integer

```
{"action": "test_script",  
  "bridge": "core",  
  "arguments": {"script_code": "print('Hello World')",  
               "script_type": "main",  
               "script_id": 0}}
```

7.2.5.15 Script Receive

Use this function to send a message from your connection channel to the application script. The message can be received with the **receive_message** function of the script.

Action	script_receive
Bridge	core
Argument Keys	message
Argument Value	Message that will be received by the script.
Argument Type	Various

```
{"action": "script_receive",  
  "bridge": "core",  
  "arguments": {"message": 123}}
```

7.2.5.16 Move Pose

Move the robot to pose with a certain id or a certain name. The status needs to be Ready and will switch to Running. While the robot is moving, its motion can be paused. Once the movement is finished, the status will switch back to Ready.

Action	move_pose
Bridge	core
Argument Keys	pose_id
Argument Value	ID or name of the pose to move towards.
Argument Type	Integer or String

```
{"action": "move_pose",  
  "bridge": "core",  
  "arguments": {"pose_id": 2}}
```

```
{"action": "move_pose",  
  "bridge": "core",  
  "arguments": {"pose_id": "start_pose"}}
```

7.2.5.17 Play Path

Play the path with a certain id or a certain name. The status needs to be Ready and will switch to Running. While the robot is moving, its motion can be paused. Once the movement is finished, the status will switch back to Ready.

Action	play_path
Bridge	core
Argument Keys	path_id
Argument Value	ID or name of the path to execute.
Argument Type	Integer or String

```
{"action": "play_path",  
  "bridge": "core",  
  "arguments": {"path_id": 2}}
```

```
{"action": "play_path",  
  "bridge": "core",  
  "arguments": {"path_id": "p1"}}
```

7.2.5.18 Toggle Tool

Toggle the tool function, which is usually picking and placing. The status needs to be Ready and will switch to Running. Once the function is finished, the status will switch back to Ready.

Action	toggle_tool
Bridge	core
Argument Keys	pick
Argument Value	true to pick an object, false to place an object. This is defined by the picking behavior .
Argument Type	Boolean

```
{"action": "toggle_tool",  
  "bridge": "core",  
  "arguments": {"pick": false}}
```

7.2.5.19 Write Digital & Analog Outputs

Writing one or many outputs (**digital** or **analog**) of either the **main** controller or the **tool** controller (signal lines fed through from main controller to tool). For each of these combinations, a different function (with same arguments) is used. A list of all I/Os can be found in the [I/O Configuration Menu](#) or the [I/O Menu](#).

Action	write_digital_main_output write_analog_main_output write_digital_tool_output write_analog_tool_io	
Bridge	core	
Argument Keys	identifiers	values
Argument Value	There are many ways to identify a single I/O, either by • number (Integer) • custom name (String) If writing multiple I/Os simultaneously, you may choose to combine numbers or custom names in a List.	Depending on which ‘identifiers’ type you chose, values can be • true/false (Boolean) for digital I/Os • numbers (Float) for analog I/Os or a List of these values if writing multiple I/Os simultaneously.
Argument Type	Integer / String / List	Boolean / Float / List

```
{
  "action": "write_digital_tool_io",
  "bridge": "core",
  "arguments": {
    "identifiers": 3,
    "values": true
  }
}
```

This example writes the digital tool output 3 and sets it to true.

```
{
  "action": "write_analog_tool_io",
  "bridge": "core",
  "arguments": {
    "identifiers": "my_analog_output",
    "values": 9.5
  }
}
```

This example writes the analog tool output which was given a custom name “my_analog_output” and sets it to 9.5V.

```
{
  "action": "write_digital_main_io",
  "bridge": "core",
  "arguments": {
    "identifiers": [1, 2],
    "values": [true, false]
  }
}
```

This example writes the digital main outputs 1 and 2 and sets 1 to true and 2 to false.

7.2.5.20 Answer Dialog

Answer an open dialog window. The dialog answer depends on the type of dialog.

Action	answer_dialog	
Bridge	core	
Argument Keys	dialog_id	dialog_answer
Argument Value	The unique identifier of the dialog to send the answer to. The ID is part of the incoming open_dialog response.	The answer to send to that dialog.
Argument Type	String	Boolean / Integer / String

```
{"action": "answer_dialog",  
  "bridge": "core",  
  "arguments": {"dialog_id": "5cee85a8-9ab2-4de8-b1ea-52ded8cc68c0",  
                "dialog_answer": true}}
```

7.2.5.21 Set State Variable

Set the value of a state variable “variable_name” to “variable_value”. The variable has to exist already in the script with id “script_id”. And the script needs to be saved in the database. This command can be executed while a script is running.

Action	set_state_variable		
Bridge	core		
Argument Keys	script_id	variable_name	variable_value
Argument Value	ID of the script containing the state variable to modify.	Name of the state variable to modify.	New value.
Argument Type	Integer	String	Various

```
{"action": "set_state_variable",  
  "bridge": "core",  
  "arguments": {"script_id": 2,  
                "variable_name": "integer_1",  
                "variable_value": 42}}
```

7.2.5.22 Set Global Variable

Set the value of a global variable “name” to “value”. Global variables are saved beyond the runtime of a script and are also saved even when the robot is restarted.

