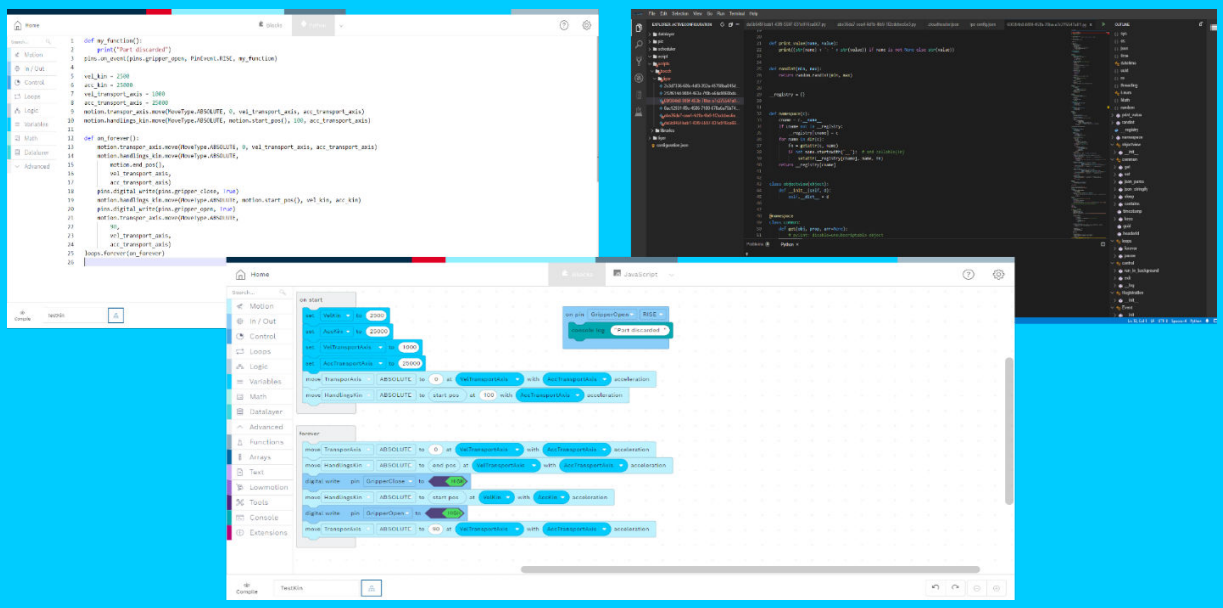
rexroth
A Bosch Company

Application Manual

IDE App

Integrated Development Environment



R911410625, Edition 05

Liability

The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.

DOK-XCORE*-IDE********-AP05-EN-P DC-AE/EPI5 (TaDo/MePe)

d6bf9650d4987a8a0a347e8873733cf8, 5, en_US

Table of contents

1 About this documentation

Editions of this documentation

Edition	Release date	Note
01	2021-04	First edition Version IDE-V-0108
02	2021-08	Revision Version IDE-V-0110 • ➦Chapter 8 Authorization on page 19 revised
03	2022-01	Revision Version IDE-V-0112 • New: ➦Chapter 10.8 ctrlX AUTOMATION Data Layer browser on page 53 • ➦Chapter 7 Installation on page 17 revised • ➦Chapter 8.1 Visual Coding IDE on page 19 revised • ➦Chapter 10.2 Motion and Data Layer integration on page 47 revised • ➦Chapter 9.3.5 Starting script on page 28, ➦Chapter 9.3.4 File handling on page 27: Path modified
04	2022-04	Revision Version IDE-V-0114 Fig. 16: Graphics updated Fig. 24: Graphics updated Fig. 21: Graphics updated
05	2022-10	Revision Version IDE-V-0116 • ➦Chapter 9.3.2 Basis and advanced extensions on page 26: Table updated • ➦Chapter Motion extension on page 33 revised • New: ➦Chapter ctrlX AUTOMATION extension on page 42 • New: ➦Chapter Motion extension on page 33

2 Important directions on use

2.1 Intended use

2.1.1 Introduction

Rexroth products are developed and manufactured to the state-of-the-art.

The products are tested prior to delivery to ensure operational safety and reliability.

⚠ WARNING	**Personal injury and damage to property due to incorrect use of products!** The products may only be used as intended. Failure to use the products as intended may cause situations resulting in property damage and personal injury.

NOTICE	**Damages resulting from unintended use** Rexroth As the manufacturer does not assume any warranty, liability or compensatory claims for damages resulting from unintended use of the products. The user alone shall bear the risks of an unintended use of the products. Before using Rexroth products, make sure that all the prerequisites for an intended use of the products are met: – Personnel that in any way, shape or form uses Rexroth products must first read and understand the relevant safety instructions and be familiar with their intended use – Leave hardware products in their original state, i.e., do not make any structural modifications. It its not permitted to decompile software products or alter source codes – Do not install damaged or defective products or commission them – It has to be ensured that the products have been installed as described in the relevant documentation

2.1.2 Areas of use and application

Products of the ctrlX series are suitable for Motion/Logic applications.

NOTICE	Products of the ctrlX series may only be used with the accessories, mounting parts, and other components specified in this documentation. Components that are not expressly mentioned must neither be attached nor connected. The same applies to cables and lines. Only to be operated with the hardware component configurations and combinations expressly specified and with the software and firmware specified in the corresponding documentations and functional descriptions.

Products of the ctrlX series are suitable for single-axis as well as for multi-axis drive and control tasks. Device types with different equipment and interfaces are available for using the system in specific applications.

Typical areas of application:

- Building automation
- IoT and Security Gateway or Device
- Handling & Robotic

Controls of the ctrlX CORE series may only be operated under the mounting and installation conditions, in the position of normal use and under the ambient conditions (temperature, degree of protection, humidity, EMC, etc.) specified in the related documentations.

2.2 Unintended use

"Unintended use" refers to using the ctrlX products outside of the above-mentioned areas of application or under operating conditions and technical data other than described and specified in the documentation.

ctrlX products must not be used if they are exposed to following conditions:

- Operating conditions that do not meet the specified ambient conditions. Operation under water, under extreme temperature fluctuations or under extreme maximum temperatures is prohibited
- Applications that have not been expressly authorized by Rexroth

3 Safety instructions

The Safety instructions contained in the available application documentation feature specific signal words (DANGER, WARNING, CAUTION or NOTICE) and, where required, a safety alert symbol (in accordance with ANSI Z535.6-2006).

The signal word is meant to draw the reader's attention to the safety instruction and identifies the hazard severity.

The safety alert symbol (a triangle with an exclamation point), which precedes the signal words DANGER, WARNING and CAUTION, is used to alert the reader to personal injury hazards.

The Safety instructions in this documentation are designed as follows:

⚠ DANGER	In case of non-compliance with this safety instruction, death or serious injury **will** occur.
⚠ WARNING	In case of non-compliance with this safety instruction, death or serious injury **could** occur.
⚠ CAUTION	In case of non-compliance with this safety instruction, minor or moderate injury could occur.
NOTICE	In case of non-compliance with this safety instruction, property damage could occur.

4 Security

Operating units, systems and machines always require an implementation of an overall state-of-the-art concept of the IT security.

Bosch Rexroth products are part of an overall concept. The characteristics of Bosch Rexroth products have to be considered in an overall IT security concept.

For the characteristics to be considered, refer to the following documents:

- “Security Guideline Electric Drives and Controls" (R911342562)
- "Secure Configuration Manual ctrlX DRIVEplus" (R911411573)

5 Free and open source software information

The IDE App, the graphic user interface (GUI) and other software in the package include open-source components. Note the terms of use including the open-source software section. For a complete overview on all included open-source components with copyright notes and license texts and notes on how to replace certain components, refer to the information documents on free and open-source software (FOSS).

6 Introduction and overview

The IDE App is an application to create automation programs or program sequences. It provides browser-based engineering environments to develop programs for different runtime systems.

The implemented source code editors support with common comfort functions such as autocomplete, syntax highlighting, linting, etc.

For an easier diagnostics of the created programs, a debug connection can be set up from the browser to the runtime system on the control using the engineering environment. This allows a complete analysis of fault conditions or the runtime response of a program.

7 Installation

License

The following licenses are required to operate the IDE App on the ctrlX CORE control:

Type code	Part number
SWL-XC*-IDE-TEXTUALCODE**-NNNN	R911409788
SWL-XC*-IDE-VISUALCODE***-NNNN	R911409789

Further information

- “ctrlX CORE - Runtime”, Application Manual (R911403768), chapter ➦ Licenses – Overview
- “ctrlX CORE - Runtime", Application Manual (R911403768), chapter ➦ ctrlX CORE – License information

Installation

ctrlX CORE side navigation *“Settings → Apps”*

For more information on how to install the app, refer to the following chapters:

- “ctrlX CORE - Runtime", Application Manual (R911403768), chapter ➦ Apps – Basics
- “ctrlX CORE - Runtime", Application Manual (R911403768), chapter ➦ Apps – Installation

Important information

The IDE app can only check for the availability of licenses if the following permission is set for the active user.

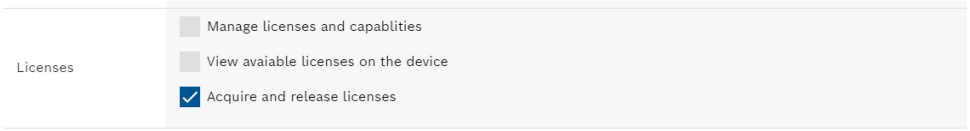


Fig. 1: Permission to query licenses in the license manager

Without this permission, the license in the ctrlX CORE license manager cannot be queried when starting this app. Thus, the app cannot fully be used.

Behavior:

- Visual Coding: Error message "No license available"
- Textual Coding: Endless loading spinner when starting the application

Call:

ctrlX CORE side navigation *“IDE”*

8 Authorization

The IDE App uses the Security concept of the control system and integrates its user management.

The authorization scopes described in the following are supported. These scopes have an impact on the functional scope of the IDE App and provide a targeted, user-oriented or scope-oriented provision of the functions. This means, depending on the selected scope, the user selects the accessible functional scope within the software.

It is differentiated between the following two development environments within the IDE App.

- Visual Coding
- Textual Coding

These development environments do not have any functional connection are taken into consideration individually.

To facilitate full access to the IDE App, “Full Access" can be enabled in the Identity Management of ctrlX AUTOMATION. Subsequently, access to the entire functional scope of the software is possible.

8.1 Visual Coding IDE

This scope sets the basic permission to use the Visual Coding app.

If the scope is not active, the app cannot be used. After the start, the app is in an endless loading loop. Select "back" to open the start page of the ctrlX CORE.

8.2 Textual Coding IDE

The access rights in the scopes in the Textual Coding build on each other. Consequently, activate both scopes for full access. The use of the Advanced access without activation of the base access results in an inconsistent state so that basic functions would not be available.

If functions are accessed that are not within the enabled scope, the query sent to the service is rejected and an error message is displayed in the software. To be able to access the relevant service, assign the relevant authorization for the active user via the Identity Management service of the control system.

8.3 Base

- Basic Textual Coding permissions -

The Base scope provides the basic functionality of the Textual Coding software so that the software can be used in connection with ctrlX CORE.

The following table describes the services manages via this scope.

Service	Description
REST	Access and interaction with ctrlX AUTOMATION via REST, e.g. Start/stop file system accesses or scrip processing

Service	Description
core	The core extension is the main extension for all applications and provides the basic framework for all depending extensions. The extension provides the basic APIs for all applications, including: • Application APIs • Shell APIs • Basic widgets • Items (e.g. commands, menu items, shortcuts)
filesystem	The filesystem extension provides functions for interaction with the file system, including services such as monitoring, uploaded and the basic file tree view.
workspace	The workspace extension provides functions and services to manage work spaces (projects) within the application
mini-browser-service	The mini-browser-service extension provides a browser widget with backend end points
debug	The debug service is used to initialize a new debug session. This service provides functions to configure and start a new debug session.
file-search	The file-search extension provides the command and the option to quickly open each file in a certain work space
search-in-workspace	The search-in-workspace extension provides the option to search all files in a certain work space with different search routines

8.4 Advanced

- Advanced Textual Coding permissions -

The Advanced scope extends the Base scope by higher-order functions and provides in-depth access to the application and the system.

The following list includes the function extensions that can be released via the advanced scope.

Service	Description
tasks	The tasks extension enables the execution of scripts or binary files in the backend of the application
shell-terminal	The shell-terminal extension provides the option to include integrated terminals in the application which can be used in several different scenarios
plugin-ext	The plugin-ext extension provides the plugin API (only for internal use)

9 Visual Coding

9.1 Introduction and overview

The Visual Coding editor provides a platform to manage and configure individual automation tasks in a tool. The integrated editor supports the programming with visual elements (blocks) in Python and JavaScript and allows the deployment on a target system irrespective of the selected language representation.

Open the Visual Coding software via the start screen of the ctrlX CORE in the side navigation.

You are routed automatically to the start screen of the software. The software is opened in the same window.