Action	set_global_variable	
Bridge	core	
Argument Keys	name	value
Argument Value	Name of the global variable	Value of the global variable
Argument Type	String	Various

```
{"action": "set_global_variable",  
  "bridge": "core",  
  "arguments": {"name": "my_variable", "value": 3}}
```

7.2.5.23 Get Global Variable

Get the value of a global variable “name”. Global variables are saved beyond the runtime of a script and are also saved even when the robot is restarted.

Action	get_global_variable	
Bridge	core	
Argument Keys	name	
Argument Value	Name of the global variable	
Argument Type	String	

```
{"action": "get_global_variable",  
  "bridge": "core",  
  "arguments": {"name": "my_variable"}}
```

7.2.5.24 Set Shared Variable

Set the value of a shared variable “name” to “value”. Shared variables are only available during the run time of a script. They are deleted after the script has finished or the robot has stopped.

Action	set_shared_variable	
Bridge	core	
Argument Keys	name	value
Argument Value	Name of the shared variable	Value of the shared variable
Argument Type	String	Various

```
{"action": "set_shared_variable",  
  "bridge": "core",  
  "arguments": {"name": "my_variable", "value": 3}}
```

7.2.5.25 Get Shared Variable

Get the value of a shared variable “name”. Shared variables are only available during the run time of a script. They are deleted after the script has finished or the robot has stopped.

Action	get_shared_variable	
Bridge	core	
Argument Keys	name	
Argument Value	Name of the shared variable	
Argument Type	String	

```
{"action": "get_shared_variable",  
  "bridge": "core",  
  "arguments": {"name": "my_variable"}}
```

7.3 REST Connection Module

7.3.1 Establishing a Connection

The REST connection module provides the resource **https://<IP-ADDRESS>/rest/gateway** to receive commands from connecting clients. Note that there is only one connection channel. If multiple users connect via REST, they share this channel.

Use the following example script to take control of the robot using the REST connection, and then to start sending commands to the robot. Note that before sending commands you need to be logged in, and you need to have active control. It is therefore also recommended to release the control and logout once the connection is no longer used.

```
import json
import requests

IP_ADDRESS = "10.0.0.203"
TOKEN = "MY_ACCESS_TOKEN"
SESSION_ID = "MY_SESSION_ID"

REST_URL = f"https://{IP_ADDRESS}/rest/gateway"
HEADERS = {"Accept": "application/json", "Content-Type": "application/json"}

def request(method, url, headers, data, timeout=None):
    try:
        response = method(url=url, headers=headers, data=data, timeout=timeout,
                           verify=False)
        response_code = response.status_code

        # this is equal to 'response.status_code < 400' (errors have codes 400+)
        if response.ok:
            # try to parse json payload
            try:
                response_payload = response.json()
            except json.decoder.JSONDecodeError:
                # usually this happens because the response payload is empty
                # this happens for example when a request returns "", 200
                response_payload = dict()
                print("response payload json was empty")
        else:
            print("response.ok returned False")

    except requests.Timeout:
        response_payload = dict()
        response_code = 408 # timeout
    except Exception as e:
        # usually connection problems...
        print(f"rest client could not connect to specified endpoint: {e}")
        response_payload = dict()
        response_code = 404 # not found
```

```
return response_code, response_payload

def post_request(data, timeout=None):
    return request(method=requests.post, url=REST_URL, headers=HEADERS,
data=data, timeout=timeout)

# login
data = {"bridge": "core", "action": "token_login", "arguments": {"token":
"<TOKEN>", "session_id": "<SESSION_ID>"}}
response = post_request(data, timeout=10.0)
# acquire the active control before start sending commands to the robot
data = {"bridge": "core", "action": "request_active_control", "arguments": {}}
response = post_request(data, timeout=1.0)

# add your commands here

# when you're done using this connection, give away the active control
data = {"bridge": "core", "action": "release_active_control", "arguments": {}}
response = post_request(data, timeout=1.0)
# when you're done working with this connection, you should logout
data = {"bridge": "core", "action": "logout", "arguments": {}}
response = post_request(data, timeout=1.0)
```

7.3.2 Sending Commands

In addition to the commands mentioned in the example above, many more commands are available. Please refer to the list of commands listed in the [TCP connection module](#), since the same commands are also used for the REST connection.

In addition to sending a “bridge”, “action”, “arguments” and an optional “sequence” parameter, you may also send an optional “timeout” parameter. If you send a timeout, the REST request will not result in the default response if successfully sent (200, {}), but instead the command will be blocking and you will get the response as (200, action_return_value), with the [Action Return Value](#) containing the information which API function was called, whether the called function was successful, and the actual return value of the called function.

7.3.3 Settings

In the [Settings Menu](#) of the Desktop Interface you can find a setting for disabling the authentication for the REST connection module. Disabling the authentication removes the need to login and logout.

Authentication Settings

Disable Authentication for REST



Figure 7.3.3: REST Module Settings

7.4 ROS Handler Module

The ROS handler module allows you to integrate your robot into a ROS 2 environment (Robot Operating System) and use ROS 2 commands to communicate with AX Portal. The instructions in this chapter are compatible with ROS Jazzy and [CycloneDDS](#) as [ROS middleware \(RMW\)](#).