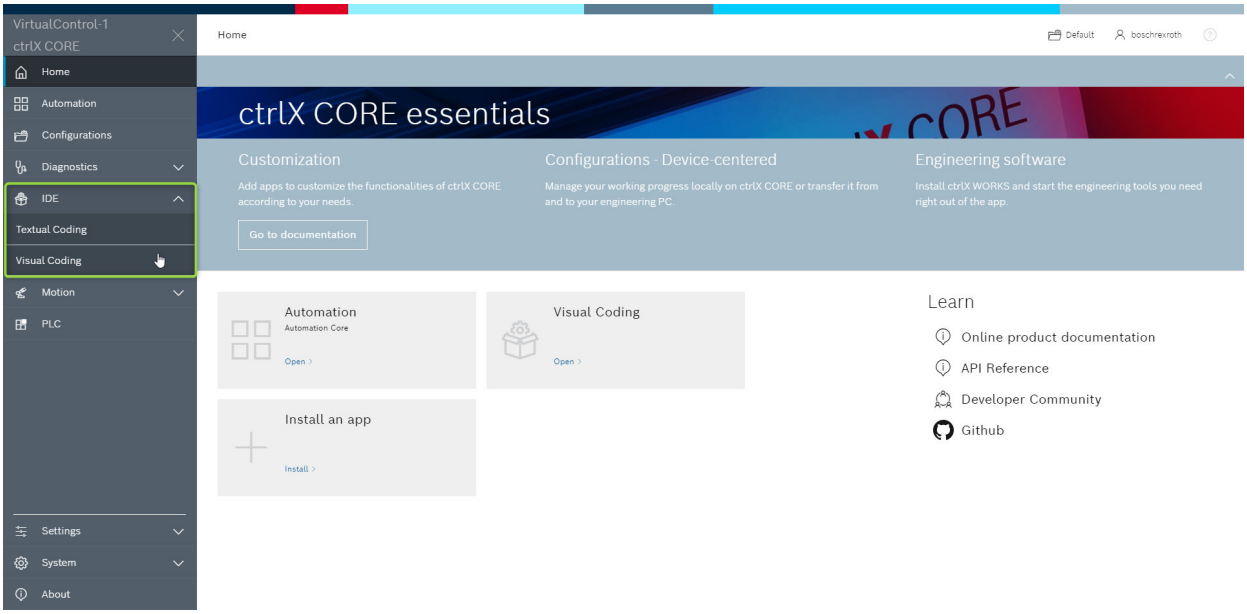


Fig. 2: Overview on the IDE App

Use the logo in the upper left corner to navigate from the Visual Coding interface to the ctrlX CORE interface.

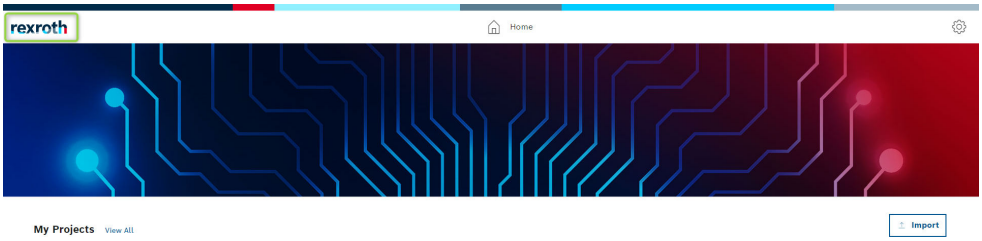


Fig. 3: Navigating from Visual Coding to ctrlX CORE

9.2 Project management and project synchronization

9.2.1 Introduction and overview

Existing projects can be managed on the Visual Coding start page. All projects are shown as tiles when the page opens.

There are two storage directories to store Visual Coding projects.

- File system of the host, e.g. ctrlX AUTOMATION (Active Solution)
- Web browser (browser database)

The Visual Coding software automatically synchronizes from and to the databases available when opening the start page.

Visual Coding supports an offline mode. Thus, if there is no active communication connection to the target system, the editor can also be used offline. In this case, the projects are only saved in the browser database. If there is an active communication connection, the projects are loaded from the control or synchronized to the control when opening the start page.

A project includes all dependencies required to restore a project. The configurations belonging to a project, e.g. extensions or configured objects used such as axes or kinematics. This includes multiple meta information, e.g. which functions are used and how the blocks are allocated in the interface.

The compilation created for the target system is not part of the project. The compilation is stored separately on the target system upon start or deployment. The programs shown in the editor do **not** represent the executable program that is loaded as compilation to the control system. While compiling, this compilation is supplemented by further essential program components.

Projects of the Visual Coding software are stored in the file system of the host. An example shows the ctrlX AUTOMATION in the Explorer view of the Textual Coding software.

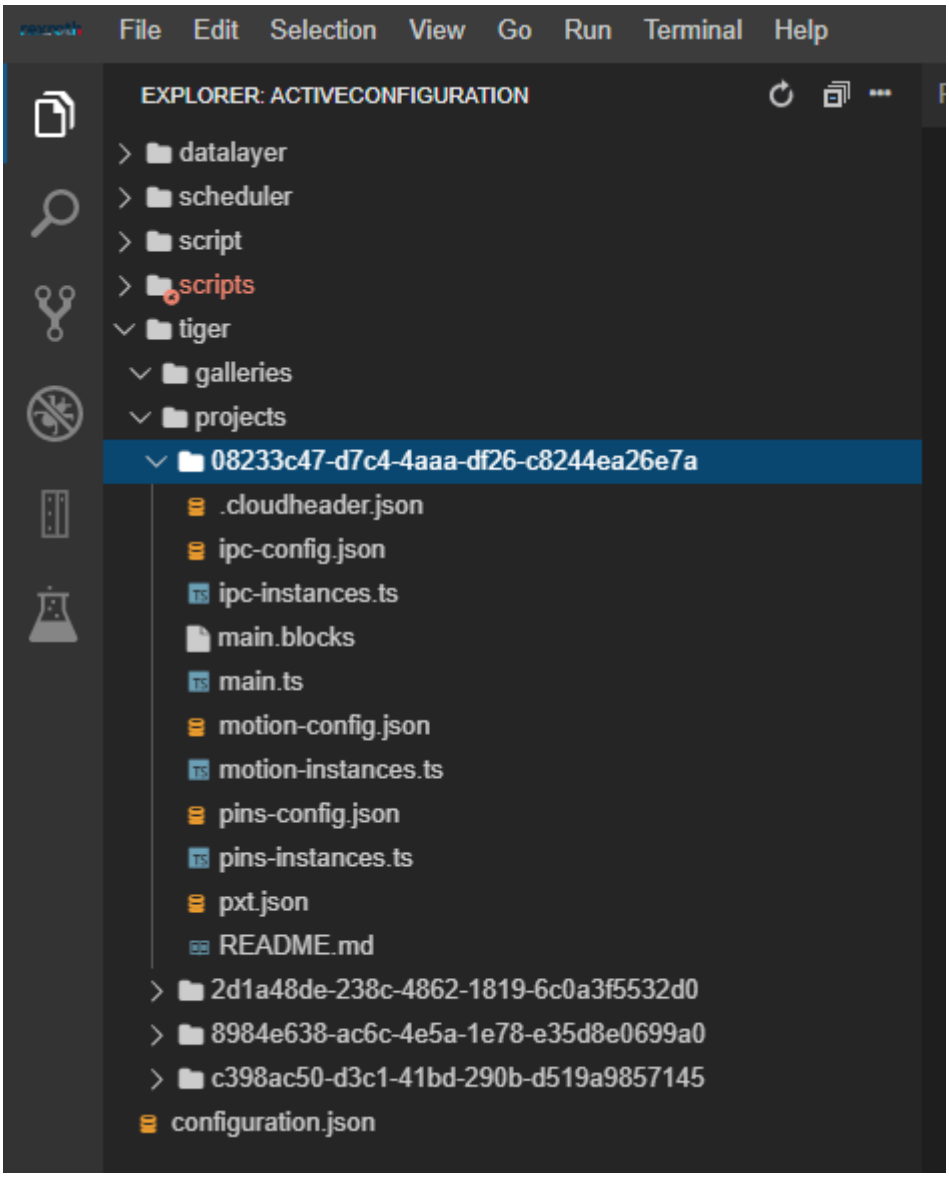


Fig. 4: Storage location of the Visual Coding software projects

9.2.2 Project import

Press the Import button to import previously exported projects into a Visual Coding software.

The following import sources are available:

- Local file system
- GitHub repository

If the project was imported successfully, it automatically opens in editing mode.

9.2.3 My projects “Show all”

All projects are shown in an overview.

The view can optionally be switched via a button in the upper right corner from a representation as list to a representation with widgets.

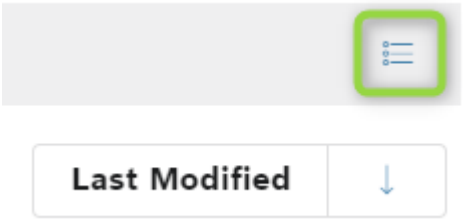


Fig. 5: View of all projects

If a project is selected, it can be opened, copied or deleted.

Multiple projects can be selected in the list view and deleted at once.

9.2.4 Creating a new project

To specify a name for the project to be created, click on the widget “New project” and a dialog field opens. Any name can be specified. It is intended to facilitate the identification in the Visual Coding software.

To open the board selection, click on the “Create” button. The target system can now be selected. The project is created with all its board-specific dependencies and opened in editing mode. The initial project configuration differs with regard to the selected board. The user is thus always requested to execute this configuration.

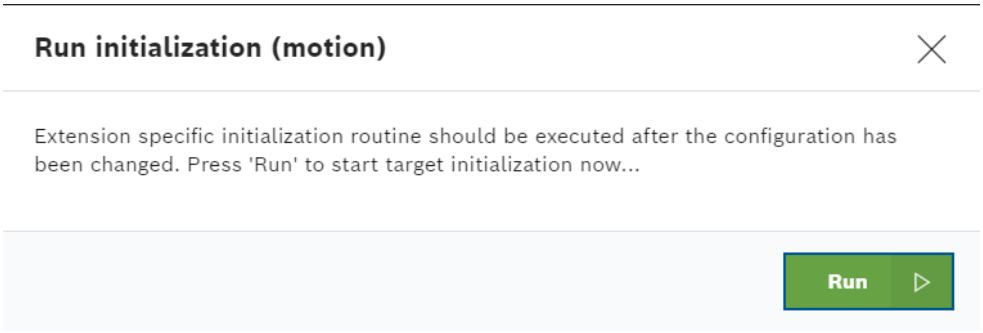


Fig. 6: Request to configure

9.3 Code editor

9.3.1 Structure

The editor section consists of different components.

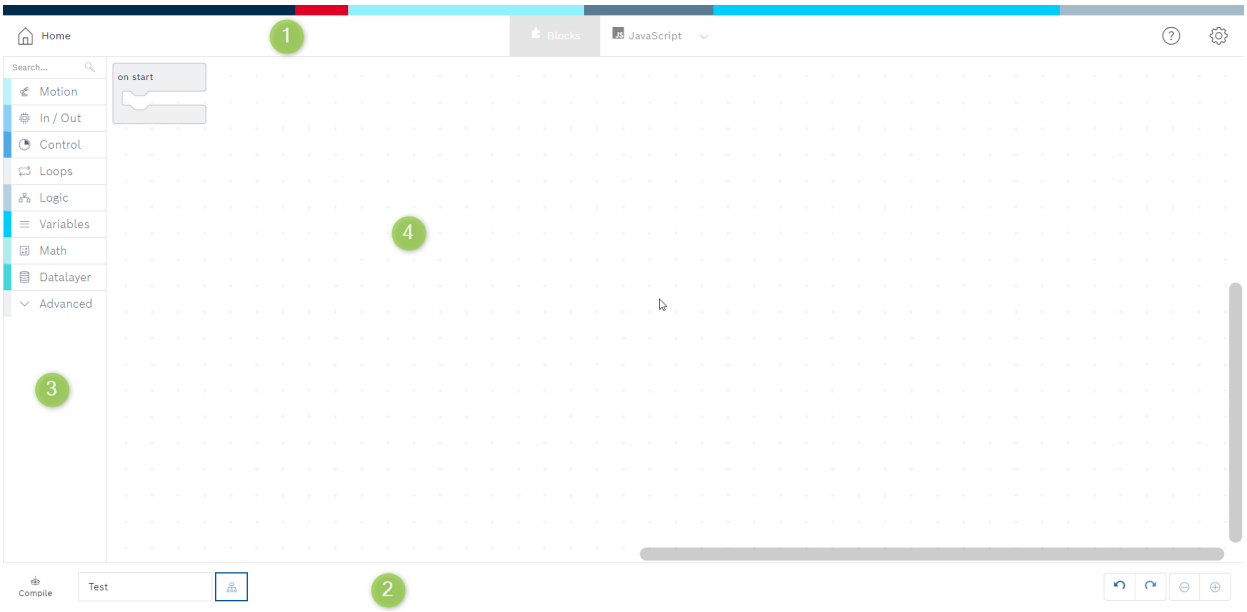


Fig. 7: Design of the editor section
1 Header
2 Footer
3 Extension libraries
4 Code editor (depending on the active representation)

Header

The header consists of the following components:

- Home:
  Navigating to the Visual Coding start page
- Editor switching:
  Blocks <> Python <> JavaScript
- Help:
  Shows the integrated help functions
- Settings:
  Shows the product-specific setting options

Footer

The footer consists of the following components:

- Compile:
  Compiles the Visual Coding project to a file that can be executed for the target system
- Start:
  Executes the created file
- Debug:
  Debugging of the currently edited project
- Project name:
  Shows the current project name. Changes possible
- Github login:
  Allows to log into a GitHub account
- Undo:
  The latest changes can be undone
- Zoom:

Zooming in or out of the editor section

9.3.2 Basis and advanced extensions

Table 1: Basis extensions

Basis extensions	Description
Motion	The Motion extension provides a selection of Motion functions to traverse Motion objects such like axes or kinematics or to query states. The objects to be traversed have to be configured before via the project settings.
I/O	The I/O extension provides selected functions to access inputs and outputs in read-only and/or write access. Digital and analog I/Os is supported. To access the objects, they already have to be configured via the project settings.
Control	The Control extension provides specific functions to affect the program sequence.
Loops	The Loops extension provides control structures such as loops to implement defined repetitions of a specific program sequence.
Logic	The Logic extension provides control structures for comparative operations to check for example whether the state of axis A or the state of axis B has been reached to respond individually during the program sequence.
Variables	Use the Variables extension to create variables. It provides further functions to use these variables during the program sequence.
Math	The Math extension provides blocks to implement to most simple mathematic operations.
Datalayer	The Datalayer extension provides functions for a write or read-only access to Data Layer endpoint of the ctrlX AUTOMATION.

Table 2: Advanced Extensions

Advanced Extensions	Description
Server	The Server extension provides functions to host resources via the Visual Coding app. Different technology providers, e.g. REST or ctrlX SKD, are available:
Functions	The Functions extension allows to define own functions in the editor. These functions can be called multiple times during the sequence to avoid code doubling.
Arrays	The Arrays extension provides basic functions when using fields.
Text	The Text extension provides basic functions when using texts.
Low Motion	The Low Motion extension includes all basic functions provided on the ctrlX CORE. These functions use the original control functionality. No comfort functions are supported. Thus, a self-implemented state monitoring is required.
Tools	The Tools extension provides the basic functions to create and read a JSON object.

Advanced Extensions	Description
Console	The Console extension provides Basis Logging functions to output status messages during the program sequence.

Table 3: Extension Pool

Extension Pool	Description
Extensions from the Extension Pool can be added to a project at any time to extend it with regard to requirements and functions	
REST	The REST extension provides a client implementation to access the resources in write and read-only mode.
IPC	The IPC-extension (Inter Process Communication) provides an interface to access resources of an external runtime environment (e.g. PLC) in write and read-only mode.
MQTT	THE MQTT extension provides basic functions to send messages to an MQTT broker via the MQTT protocol.

9.3.3 Supported editors

The Visual Coding software provides different options to create program sequences.

Three editors are supported that can be used parallely to create or edit projects.

- Visually (graphic blocks)
- Python (text)
- JavaScript (text)

While editing, it can be switched at any time between the language representations using the editor switching in the header. Thus, employee A can develop a program in Python and employee B can modify it later in the visual editor, as employee B might not be comfortable working with JavaScript. In all variants, the library can drag a function from the sidebar and drop it into the editor section.

The most common functions, e.g. syntax highlighting, autocomplete, syntax errors (linting) are supported in the textual code editors. Syntactic errors are highlighted directly. The user is informed if an error is pending when the representation should be switched. Changes can be discarded or syntax errors can be troubleshot.

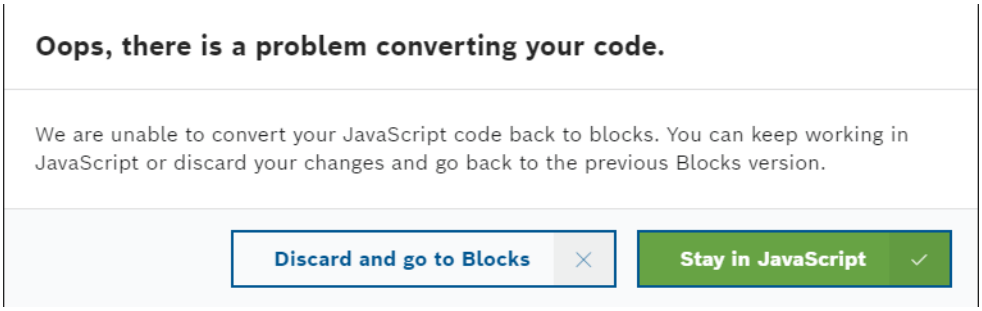


Fig. 8: Error message when switching the representation

9.3.4 File handling

The Visual Coding software provides multiple options to manage the created program.

Before executing a project on the control, it is compiled using an internal compiler. The compiled file is saved to a format that is executable on the control. All relevant dependencies are integrated into the compiled file and stored on the target system.

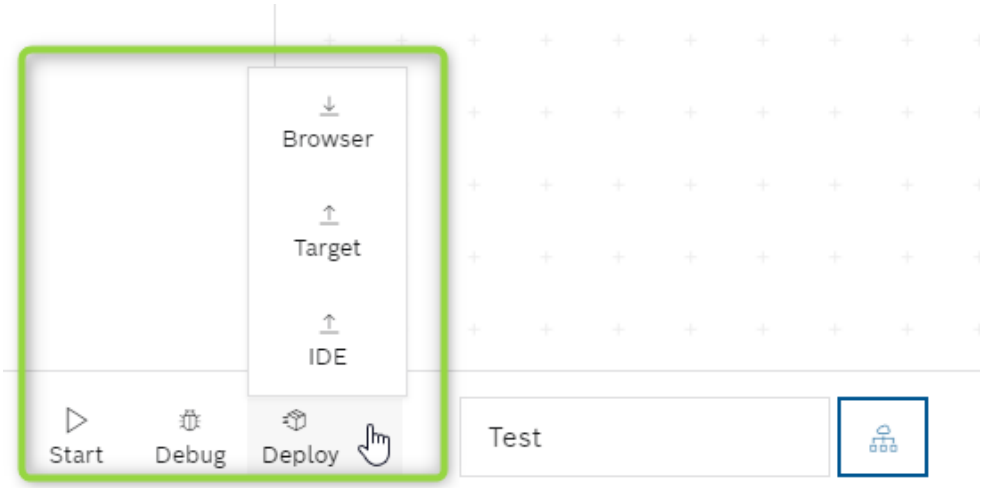


Fig. 9: Project provision

Deploy - Browser

The program created in the editor can be compiled with Visual Coding and downloaded to the own PC using the browser. An individual file name can be specified for the created Python script.

The created script can thus not only be used or saved in the Visual Coding software.

A script thus generated cannot be imported again into the graphic programming.

It is a copy of the currently valid Visual Coding project that corresponds to the respective binary format.

Deploy - Target

To manage programs created irrespective of Visual Coding, a project can be compiled and stored as Python script in the file system of the target system. An individual name can be specified for the Python script. It is stored under */activeConfiguration/scripts/bosch/tiger/UserSpecificName.py*.

A script thus generated cannot be implemented again into the graphic programming.

It is a copy of the currently valid Visual Coding project that corresponds to the respective binary format. To execute the project from Visual Coding, the previously described mechanism for UUID is used and another script is stored in the file system.

Deploy - TC (Textual Coding)

The current Python script opens under the UUID of the Visual Coding project and shows it in the Textual Coding editor.

9.3.5 Starting script

The “Start" button is shown after the compilation.

If the compilation cannot be completed (“Start" button is not shown), there might be a syntax error.

For a more detailed diagnostics and thus a better analysis, go to the JavaScript editor.

Press the “Start” button to execute a created project on the target system. To open from Visual Coding, a name is automatically specified for this file. It is based on a UUID. The created Python script is stored on the ctrlX CORE under the path */skripts/Bosch/TIGER/UUID.py/activeConfiguration/scripts/bosch/tiger/UUID.py* in the Active Solution and executed using the script manager and the Python app. The created script instance is identical to the file name to allow a defined assignment.

The UUID is used to automatically specify a unique name allowing to create the Python script in the file system and to ensure a unique assignment in the system.

Live view

The Live view is started automatically as soon as a project is being processed.

This view is used to show the program progress of the active project if the project is being processed or to show in the console outputs when the console outputs are used in the program sequence.

9.3.6 Stopping script

To close a currently edited script, press the “Stop” button.

The respective script instance is closed and all relevant resources are released. The Live view is closed and the editor view opens.

9.3.7 Debugging script

The Visual Coding software provides an integrated debugger. It can be used in every source code representation (Blocks, Python, JavaScript) When using this function, the current work status of the project is compiled, stored in the file system and executed. A debug connection is set up when starting the script allowing a monitoring and a single step processing of the source code. The Editor view switches automatically to the Debug view after the connection to the web browser has been set up at runtime. Editing stops at the first line of the active source code representation.

Highlighting the active line might take a moment.

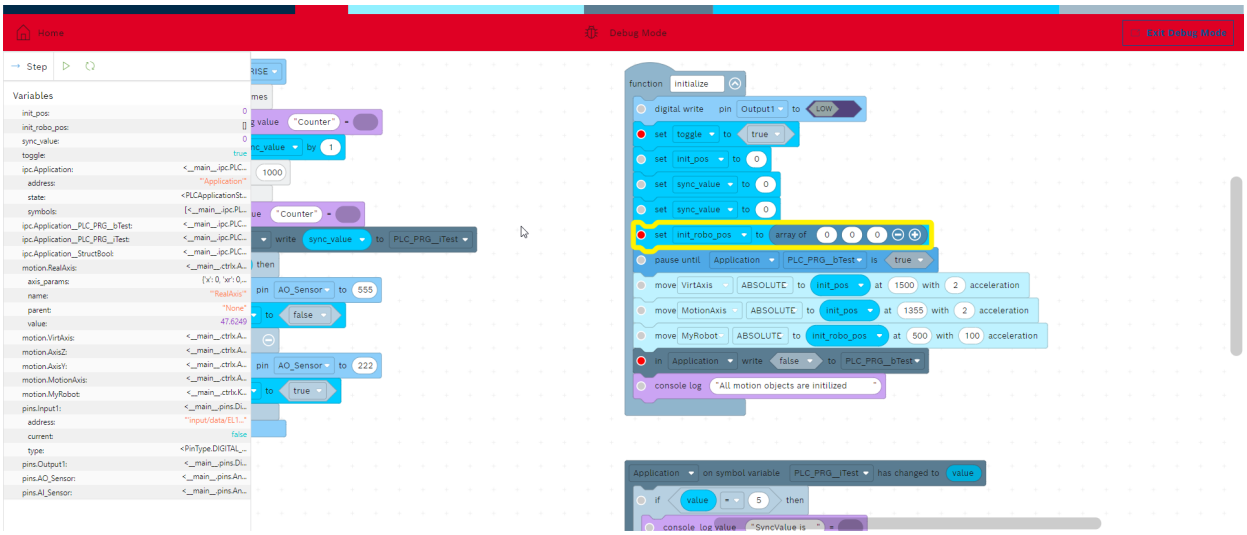


Fig. 10: Debug view

Breakpoint

In the Debug view, breakpoints can be set in each program line. They interrupt the program processing. Highlighting the respective control element on the left side of the editor enables (red circle) or disables (transparent circle) a breakpoint.

Variable watch

For the analysis, all variables of the program are shown in the left sidebar, while the program processing is interrupted (by an active breakpoint or when starting the debugger). Thus, the variable values can be checked at any time to analyze possible malfunctions.

Single step

The single-step function allows to the process a program line and to stop at the next program line after the program sequence has been stopped (by an active breakpoint or when starting the debugger). Programs can thus be analyzed specifically and the variables states can be checked at the processing point in time.

Run

Run continuous the automatic processing of the program sequence until it is interrupted again by a breakpoint. The variable states are updated during program processing.

9.3.8 Attaching active script

It can be attached to a running script and the debug option can be used with the Visual Coding software.

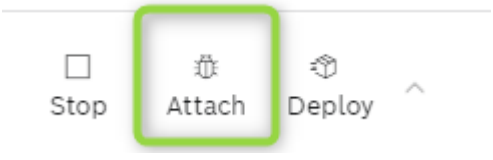


Fig. 11: “Attach” button

However, the “Attach” function has to be active in the project settings before creating (compiling) the script. Only the relevant runtime code is thus generated to use the extension. Enabling the function increases the script runtime, as it has to run through additional code.

During “Attach“ on a running script, a debug connection is set up subsequently to the running process. As the script is processed at this point in time, the debugger does not interrupt the program. If a breakpoint was set and the program processing is at this breakpoint, the program is interrupted.

The program can be analyzed using the following methods:

- Breakpoint
- Variable watch
- Single step
- Run

To exit the Debug mode, press the “Exit Debug” button in the upper right corner. After exiting the debugging, the script goes automatically into the editing mode.

9.4 Project settings

9.4.1 Introduction and overview

Individual settings and configurations can be made for each project in this section. These settings are only applied to the active project and have to made again for further projects if required.

To open the project settings, press the ⚙ button in the upper right corner.