WARNING

The ROS handler provides access to certain low-level functions. For instance, it is possible to send direct control commands to the robot's motors, which will be executed immediately. In the GUI, some functions may require additional checks or dialog confirmations, which may not be available here. Therefore, be especially cautious when controlling your robot directly via ROS.

Note:

- This is the first AX Portal release with ROS 2 and additional features may be expected in the future.
- As of now, the ROS handler is not available with the AX Portal simulator, but only with actual robots.

7.4.1 Installation

To communicate with AX Portal's ROS handler, you require a ROS installation on your machine(s). To set this up, the [official ROS documentation](#) provides different options, i.e. deb package, docker image, source installation, etc. In this example, we will use the official ROS docker image to communicate with the robot. The following docker compose file creates a simple ROS environment on your computer:

```
None
services:
  ros2_development:
    image: osrf/ros:jazzy-desktop
    container_name: "ros2_development"
    command: /bin/bash -c
      "apt update &&
      apt install -y ros-jazzy-rmw-cyclonedds-cpp &&
      sleep infinity"
    environment:
      - RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
      - CYCLONEDDS_URI=/ros/cyclonedds_config.xml

  network_mode: "host"

volumes:
  - "./cyclonedds_config.xml:/ros/cyclonedds_config.xml"
```

Note that we are installing `ros-jazzy-rmw-cyclonedds-cpp` on startup and set `RMW_IMPLEMENTATION` to `rmw_cyclonedds_cpp` to tell ROS to use CycloneDDS, the same RMW that is on the robot. Additionally, we share a `cyclonedds_config.xml` and set the `CYCLONEDDS_URI` to its location to have a custom CycloneDDS setup.

7.4.2 Configuration

ROS communication between the robot and your computer requires adjustments on both machines. On your laptop, you'll need to set a new environment variable and provide a network configuration file. On the robot, you don't have direct access to these elements, but you can edit them in the user interface via the ROS section of the settings menu:

The screenshot displays the 'ROS Function Settings' window. It contains the following elements:

- Network Interface:** A blue dropdown menu currently showing 'Wifi'.
- Multicast:** A toggle switch with a blue circle and an 'X' icon, currently in the 'off' position.
- IP Addresses of ROS 2 Peers:** A list of text input fields. The first two contain '192.168.0.1' and '192.168.0.2'. Each field has a trash icon to its right. A plus icon is at the bottom right of the list to add more addresses.
- ROS Domain ID:** A blue dropdown menu currently showing '15'.

Figure 7.4.2: ROS Handler Settings

In the coming sections, you will learn the meaning of each of these settings. You will also learn what adjustments are needed analogously on your computer's side.

7.4.3 Network Discovery with Cyclone DDS

We have referred to Cyclone DDS a lot earlier. What is it about? Here's what you need to know as an AX Portal user:

When Cyclone DDS is selected as ROS middleware, it takes over the task of distributing ROS-related data packages through your network. It is responsible to establish a network connection between your devices, so ROS 2 can send data through those connections.

7.4.3.1 Network Interface

First, Cyclone DDS needs to know which network interface will be used by ROS. The default is ethernet.

Use the "Network Interface" setting under "ROS Function Settings" to select your desired interface.

Note: When AX Portal starts up the ROS handler, Cyclone DDS scans the network interface for IP addresses of peer machines. If it cannot access this interface, the ROS handler will fail to load. Therefore, if you've set the network interface to ethernet, make sure the cable is plugged in *before you start up the robot*.

7.4.3.2 Discovery Methods

Cyclone DDS provides two methods for discovering other devices in the network: multicast and peer-to-peer. For multicast, it will send every network package to every reachable client in the network range. The advantage is simplicity: You won't need to bother defining which computers are allowed to talk to each other. In this setup, the `ROS_DOMAIN_ID` environment variable (see below) is still available to prevent two robots from mixing up messages.

The drawback is network overhead: All packages will be processed by each network client. Even setting different `ROS_DOMAIN_ID`s cannot prevent this. Even if a computer doesn't have ROS installed: Since Cyclone DDS is UDP based, all computers in the network will still receive and process those UDP packages. That's usually not a problem for smaller networks and small amounts of data. But if a robot would e.g. send a camera stream to every client in a large network, this can noticeably slow down the entire network. In such a scenario, it's safer to explicitly define a number of IP addresses in the Cyclone DDS configuration, which are allowed to communicate with each other through peer-to-peer connections.

In the coming two sub-sections, you will learn how to set up your network for multicasting or peer-to-peer communication, respectively. Hint: On most Linux distros, the command `ip addr show` comes in handy to find out your IP addresses and corresponding network interface names.

7.4.3.2.1 Peer to Peer

During the installation instructions above, in the docker compose file, we mounted a Cyclone DDS configuration file. Here's an example Cyclone DDS config, to set up your machine for peer-to-peer communication:

```
None
<?xml version='1.0' encoding='utf-8'?>
<CycloneDDS                                     xmlns="https://cdds.io/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://cdds.io/config
https://raw.githubusercontent.com/eclipse-cyclonedds/cyclonedds/master/etc/cyclonedds.xsd">
  <Domain id="any">
    <General>
      <Interfaces>
        <NetworkInterface name="INTERFACE_NAME" multicast="false" />
      </Interfaces>
      <AllowMulticast>false</AllowMulticast>
      <EnableMulticastLoopback>false</EnableMulticastLoopback>
    </General>
    <Discovery>
      <MaxAutoParticipantIndex>119</MaxAutoParticipantIndex>
      <ParticipantIndex>auto</ParticipantIndex>
      <Peers>
        <Peer address="192.168.1.1" />
        <Peer address="192.168.1.2" />
      </Peers>
    </Discovery>
  </Domain>
</CycloneDDS>
```

```
        </Peers>
    </Discovery>
</Domain>
</CycloneDDS>
```

First, replace `INTERFACE_NAME` with the actual network interface name on your computer that is used for ROS communication. Examples: `enp1s0`, `wlp2s0`, etc.

Next, adjust the list under “Peers”, so that it reflects the IP addresses of the machines in your network, which you want to connect with each other via ROS.

Analogously to the file above, there is a similar file stored on your robot. You cannot access it directly, but you can edit its content through AX Portal via the “ROS Function Settings” section of the settings menu. There, the “IP Adresses of ROS 2 Peers” setting provides a list of text fields, which you can fill with IP addresses.

Beware:

- A restart of the robot is required for the change to take effect.
- If any IP address in the list is malformed, AX Portal will raise an error and the new list won't be saved.
- The list needs to contain the robot's own IP address. For now, it is not read automatically.
- When an IP address is removed from the list and the new list is saved, the IP will also be removed from the Cyclone DDS configuration.
- Due to firewall settings it may be necessary that all ROS participants are in the same network subrange.

7.4.3.2.2 Multicast

Here's another example Cyclone DDS config, to set up your machine for multicasting:

```
None
<?xml version="1.0" encoding="utf-8"?>
<CycloneDDS
  xmlns="https://cdds.io/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://cdds.io/config
https://raw.githubusercontent.com/eclipse-cyclonedds/cyclonedds/master/etc/cyclonedds.xsd"
>
  <Domain Id="any">
    <General>
      <Interfaces>
        <NetworkInterface name="INTERFACE_NAME" multicast="true" />
      </Interfaces>
      <AllowMulticast>true</AllowMulticast>
```

```
<EnableMulticastLoopback>true</EnableMulticastLoopback>
</General>
</Domain>
</CycloneDDS>
```

Again, replace `INTERFACE_NAME` with the actual network interface name on your computer that is used for ROS communication. Examples: `enp1s0`, `wlp2s0`, etc.

To switch your robot to multicast, simply toggle the “Multicast” setting button. Again, a restart of the robot is required for the change to take effect.

7.4.4 Establishing a Connection

Now to start the ROS environment on your computer, navigate to your docker compose file and run:

```
None
docker compose up -d
```

Verify that the docker container is running with

```
None
docker ps -a
```

which prints something similar to the following output:

```
None
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
132bdd6fa4de   osrf/ros:jazzy-desktop             "/ros_entrypoint.sh ..." 2 seconds ago Up 1 second
ros2_development
```

Enter the container in interactive mode with

```
None
docker exec -it ros2_development bash
```

Do not forget to source the ROS setup file every time you enter the container:

```
None
source /opt/ros/jazzy/setup.bash
```

To be able to have multiple robots in the same network, we use a different `ROS_DOMAIN_ID` on every robot. Go to the “ROS Function Settings” in AX Portal and set the desired ROS Domain ID. The allowed range is 6-41. Also for this, note that the robot has to be restarted after this action. Now, go back to the terminal where your ROS docker container is opened and set the same value for `ROS_DOMAIN_ID`:

```
None
export ROS_DOMAIN_ID=SELECTED_NUMBER
```

You should now be able to see an AX Portal ROS node called ‘robot’ with:

```
None
ros2 node list
```

7.4.5 Access Robot Data Through ROS

Once you see the node, you can view the list of available topics by running:

```
None
ros2 topic list
```

Through ROS 2 topics, you can stream live data from your robot. E.g. to subscribe to joint positions, velocities etc., run:

```
None
ros2 topic echo /joint_states
```

7.4.6 ROS 2 Functionalities in Scripts

You can import `rcipy` directly in scripts. This allows you to create ROS 2 objects like publishers, subscribers, etc.

The ROS handler creates its own ROS node. Use the `ros_handler.ros_node()` function to access it. Attention: Do not run `rcipy.spin` on this node, since this already happens within the ROS handler.