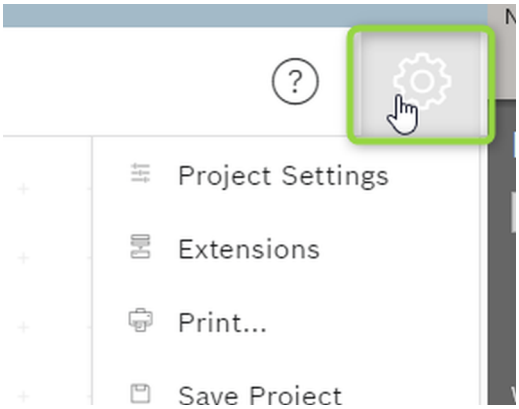


Fig. 12: Project settings

9.4.2 General project settings

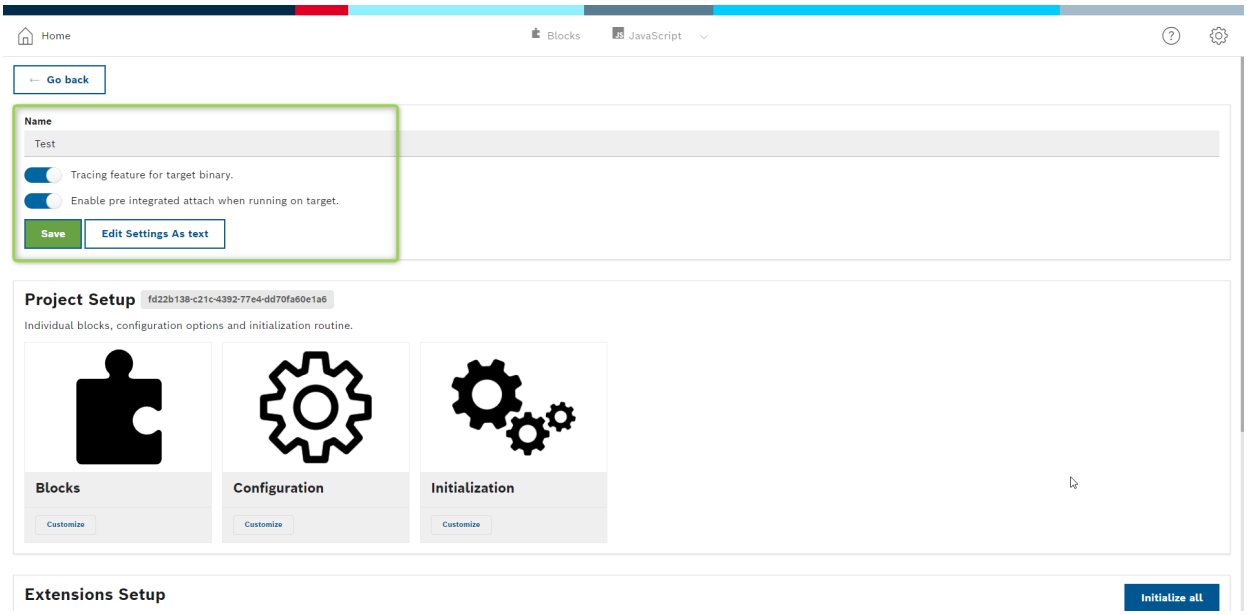


Fig. 13: General project settings

Name

Describes the plain text name used to manage the project in Visual Coding. This name does not have to be unique and can be freely selected.

Tracing

Use this switch to enable a tracing of this program progress. A communication connection is set up while executing the program. This connection sends the program progress to the connected web browser. Thus, the program progress can be monitored in the Live view of a project. After this switch has been enabled or disabled, compile the program again to enable the state in the program code.

This function increases the code runtime, as an additional program code has to be executed to send relevant information to the browser. It is not a real-time communication. Thus, the display (especially basic program calls like loops or conditions) does not correspond to the actually active program line. This program code is not added initially when the function is disabled.

Attach Debug

Use this switch to set up a subsequent debug connection to a running program. The program code is automatically added initially. It allows a subsequent debug connection. After this switch has been enabled or disabled, compile the program again to enable the state in the program code.

This function increases the program runtime, as there is an additional program code. This program code is not added initially when the function is disabled.

9.4.3 Extension configuration and initialization

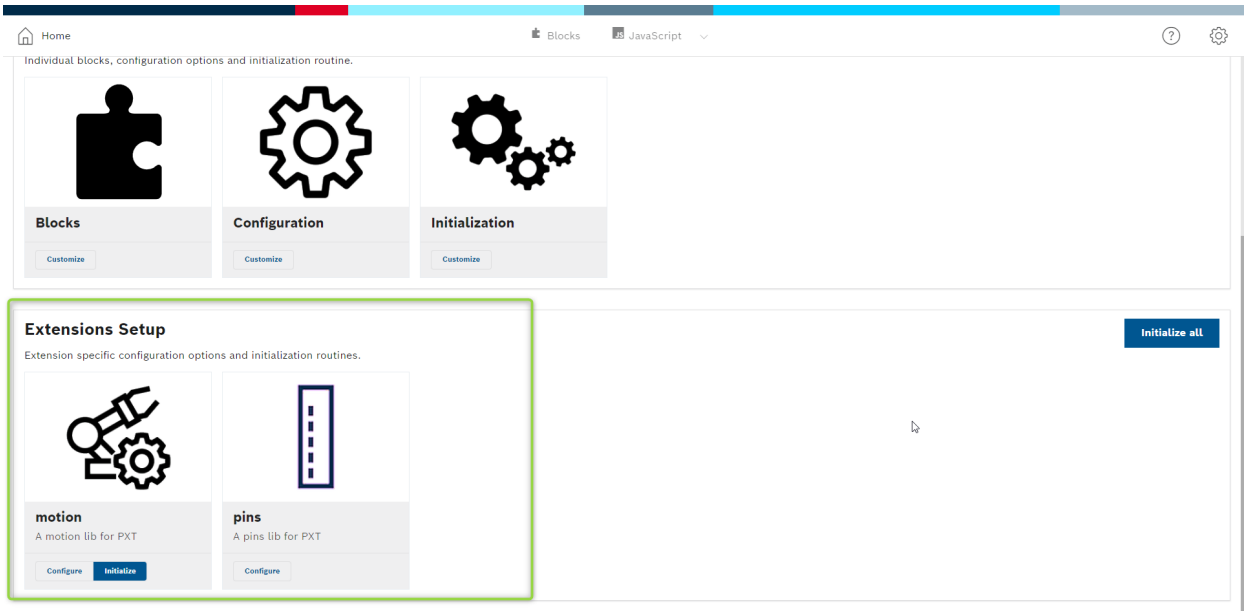


Fig. 14: Extension configuration and initialization

In Visual Coding, the functionality is described as extensions.

These extensions are used to individually design the project functionality. Extensions can allow access to external objects. These objects can be axis or kinematic objects or variables for an interprocess communication with a PLC application running parallelly. To access these objects, configure the respective resources in a project. To ensue that the target system goes into a state that allows to process the created script including the active configuration, an extension can provide an individual configuration sequence. The configuration can be executed individually or all together (sequentially).

The project and extension configurations ae monitored continuously. If a change in the configuration is detected, it is noted automatically after saving that an initialization is required to ensure that the target system goes into the correct state.

Motion extension

Different objects can be configured for the Motion library.

These are the following:

- Axis
- Kinematics
- Points

There is own configuration for each of these objects which has to be executed before its use.

Axis

To create an axis object, a new object can be generated or objects previously generated on the target system can be used for example.

Creating new axis object

The + button opens a dialog to configure a new axis.

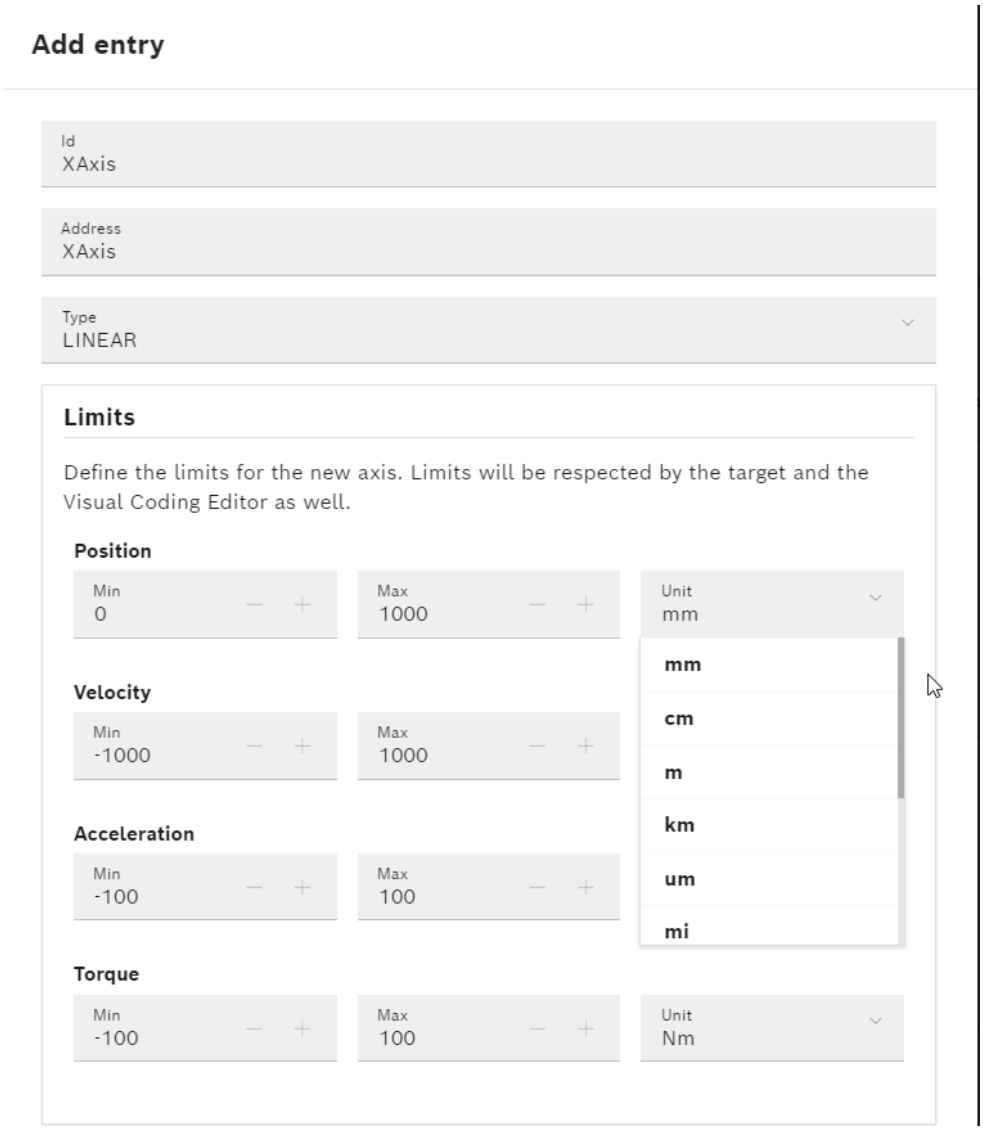


Fig. 15: Creating new axis object

ID	Name in the Visual Coding editor
Adress	Name and representation on the target system Note the target system restrictions when specifying the name
Type	Defines the scaling type of the axis Rotary and translatory scalings are supported The scaling type directly affects the available units for position, velocity and acceleration
Limits	Limit values for the axis to be created • Min = Minimum limit value • Max = Maximum limit value • Einheit = Unit for the physical quantity (position, velocity, etc.).
Axis params	Describes the orientation of an axis in space **Internal only**
parent	**Internal only**

Importing existing object

To access existing target system objects, objects can be imported via Lookup. To show the axes on the system, press the button. They can now be selected from the drop-down menu. To import all relevant elements and to show them in the overview, select and confirm them.

Fig. 16: Importing existing object

Kinematics

To create a kinematic object, a new object can be generated or objects previously generated on the target system can be used for example.

Creating new kinematic object

To configure a new kinematics, press the + to open the respective dialog.