7.4.7 Call an Action

The AX Portal ROS node creates an action server called ‘/robot/execute_function’ for executing bridge functions. The action has a custom definition `ExecuteFunction` action from package `generic_messages` (https://github.com/Epson-Robots/generic_msgs) that has to be installed for acquiring the action definition. Clone the package to your workspace and build it with `colcon`:

None

```
mkdir -p /colcon_ws/src
cd /colcon_ws/src
git clone https://github.com/Epson-Robots/generic_msgs
cd ..
colcon build --packages-up-to generic_messages
source install/setup.bash
```

In order to send ROS2 actions to the robot, your client needs to acquire active control. To do so, run the command below from your terminal and confirm the dialog in the AX Portal GUI:

None

```
ros2 action send_goal --feedback /execute_function
generic_msgs/action/ExecuteFunction "{action: request_active_control, bridge:
core, arguments: '{}'}"
```

Now you can run a bridge function from a terminal with the following commands:

None

```
ros2 action send_goal --feedback /execute_function
generic_msgs/action/ExecuteFunction "{action: read_actuator_position, bridge:
core, arguments: '{\"actuator_ids\": 6}'}"

ros2 action send_goal /execute_function generic_msgs/action/ExecuteFunction
"{action: move_joints, bridge: core, arguments: '{\"joints\": 6,
\"joint_position\": 100.0}'}"
```

7.4.8 Authentication Settings

The ROS handler currently only works without authentication. In the [Settings Menu](#) of the Desktop Interface you need to manually disable the authentication for this specific connection module.

Authentication Settings

Disable Authentication for ROS



Figure 7.4.8: ROS Authentication Settings

7.5 Modbus Connection Module

7.5.1 Overview

The protocol used for the Modbus module is [TCP Modbus](#). On the server side the module is Python based and was realized by using the [PyModbus library](#). For details also checkout the [PyModbus Documentation](#). The module is separated into reading registers (Analog Input Registers, 30000-39999) and writing registers (Analog Output Holding Registers, 40000-49999). Any TCP Modbus client can be used to communicate with the server. The port that is to be used is **5020**. In the final section of this chapter several [Python Code Examples](#) are given, some of which explain how to establish a connection to the server using a Python script.

7.5.2 Settings

The Modbus connection currently only works without authentication. In the [Settings Menu](#) of the Desktop Interface you need to manually disable the authentication for this specific connection module.

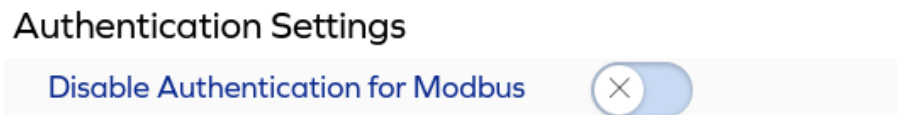


Figure 7.5.2: Modbus Module Settings

7.5.3 Writing Registers (Simple Description)

The Modbus add-on reacts to any change in information for both the command register and the property register. Clearing the data of the writing registers is within the responsibility of the user.

7.5.3.1 Command Registers

Write a single bit to the Command Register (30000) to execute a command. Some commands require an argument in the Command Data Register (30001). If you cannot write both registers at the same time, make sure to write the Command Data Register first.

Absolute Address: 30000 - 30001 Relative Address: 0x000 - 0x001		
Bit	Command Register (30000)	Data (30001)
0	Power Up	Tool ID
1	Power Down	
2	Stop	
3	Pause	
4	Resume	
5	Calibrate	Script ID
6	Enable Hand-Guided Control (HGC)	
7	Disable Hand-Guided Control (HGC)	
8	Change Project	Project ID
9	Start Script	Script ID
10	Move To Pose	Pose ID
11	Play Path	Path ID
12	Tool Place	
13	Tool Pick	
14 ... 15	<i>reserved</i>	

7.5.3.2 Property Registers

Write a single bit to the Property Register (30002) to execute a command. Some commands require an argument in the Property Data Register (30003). If you cannot write both registers at the same time, make sure to write the Property Data Register first.

Absolute Address: 30002 - 30003 Relative Address: 0x002 - 0x003		
Bit	Property Register (30002)	Data (30003)
0	Write Digital Outputs	IO State
1	Set Script Events	Event ID
2	Acknowledge Error	
3	Read Position	
4	Request Active Control	
5	Release Active Control	
6	Switch to Manual Mode	
7	Switch to Automatic Mode	
8	Reset Safety	
9	Proceed Motion	
10 ... 15	<i>reserved</i>	

7.5.3.3 User Writing Registers

Absolute Address: 30016 Relative Address: 0x010		...	Absolute Address: 30255 Relative Address: 0x0FF	
0 ... 15	Custom Data		0 ... 15	Custom Data

Use the User Writing Registers to send information to your script application. The **modbus.write_user_register** function of the script application can be used to read custom data.

7.5.4 Writing Registers (Detailed Description)

7.5.4.1 Do Nothing

Clearing the data of the writing registers is within the responsibility of the user. In order to clear either the Command Register (30000) or the Property Register (30002), simply write the integer 0.

0x000	0x001	0x002	0x003
0		0	

7.5.4.2 Power Up

Before working with the robot, it must power up its motors. The ID of the custom tool to connect with is specified by the number n. The robot will change its status from Unpowered (1, Bit 0) to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
1	n		

7.5.4.3 Power Down

When the working day is done, the robot can power down its motors. The robot can be powered down while being in any status. Once the robot is powered down, the status will change to Unpowered(1, Bit 0).

0x000	0x001	0x002	0x003
2			

7.5.4.4 Stop

Interrupt all robot movement and change the status from Running (64, Bit 6) back to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
4			

7.5.4.5 Pause

Pause the robot while it is Running (64, Bit 6). The status will switch to Paused (128, Bit 7).

0x000	0x001	0x002	0x003
8			

7.5.4.6 Resume

Resume the robot if it is Paused (128, Bit 7). The status will switch to Running (64, Bit 6).

0x000	0x001	0x002	0x003
16			

7.5.4.7 Calibrate

The calibration currently has no effect. This bit is reserved.

0x000	0x001	0x002	0x003
32			

7.5.4.8 Enable Hand-Guided Control (HGC)

Enable hand-guided control for all robot joints so that they are in gravity compensation mode and can be moved by hand. The status needs to be Ready (8, Bit 3) and will switch to Hand-Guided Control (2048, Bit 11).

0x000	0x001	0x002	0x003
64			

7.5.4.9 Disable Hand-Guided Control (HGC)

Hold all robot joints so that they are in position control mode and can no longer be moved by hand. The status needs to be Hand-Guided Control (2048, Bit 11) and will switch to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
128			

7.5.4.10 Change Project

Change the current database project to the project with the id n. The status needs to be Ready (8, Bit 3).

0x000	0x001	0x002	0x003
256	n		

7.5.4.11 Start Script

Run the Script with the ID n. The status needs to be Ready (8, Bit 3) and will switch to Running (64, Bit 6). While a script is running the robot can be paused. Once the script is finished, the status will switch back to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
512	n		

7.5.4.12 Move Pose

Move the robot to pose with ID n. The status needs to be Ready (8, Bit 3) and will switch to Running (64, Bit 6). While the robot is moving, its motion can be paused. Once the movement is finished, the status will switch back to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
1024	n		

7.5.4.13 Play Path

Play the path with ID n. The status needs to be Ready (8, Bit 3) and will switch to Running (64, Bit 6). While the robot is moving, its motion can be paused. Once the movement is finished, the status will switch back to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
2048	n		

7.5.4.14 Tool Place

Place an object using the tool (e.g. by opening the gripper). The status needs to be Ready (8, Bit 3) and will switch to Running (64, Bit 6). While the gripper is moving, its motion can be paused. Once the movement is finished, the status will switch back to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
4096			

7.5.4.15 Tool Pick

Pick an object using the tool (e.g. by closing the gripper). The status needs to be Ready (8, Bit 3) and will switch to Running (64, Bit 6). While the gripper is moving, its motion can be paused. Once the movement is finished, the status will switch back to Ready (8, Bit 3).

0x000	0x001	0x002	0x003
8192			

7.5.4.16 Write Digital Outputs

Function reserved for writing digital outputs.

0x000	0x001	0x002	0x003
		1	d

7.5.4.17 Set Script Events

If a user application is supposed to wait for an event sent by the Modbus device, the script events can be used. There are a total of 16 events. Depending on data d events are set or cleared. The **modbus.wait_for_event** script function waits until a specific event is set. If d is 0, all events are cleared. If d is 65535, all events are set. If d is 9, the first and fourth events are set, the others are cleared.

0x000	0x001	0x002	0x003
		2	d

7.5.4.18 Acknowledge Error

Errors need to be acknowledged before they are cleared. However this is only necessary to read the errors. The robot state is not influenced by the acknowledgement. If both the major and minor error codes (40004-40005) are 0, no further error needs to be acknowledged.

0x000	0x001	0x002	0x003
		4	

7.5.4.19 Read Position

Update the positional data of the robot. The current robot joint angles and robot coordinates are written into the corresponding reading registers (40256-40267), whenever the positional data is requested.

0x000	0x001	0x002	0x003
		8	

7.5.4.20 Request Active Control

Request active control in order to send commands to the robot. User Data Registers and Reading Registers can be accessed without active control. The watcher that currently has active control is notified and needs to actively release control. The corresponding reading register (40008) is set once control has been granted.

0x000	0x001	0x002	0x003
		16	

7.5.4.21 Release Active Control

Release active control so another watcher can send commands to the robot. User Data Registers and Reading Registers can be accessed without active control. The corresponding reading register (40008) will switch to 0.

0x000	0x001	0x002	0x003
		32	

7.5.4.22 Switch to Manual Mode

Switch to Manual Mode. In this mode the robot will only move while the user presses the enabled button. Note that a safety reset will be required for the robot to switch modes.

0x000	0x001	0x002	0x003
		64	

7.5.4.23 Switch to Automatic Mode

Switch to Automatic Mode. Note that a safety reset will be required for the robot to switch modes.

0x000	0x001	0x002	0x003
		128	

7.5.4.24 Reset Safety

Start/Restart Interlock Reset. For certain safety functionalities the robot requires a distinct, intentional action from the user in order to continue. To check whether such an action is required use the corresponding reading register (40009).

0x000	0x001	0x002	0x003
		256	

7.5.4.25 Start Motion Interrupt

After a safety interrupt, the robot may proceed and resume its motion. This property can be used to make the robot move again. Otherwise a Stop command can be sent to stop the running application. To check whether such an action is required use the corresponding reading register (40009).