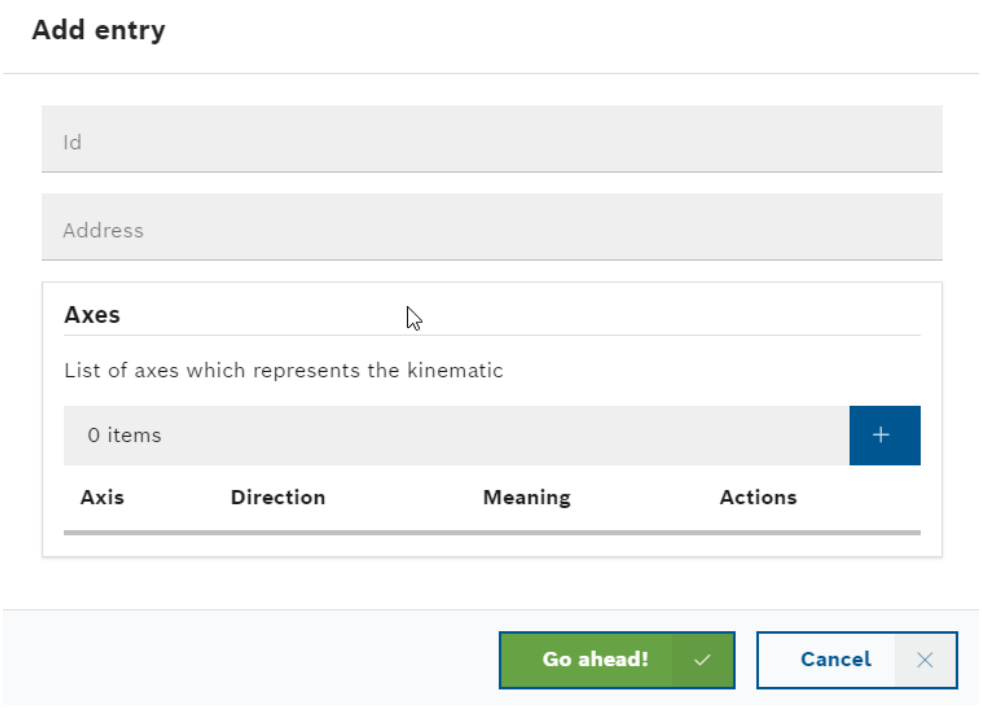


Fig. 17: Creating new kinematic object

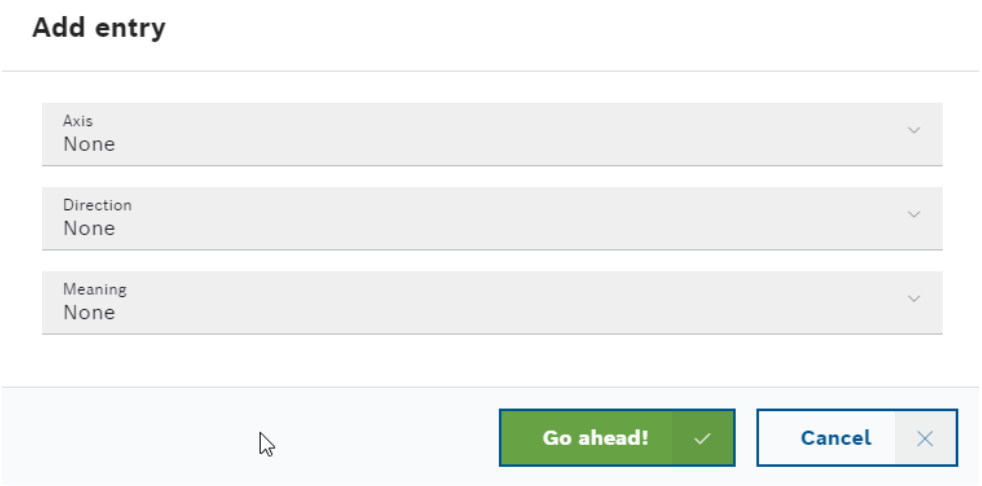


Fig. 18: Configuration of the kinematic object

ID	Name in the Visual Coding editor
Adress	Name and representation on the target system Note the target system restrictions when specifying the name
Adding axis	To add existing axes to the kinematics, use this dialog. Configure them before using the axis configuration dialogs
Selecting axis	Previously defined axes can be selected in the drop-down menu and added to the kinematics
Axis	Selecting axis using ID The drop-down list shows all axes available in the Visual Coding project
Direction	Selecting direction • Positive • Negative
Meaning	Orientation • MAIN AXIS X • MAIN AXIS Y • MAIN AXIS Z • FREE AXIS 7 • FREE AXIS 8 • FREE AXIS 9

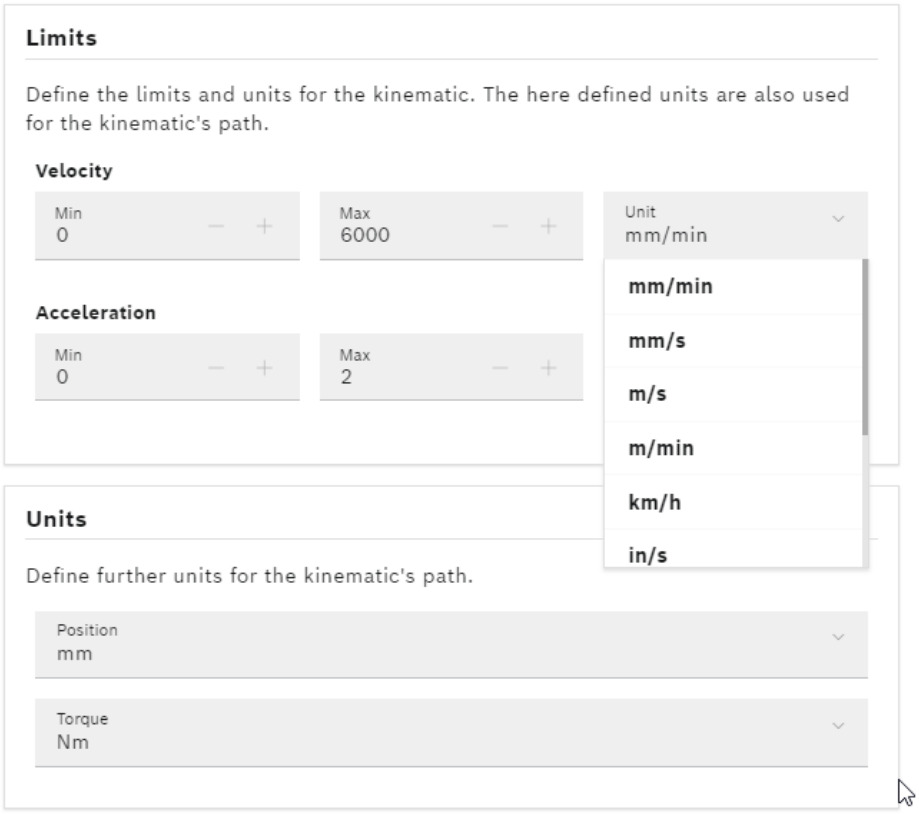


Fig. 19: Limit values and units of the kinematics

Limit values	Limit values for the kinematics to be created Min = Minimum limit value Max = Maximum limit value Einheit = Unit for the physical quantity (velocity, acceleration, etc.).
Units	Defines the unit of physical values on the TCP path.

Importing kinematic object

To access existing target system objects, objects can be imported via Lookup.

All available kinematics are shown in the open dialog. To import all relevant elements and to show them in the overview, select and confirm them.

Initializing Motion extension

An initialization routine is supported for axes and kinematics. These initialization routine checks the states of the selected objects and applies these objects to a defined state.

That means:

- If there are no objects on the target system, these objects are created
- If errors are present at one or multiple objects, these objects are deleted
- If the target system in not in the operating mode, this mode is switched

I/O extension

Use Visual Coding to access input and output modules of the target system. For a read-only or write access to the correct elements, configure these elements in the project settings. A plain text name can be assigned to an input/output in the configuration to ensure a unique reference.

Creating an I/O object

To open the dialog to define an I/O as access point in the Visual Coding editor, press the + button.

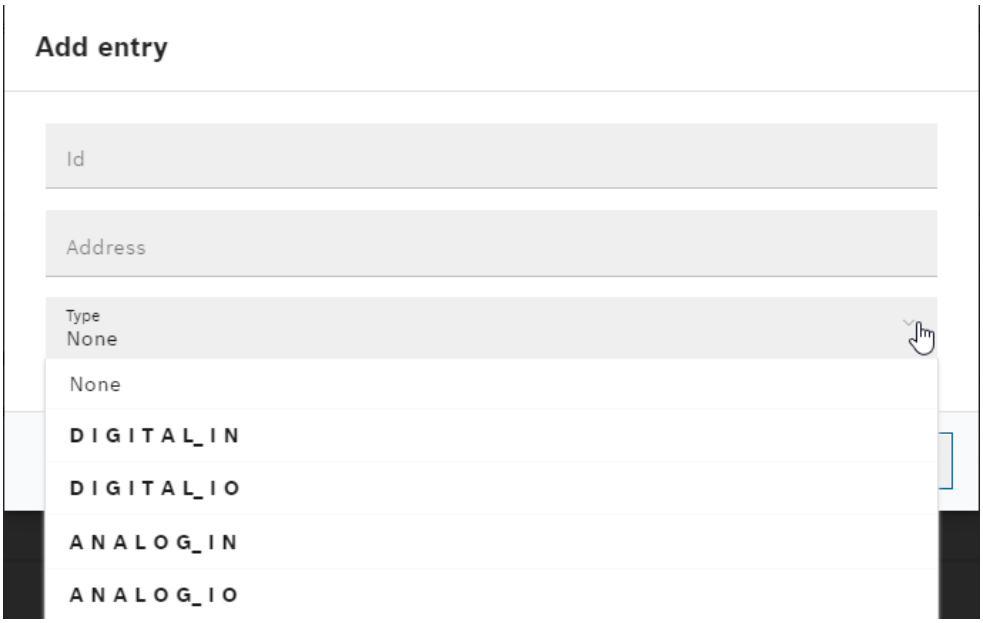


Fig. 20: Creating an I/O object

ID	Plain text name to be used in Visual Coding
Adress	Name and address on the target system, e.g. representation of an input/output as endpoint on the Data Layer
Type	DIGITAL_IN = Read-only access to a digital input DIGITAL_IO = Read-only and write access to a digital output ANALOG_IN = Read-only access to an analog input ANALOG_IO = Read-only and write access to an analog output

Importing I/O object

To access existing target system objects, objects can be imported via Lookup. Press the button and the existing I/O endpoints are shown on the system and can be added to the project via the “Browse” dialog.

Fig. 21: Showing the existing I/O endpoints

IPC (Inter Process Communication) extension

IPC variables can be used to exchange data between two processes during code processing, e.g. between the PLC runtime system and the processing process.

The + button opens a dialog to configure a new IPC variable.

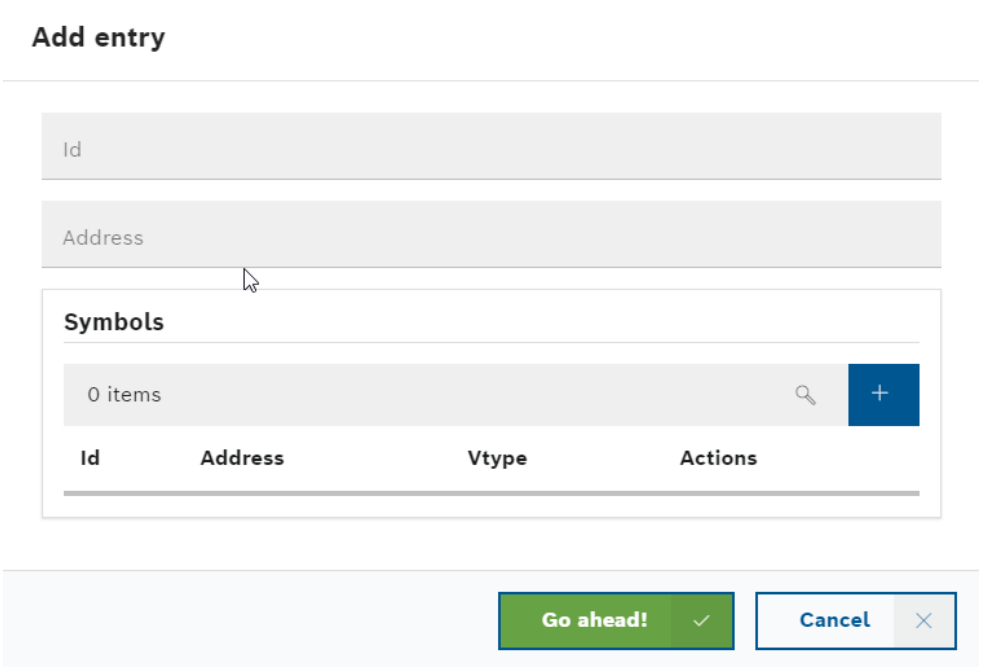


Fig. 22: Creating new IPC variable

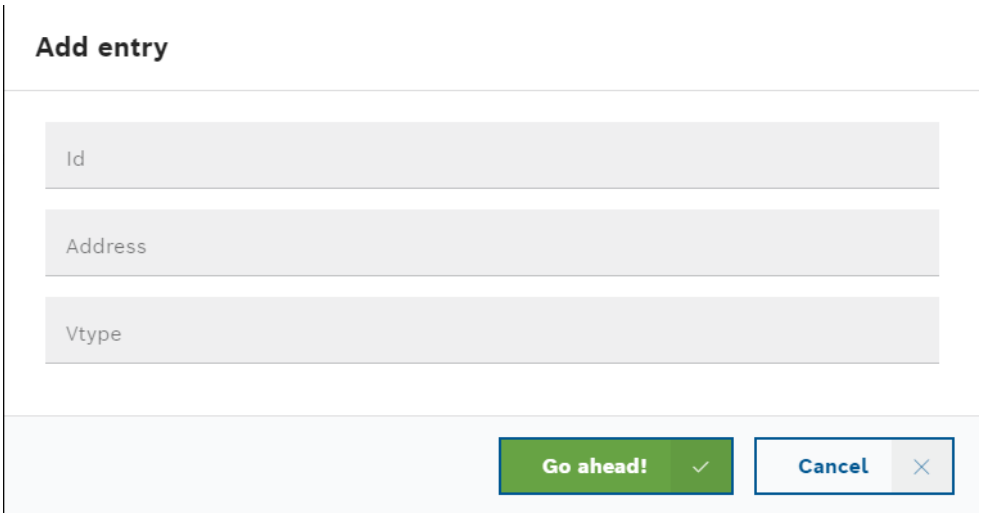


Fig. 23: Configuring IPC variables

ID	Name of the PLC application in the Visual Coding project
Address	PLC application name on the target system
Symbols	List of available symbol variables of the PLC application
Symbols - ID	Name in the Visual Coding editor
Symbols - Address	Name and address on the target system, e.g. representation of an input/output as endpoint on the ctrlX AUTOMATION Data Layer

ID	Name of the PLC application in the Visual Coding project
vType	Representation of the data type on the ctrlX AUTOMATION Data Layer

Importing IPC object

To access existing target system objects, objects can be imported via Lookup. Press the button and the existing PLC applications are shown on the system. After selecting an application, all mapped variables are imported via the symbol configuration.

Fig. 24: Lookup - Importing existing object

The selection can be modified. Open the configuration and delete the elements that are not required. To open the project configuration, press the button. The setting for this configuration can be adjusted later on.

The following changes can be made:

- Deleting entries
- Editing entries (e.g. specifying a new ID)
- Vtype supplement

Only variables stored in the symbol configuration of a PLC application are currently supported.

There is currently only an automated support for the following data types:

- BOOL
- LREAL (floating point)
- LINT
- STRING

For all other data types applies that the Vtype **has to be** adjusted in the configuration to ensure that Vtype corresponds to the data type representation on the Data Layer. It has to be checked and entered manually. If this is not considered, the variable cannot be read/written correctly and a runtime error results (data type incorrect).

MQTT extension

THE MQTT extension provides basic functions to send messages to an MQTT broker via the MQTT protocol.

To open the dialog to define an MQTT broker used to exchange data, press the button.

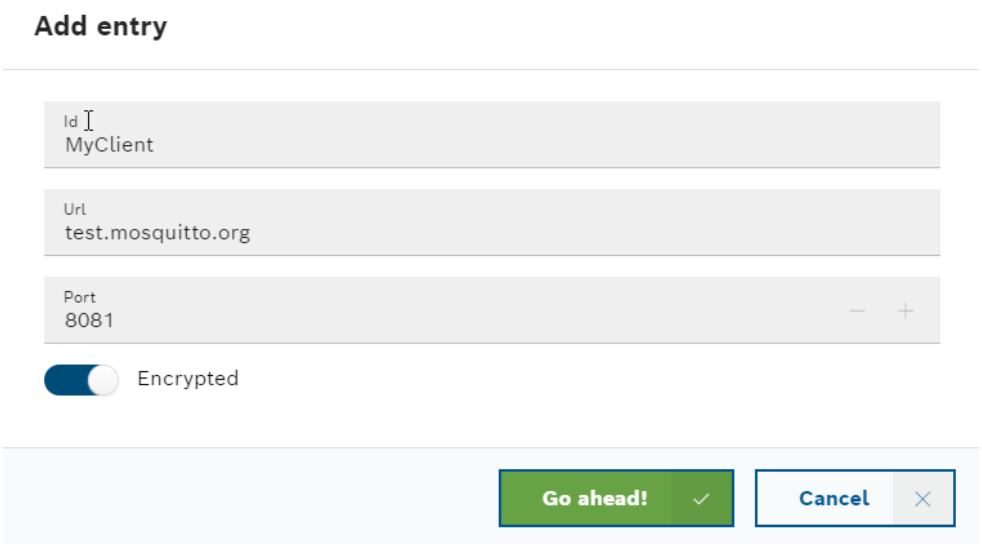


Fig. 25: MQTT broker definition

ID	Client name in the Visual Coding project
Url	Address of the MQTT broker
Port	Port used to send/receive telegrams
Encrypted	Encryption active (Yes/No)

Set a proxy server to access the internet using the environment variable http_proxy.

REST extension

The REST extension provides functions to send REST messages via the HTTP protocol.

To open the dialog to define a REST client used to send requests, press the + button.

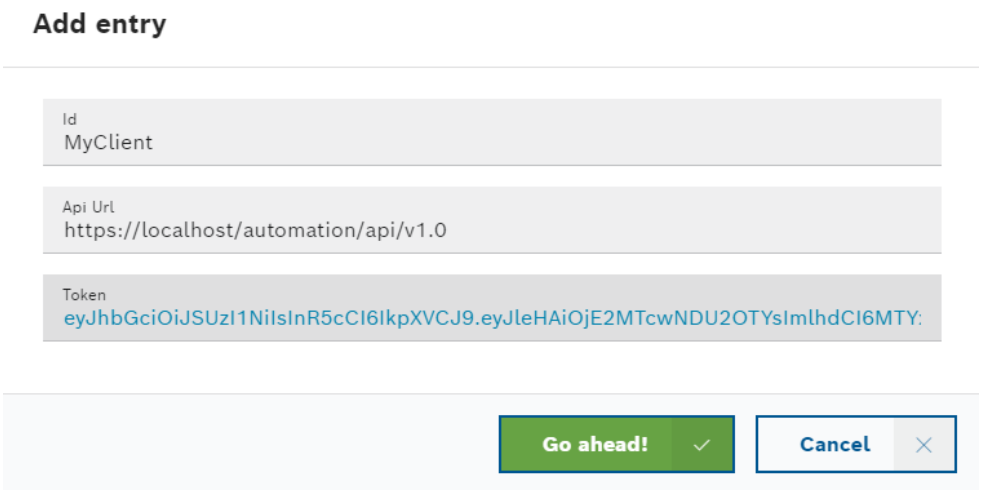


Fig. 26: REST client definition

ID	Client name in the Visual Coding project
Api Url	URL prefix for each REST request. URL has to be qualified completely

ID	Client name in the Visual Coding project
Token	Session token of the current user under which the requests should be processed Note: The token validity elapses after a period “X”

ctrlX AUTOMATION extension

The ctrlX AUTOMATION extension provides resources and functions allowing an explicit interaction with the target system. ctrlX AUTOMATION-specific objects can be configured to use them in the Visual Coding project.

Variables

Select the + button to open a dialog to manually configure the path and the data type on an existing Data Layer endpoint. These values can be accessed after saving using specific blocks from the Data Layer function library.

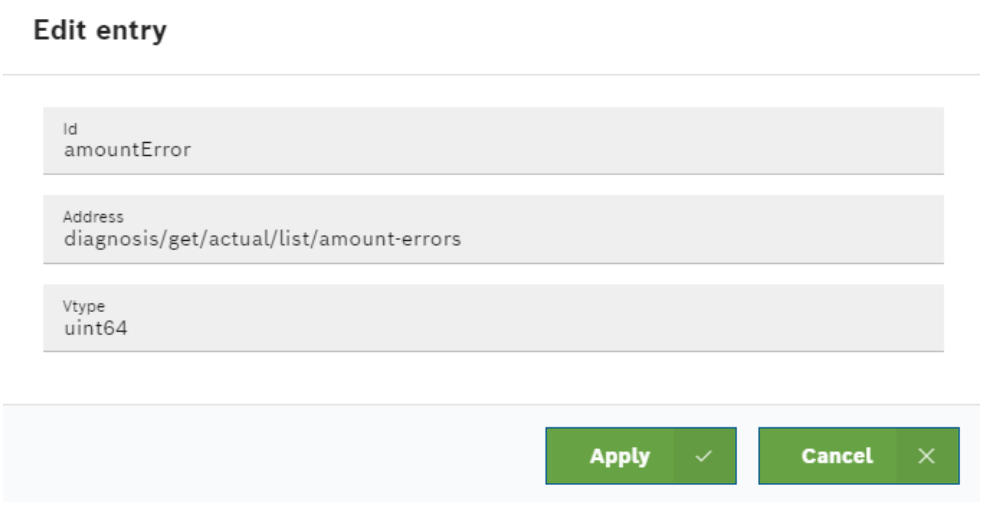


Fig. 27: Configuring variables

ID	Name of the PLC application in the Visual Coding project
Address	Name of the Data Layer endpoint in the ctrlX AUTOMATION
ID	Name in the Visual Coding editor
vType	Representing data type in the Data Layer of ctrlX AUTOMATION

To access existing target system objects, objects can be imported via Lookup.

To show the existing endpoints on the system, press the button.

After the selecting the Data Layer endpoints, the mapped variables are imported.

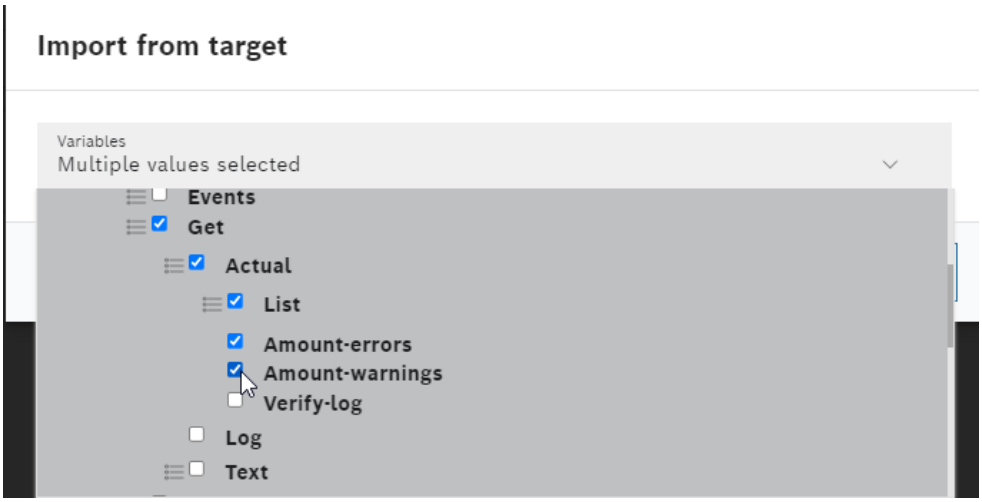


Fig. 28: Configuring variables in Lookup

Server extension

The server extension provides functions to host resources via the Visual Coding app.

Different technology providers are available:

- REST
- Data Layer endpoints

The technology providers allows an explicit interaction with the target system. ctrlX AUTOMATION-specific objects can be configured to use them in the Visual Coding project.

Select the + button to open a dialog to configure a new resource.

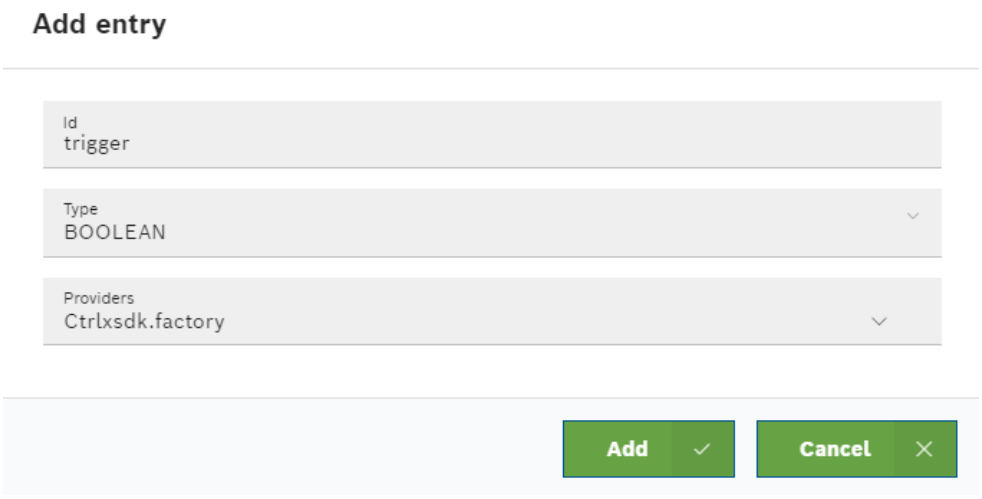


Fig. 29: Configuring a resource

ID	Name of the PLC application in the Visual Coding project
Id	Name of the resource in the Visual Coding project and name of the endpoint in the Data Layer
Type	Data type used to create the resource as endpoint

ID	Name of the PLC application in the Visual Coding project
Providers	Selection of the resource providers used to host the resource • Rest.factory -> REST resources • Ctrlxsdk.factory -> Data Layer endpoints

To access existing target system objects, objects can be imported via Lookup.

To show the existing endpoints on the system, press the button.

After the selecting the Data Layer endpoints, the mapped variables are imported.

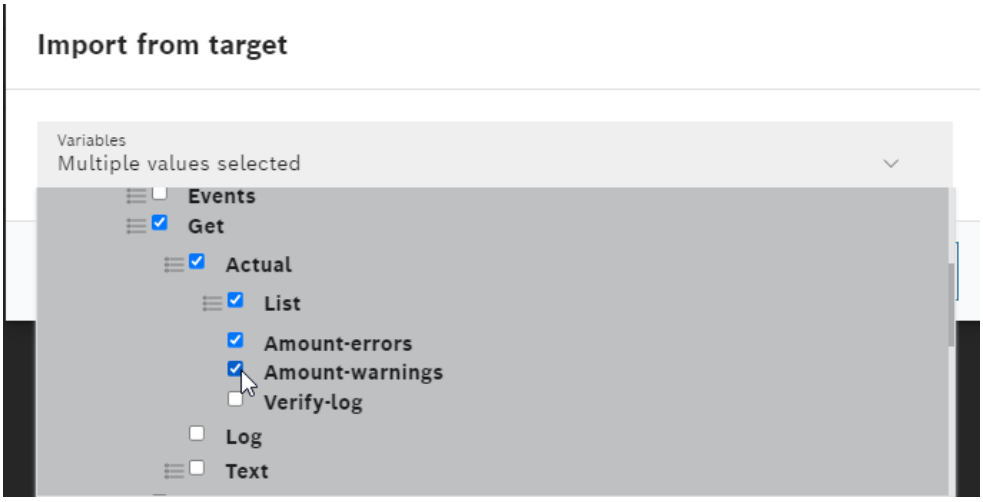


Fig. 30: Configuring variables in Lookup

9.4.4 Printing

Allows to print the sequential project progression for documentation purposes.