0x000	0x001	0x002	0x003
		512	

7.5.5 Reading Registers

7.5.5.1 Status Registers

Use the Status Register to read the current robot status. The status is provided as a bit number.

7.5.5.1.1 1st Status Register

Absolute Address: 40000 Relative Address: 0x000	
0	Unpowered
1	Powering Up
2	Loading
3	Ready
4	Processing
5	Calibrating
6	Running
7	Autonomous
8	Paused
9	Proceed
10	Protective Stop
11	Safeguard
12	Position Limit
13	Hand-Guided Control (HGC)
14	Collision
15	Emergency Stop

7.5.5.1.2 2nd Status Register

Absolute Address: 40001 Relative Address: 0x001	
0	Normal Stop
1	Error
2	Powering Down
3	Stopping
4.. 15	<i>reserved</i>

7.5.5.1.3 3rd Status Register

Absolute Address: 40002 Relative Address: 0x002	
0 ... 15	<i>reserved</i>

7.5.5.1.4 4th Status Register

Absolute Address: 40003 Relative Address: 0x003	
0 ... 15	<i>reserved</i>

7.5.5.1.5 Major Error Register

Absolute Address: 40004 Relative Address: 0x004	
0 ... 15	Information regarding the error category

The Major Error Register reads the first 2 bytes of the identifier of the current error. Acknowledge the error using the properties register in order to read the next error.

For details about the meaning of the Error Register values, have a look at the [Error Register Mapping](#).

7.5.5.1.6 Minor Error Register

Absolute Address: 40005 Relative Address: 0x005	
0 ... 15	Information regarding the exact error in the error category

The Minor Error Register reads the last 2 bytes of the identifier of the current error. Acknowledge the error using the properties register in order to read the next error.

For details about the meaning of the Error Register values, have a look at the [Error Register Mapping](#).

7.5.5.1.7 Application Register

Absolute Address: 40006 Relative Address: 0x006	
0 ... 15	Current Script ID

The Application Register reads the ID of the script that is currently running.

7.5.5.1.8 Calibration Register

Absolute Address: 40007 Relative Address: 0x007	
0	Whether the robot is calibrated

The Calibration Register reads whether the robot is calibrated. 0 means the robot is not calibrated. 1 means that the robot is calibrated. Remember that the robot needs to be connected to its motor to give meaningful feedback on calibration.

7.5.5.1.9 Active Control Register

Absolute Address: 40008 Relative Address: 0x008	
0	Whether the modbus watcher has active control

The Active Control Register reads whether the modbus watcher has active control. 0 means modbus does not have active control. 1 means modbus has active control. The modbus watcher can request active control using 0x30002 0x04 Request Active Control.

7.5.5.1.10 Interrupt Register

The Interrupt Register indicates active interrupt dialog options.

Absolute Address: 40009 Relative Address: 0x009	
0	Reset Safety required
1	Proceed Motion required
2	Switch to Manual Mode required

3.. 15	<i>reserved</i>
--------	-----------------

7.5.5.1.11 Operation Mode Register

The Operation Mode Register indicates whether the robot is in Manual (1) or Automatic (0) Mode.

Absolute Address: 40010 Relative Address: 0x00A	
0	Whether robot is in Manual Mode

7.5.5.2 User Reading Registers

Absolute Address: 40016 Relative Address: 0x010		...	Absolute Address: 40255 Relative Address: 0x0FF	
0 ... 15	Custom Data		0 ... 15	Custom Data

Use the User Reading Registers to receive information from the script application. The **modbus.read_user_register** function of the application will read the data.

7.5.5.3 Position Registers (Joints 1 through 6)

Absolute Address: 40256 Relative Address: 0x100		...	Absolute Address: 40261 Relative Address: 0x105	
0 ... 15	Joint angle of joint 1		0 ... 15	Joint angle of joint 6

Read the angular value of a specific joint in tenths of degrees. If 101 is read, the angle of the joint is 10.1°. The value is only updated if the read position flag is written in the property register. If values bigger than 32768 ($=2^{15}$) are read, these correspond to negative values. Subtract 32768 to read the absolute value. E.g. a value of 32869 corresponds to the joint angle of -10.1°.

7.5.5.4 Coordinate Register (x, y, z, roll, pitch, yaw)

Absolute Address: 40262 Relative Address: 0x106		...	Absolute Address: 40267 Relative Address: 0x10B	
0 ... 15	x-coordinate of TCP		0 ... 15	yaw-coordinate of TCP

Read the coordinates or orientation of the tool center point in the order x, y, z, roll, pitch, yaw. The spatial coordinates are read in tenths of millimeters. If a coordinate reads 314, the value is 31.4 millimeters. The rotational coordinates are read in tenths of degrees. If a coordinate reads 271, the value is 27.1 (mm or °). The value is only updated if the read position flag is written in the property register. If values bigger than 32768 ($=2^{15}$) are read, these correspond to negative values. Subtract 32768 to read the absolute value. E.g. a value of 33039 corresponds to the coordinate value of -27.1 (mm or °).