9.4.5 Saving the project

Exports a project with all its dependencies in *.png format

To import this project, use the import function on the start page.

9.4.6 Deleting the project

Deletes the project from the browse cache and from the target system. Only project data of the Visual Coding editor is removed. Previously generated script files remain on the target system.

9.5 Help

Opening the integrated help function. Additional information on the product and on individual functions are provided.

9.6 Live display

The Live view is used to retrieve data from the control system and to show that data in the interface. If a project is opened that is currently processed in an active Python instance, the Live view opens automatically. Thus, the program cannot be changed and the lines to be highlighted cannot be displayed anymore.

Program progress

Use the Visual Coding software to retrieve and show the current program progress. The active view (Block/Python/JavaScript) is shown and the active program line is highlighted.

The function has to be active in the project settings. This function increases the code runtime, as an additional program code has to be executed to send relevant information to the browser. It is not a real-time communication. Thus, the display (especially basic program calls like loops or conditions) does not correspond to the actually active program line. This program code is not added initially when the function is disabled. If the function is disabled in the project settings, the valid project representation is shown.

Process information

Data on the active Python process is shown in the process information:

- Process ID
- CPU load
- Memory consumption (in bytes)
- Duration since process start

This information increases the diagnostic capabilities for the active Python process.

Console

The Console log display is used to debug logging information previously programmed in the programming sequence. This information can be used to output debug outputs in the program sequence for example.

9.7 Creating a program

Different representations of programs can be created in Visual Coding.

In the block representation, ready-made blocks and be dragged from the Extension libraries and dropped into the editor section. The functionality can be adapted via the configuration of parameter fields. The provided functions can follow each other as program sequences.

In the representations, Python and JavaScript can also add the functions from the Extension library. The integrated linter is used to support the creation of programs and informs on syntax errors. Autocomplete supports a fast and tool-aided programming. This reduces program errors during programming.

During programming, the representations can be switched at any time to support the preferred environment of each project member.

The representation can only be switched without syntax errors.

After the program has been created, press the “Compile” button in the lower left corner for the compilation. The program created for the target system can now be executed in the runtime environment of the control.

10 Textual Coding

10.1 Introduction and overview

The Textual Coding editor provides a native engineering environment and provides all required tools to create Python scripts. The complete integrated solution allows to create and edit complex projects on the ctrlX AUTOMATION as well.

The complete functional scope of the engineering environment is supported for the following programming languages:

- Python

This includes syntax highlighting, autocomplete, linting and debugging.

The colored highlighting of the syntax is supported for the following languages:

- JSON
- CSS
- HTML
- C++
- HTML
- JavaScript
- Make
- Markdown
- Shellskript
- TypeScript
- XML
- YAML

10.2 Motion and Data Layer integration

To access BuildIn functions of the ctrlX Motion and the Data Layer components, use the Textual Coding software.

To use the BuildIn functions, the Python snap of the ctrlX AUTOMATION has to be installed.

To access the functions in an own script, import the respective components into the Python script. Use the Python “import" function.

Example

```
from ctrlpy import motion
from ctrlxpy import datalayer
```

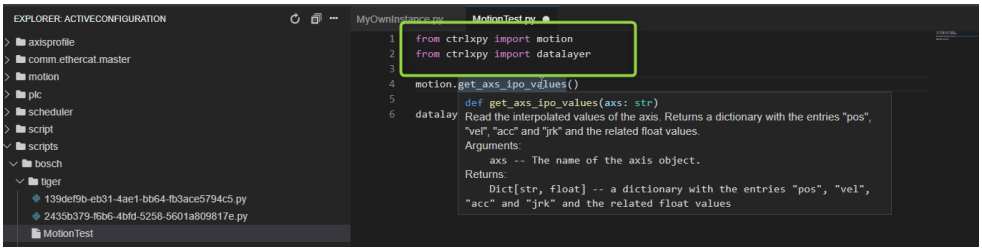


Fig. 31: Python “import" function

The Python modules for Motion and Data Layer provide an integrated functional description incl tooltip help.

For more information on the implementation of the individual function calls, use the function reference of the Python app.

To execute the created script, the modules have to be commented again, as a resolution is not possible at runtime.

10.3 Structure

The editor section consists of different components.

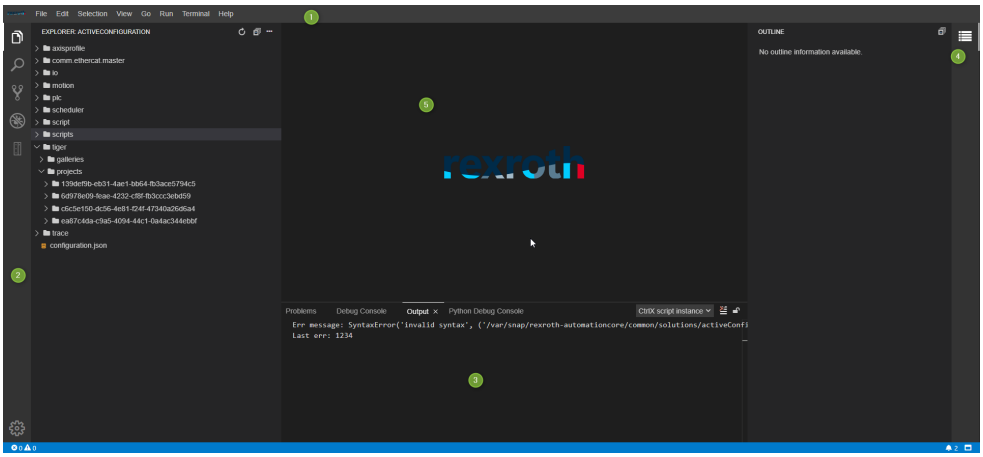


Fig. 32: ctrlX_CORE_MANUAL_IDE_textual_coding_structure_EditorOverview

1 Header
2 Quick launch
3 Output section: List of the different output windows such as terminal or console
4 Outline to display all objects in the active editor window
5 Editor: Shows all active editor windows in the “View” tab

Header

The header consists of the following components:

- File:
  Settings on file level
- Edit:
  Editing commands, e.g. Copy&Paste
- Selection:
  Command for the selection section
- View:
  Display options
- Go:
  Navigation
- Run:
  Debug interactions
- Terminal:
  Interaction with integrated terminal
- Help:
  Help section

Quick launch

The quick launch consists of the following components:

- Explorer view:
  Shows the hierarchic file tree of the active workspace on the target system
- Searching in workspace:
  Search and replace in the complete active workspace
- Debug:
  Executing dedicated debug configurations
- ctrlX AUTOMATION:
  Handling script instances on the ctrlX AUTOMATION control system

10.4 Explorer view

The Explorer view shows all files of the active workspace in a tree structure.

The complete content can be managed on file level. Files can for example be copied, deleted or moved.

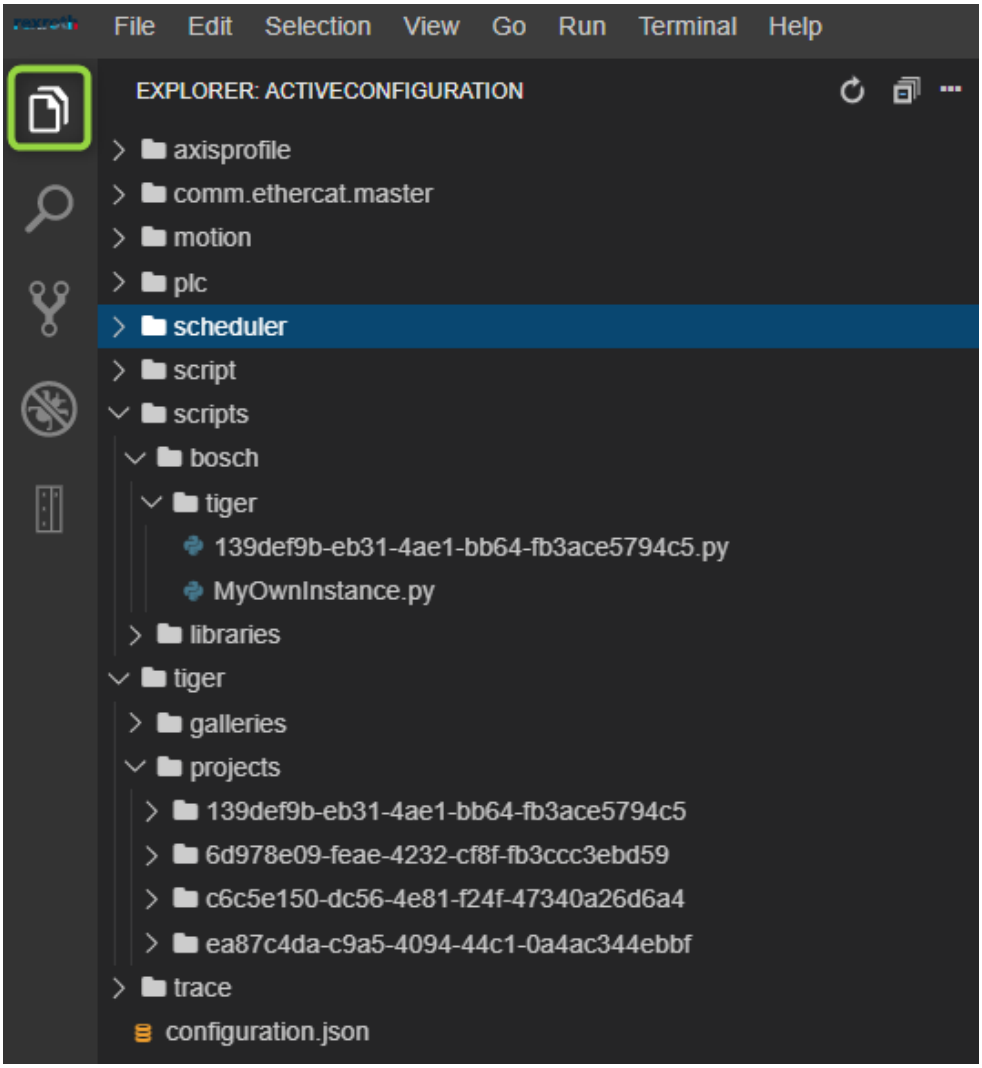


Fig. 33: Explorer view

10.5 Searching in the active workspace

This function allows to search through the complete workspace (all files).

All files with the described character string are shown in a list. Adding more characters, automatically refreshes the search and the hit list is updated continuously. This function facilitates searching for a specific content in huge projects.

Use different parameters to optimize the search function:

- Whole word
- Case sensitivity
- Regular expression
- Excluding specific files

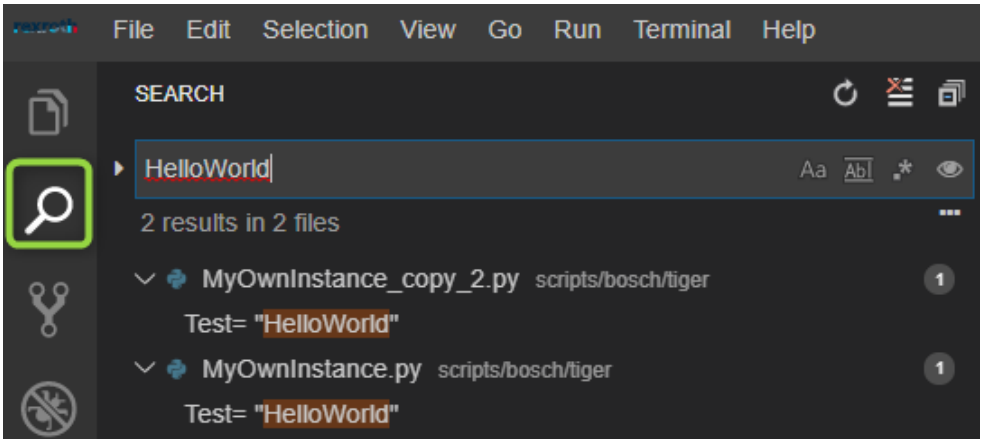


Fig. 34: Searching in the active workspace

Search and replace

In addition to the simple search function, the searched text can also be replaced by a new text. It can be decided individually for every search hit whether to replace the text or whether to execute the command again for all hits.

10.6 Debug

The Textual Coding editor provides the following onboard debug interface: The created code can be analyzed on the target system.