7.5.6 Error Register Mapping

Error information always appears in two registers. The [Minor Error Register](#) contains the actual error number. The [Major Error Register](#) contains one of the following values, which map to different error categories as follows:

101	ADDONS
102	CONFIG
103	DATABASE
104	CONTROLLER
105	EDITOR
106	BRIDGE
107	CONNECTION
108	DIALOG
109	DISRIBUTOR
110	EVENT
111	KINEMATICS
112	SOCKET
113	CORE
114	SAFETY
115	SETTINGS
116	SIGNAL
117	STATUS
118	SCRIPT
119	THREAD

201	BLOCK
202	COGNEX
203	CONTROL
204	ROS
205	MODBUS
206	REST
207	TCP

301	AX
302	CFG
303	CMK
304	CMD
305	CTL
306	ECAT
307	LIC
308	MATH
309	MCM
310	MISC
311	OS
312	PPM
313	PYAPI
314	SIG
315	SYS
316	TLC
317	UTIL

7.5.6.1 Example

If the error registers read [113, 4], this maps to the error code **CORE_004** raised by the robot.

7.5.7 Python Code Examples

These examples are to be used on an external system connecting remotely to the robot.

7.5.7.1 Example 1

```
from pymodbus.client.sync import ModbusTcpClient

def main():
    # connect to client
    client = ModbusTcpClient("192.168.80.203", port=5020)
    client.connect()

    # read all status
    address = 0x000
    count = 4
    status = client.read_input_registers(address, count)
    print("STATUS", status.registers[0], status.registers[1],
          status.registers[2], status.registers[3])

if __name__ == "__main__":
    main()
```

Connect to the server. Read the robot status.

7.5.7.2 Example 2

```
from pymodbus.client.sync import ModbusTcpClient

def main():
    # connect to client
    client = ModbusTcpClient("192.168.80.203", port=5020)
    client.connect()

    # command to connect to motors
    address = 0x000
    values = [1 << 0]
    client.write_registers(address, values, unit=1)

if __name__ == "__main__":
    main()
```

Connect to the server. Connect the robot to its motors.

Hint: The << command is used for bit shifting.

Bit	Command	Integer
0	1 << 0	1
5	1 << 5	32

7.5.7.3 Example 3

```
from pymodbus.client.sync import ModbusTcpClient

def main():
    # connect to client
    client = ModbusTcpClient("192.168.80.203", port=5020)
    client.connect()

    # command to run script with id 5
    address = 0x000
    values = [1 << 9, 5]
    client.write_registers(address, values, unit=1)

if __name__ == "__main__":
    main()
```

Connect to the server. Run application with ID 5.

7.5.7.4 Example 4

```
from pymodbus.client.sync import ModbusTcpClient

def main():
    # connect to client
    client = ModbusTcpClient("192.168.80.203", port=5020)
    client.connect()

    # read all error
    address = 0x004
    count = 2
    error = client.read_input_registers(address, count)
    print("ERROR", error.registers[0], error.registers[1])

    # acknowledge error
    address = 0x002
    values = [1 << 2]
    client.write_registers(address, values, unit=1)

if __name__ == "__main__":
    main()
```

Connect to the server. Read the current error message. Acknowledge the error message.

8. Cognex Camera Module

8.1 Overview

The Cognex Camera Module allows you to connect a pre-configured Cognex camera to AX Portal. Once connected, you can trigger the pre-configured jobs on the camera to take and process an image. The processed data is then transmitted to AX Portal, where you can read the data in your application.

8.2 Camera Setup

To set-up a Cognex camera for compatibility with AX Portal, please follow these steps:

1. Download and install the [In-Sight Explorer](#) software by Cognex.
2. Power and connect the camera to your network. You may also connect it directly to your computer, if its network setup is set to a static IP address.
3. Launch the In-Sight Explorer, allow access if requested, and install the required items (e.g. Microsoft Visual C++, Microsoft .NET Framework).
4. After that, the Easybuilder view of the Explorer opens automatically.
5. Navigate to “System” → “Add Sensor/Device to Network”. Look for your camera and add it to the network.
6. Make sure the network settings of the camera are compatible with the [network settings](#) of the AX series robot. Also make sure that the Telnet-Port is set to 3000.
7. Select your camera and hit the “Connect” button to connect it with the In-Sight Explorer.
8. Now you are ready to create jobs (e.g. for calibrating the camera, and for recognizing objects for the robot to pick) and save them on the camera. Refer to the Cognex documentation for more details.

8.3 Camera Application

Using the Cognex camera in an AX Portal application requires

- the robot and camera to be in the same network,
- that the Telnet-Port of the camera was set to 3000,
- and that the camera is set to online.

Here's an example of how a pre-configured job could be used to detect the coordinates of the circle.

```
# modify these settings to fit to your application
camera_ip = "10.0.0.210"
job_name = "CircleDetection"
event_id = 1
keyword = "coordinates"

# connect the camera to the application
cognex.connect_to_camera(camera_ip)
# select the job to trigger
cognex.change_job(camera_ip, job_name)
# trigger the event and wait for a response
response = cognex.get_camera_message(camera_ip, event_id, keyword)
print(response)
# close the connection
cognex.disconnect_camera(camera_ip)
```