Three debug configurations are supported:

- Python: Current file
  Executing the program code in a local Python runtime environment
- Launch Python in Script Engine
  Creating and executing the program code in a ctrlX AUTOMATION script instance
- Attach to Script Engine Process
  Connects to a running process of a ctrlX AUTOMATION script instance

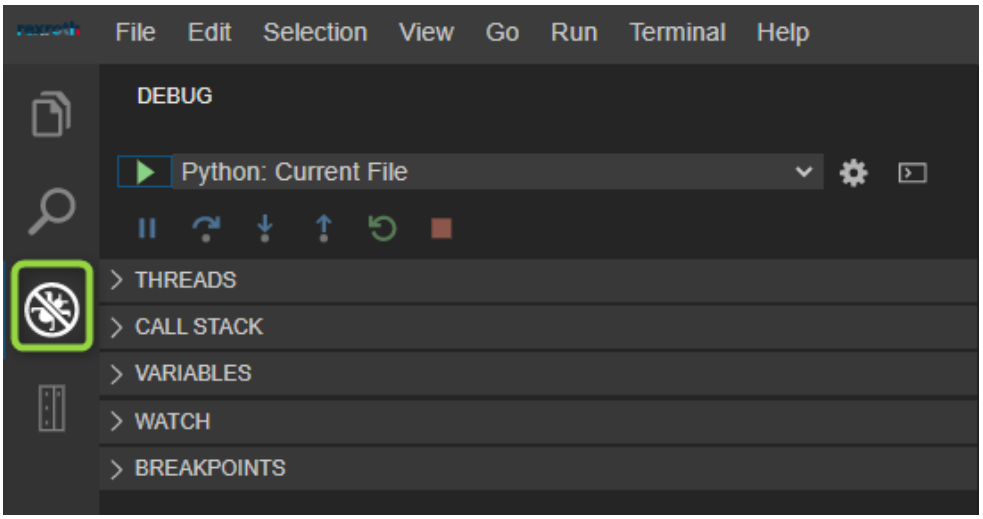


Fig. 35: Debug

Supported functions

- Start debugging:
  Starts debugging in the active configuration. If there is a debug connection to ctrlX AUTOMATION, select an available Python instance first.
  Refer to ➦ Chapter 10.7 ctrlX AUTOMATIONScript management on page 51
- Run:
  Processes the program code until a breakpoint interrupts the processing
- Step Over:
  Executes the active program line and goes to the next expression
- Step Into:
  Calls the child function and switches the view to the program code defined in this function
- Step Out:
  Exits the current view and switches to the level calling this function
- Stop:
  Ends the active debug connection
- Breakpoint:
  To improve diagnostics, breakpoints can interrupt the program code to analyze the active environment

If the program sequence is interrupted, use other analysis tools to analyze the current state. The variable state can for example be shown or the call stack can be analyzed.

10.7 ctrlX AUTOMATIONScript management

The ctrlX AUTOMATION script management provides an interface to interact with the ctrlX AUTOMATION script manager to create and manage Python script instances.

Open the extension via the quick launch.

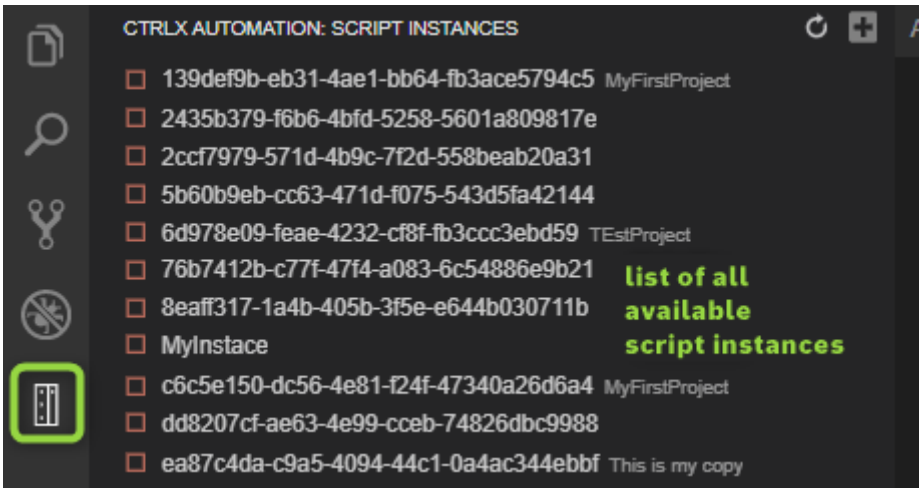


Fig. 36: Calling the ctrlX AUTOMATION script manager

Add

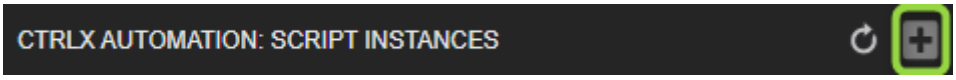


Fig. 37: Add

Creates a script instance under the specified name on the ctrlX AUTOMATION.

A script instance name can be specified in the input field. Press Enter to confirm the specification. The instance is created via the ctrlX AUTOMATION script manager and shown in the list of available script instances.

Consider the naming convention restrictions of the ctrlX AUTOMATION script instances.

Refresh

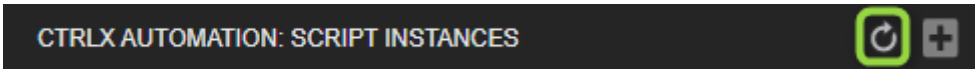


Fig. 38: Refresh

Reads the active state of all script instances created on the ctrlX AUTOMATION.

Start from active Editor

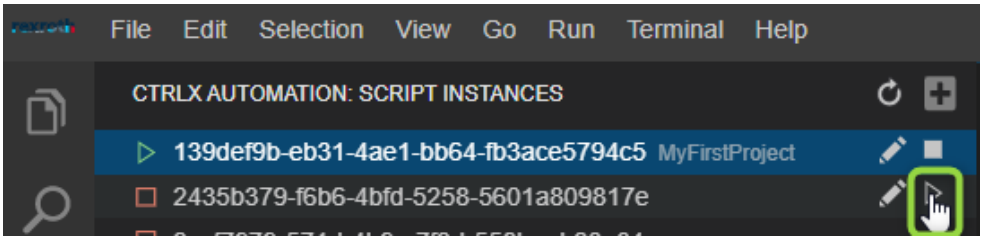
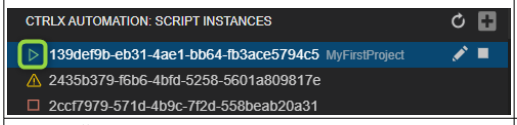
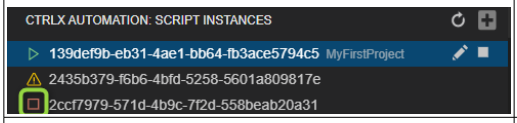
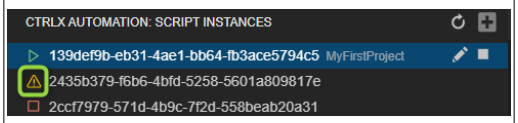


Fig. 39: Start from active Editor

Executes the file active in the editor in this script instance.

File name and storage path are provided to the script manager and the execution command is triggered.

The current status of a script instance is shown in the following.

"Run“ status 	The instance transferred to the script is currently being edited and active.
"Stop“ status 	No script is being edited. The status of the script instance is in "Ready" or "Init".
"Error“ status 	An error is present at the script instance. For more detailed information, go to the ctrlX AUTOMATION Data Layer or press the button. Error information is retrieved and output in the “Debug” console.

10.8 ctrlX AUTOMATION Data Layer browser

The ctrlX AUTOMATION Data Layer browser allow to interact with the Data Layer in the Textual Coding.

It can be browsed through the tree structure. Content of nodes can be output or commands can be issued.

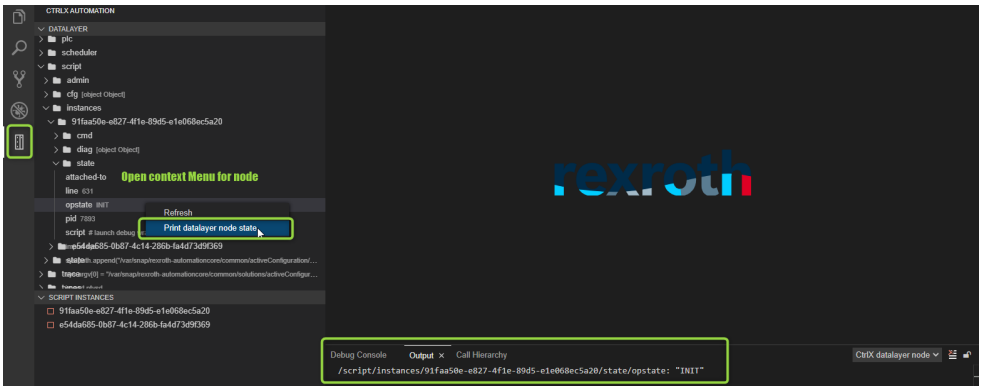


Fig. 40: Outputting a Data Layer node via the Data Layer browser

When issuing a command, note that REST has to be used and that a valid payload has to be sent as well if required. For the correct parameters, refer to the "ctrlX CORE Runtime", Application Manual (➥R911403768).

To show changes at the Data Layer nodes, reload the content manually via the "Refresh" button. There is no automatic update.

11 Uninstalling

To uninstall, note the general information on the control system.

Following data is not deleted in “Active Configuration“:

- Visual Coding projects on the control
- Visual Coding projects in the browser database
- Created Python scripts

Only *.app-specific configuration data is deleted and all connected interfaces do not apply anymore.

12 Related documentation

12.1 Overview

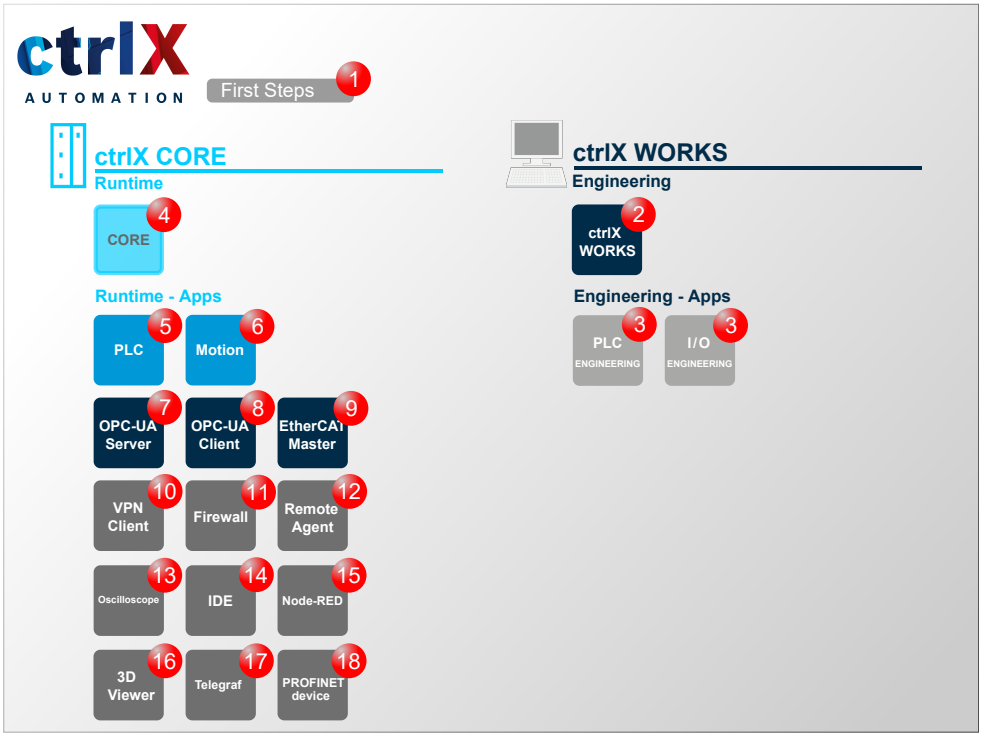


Fig. 41: Overview on further documentations

12.2 ctrlX AUTOMATION

No.	Documentation
1	**ctrlX WORKS First Steps** Quick Start Guide ➦ Web documentation link Ordering information: • DOK-XWORKS-F*STEP*****-QURS-EN-P • R911403760

12.3 ctrlX WORKS

No.	Documentation
2	**ctrlX WORKS Basic System** Application Manual ➦ Web documentation link Ordering information: • DOK-XWORKS-**********-APRS-EN-P • R911403761
3	**ctrlX PLC Engineering - PLC Programming System** Application Manual ➦ Web documentation link Ordering information: • DOK-XPLC**-ENGINEERING-APRS-EN-P • R911403764
3	**ctrlX PLC Engineering - PLC Libraries** Reference ➦ Web documentation link Ordering information: • DOK-XPLC**-LIBRARY****-RERS-EN-P • R911403766

12.4 ctrlX CORE

No.	Documentation
4	**ctrlX CORE - Runtime** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-BASE*******-APRS-EN-P • R911403768
	ctrlX CORE - Diagnostics Reference ➦ Web documentation link Ordering information: • DOK-XCORE*-DIAG*******-RERS-EN-P • R911403770

12.5 ctrlX CORE apps

No.	Documentation
5	**PLC App - PLC Runtime Environment for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-PLC********-APRS-EN-P • R911403787

No.	Documentation
6	**Motion App - Motion Runtime Environment for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-MOTION*****-APRS-EN-P • R911403791
7	**OPC UA Server App - OPC UA Server for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-OPCUA*SERV*-APRS-EN-P • R911403778
8	**OPC UA Client App - OPC UA Client for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-OPCUA*CLIEN-APRS-EN-P • R911403781
9	**EtherCAT Master App - EtherCAT master for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-ETHERCAT***-APRS-EN-P • R911403773
10	**VPN Client App - Remote Support Software for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-VPN********-APRS-EN-P • R911403775
11	**Firewall App - Security Functions for ctrlX CORE** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-FIREWALL***-APRS-EN-P • R911403783
12	**Remote Agent App - ctrlX Device Portal Connection for ctrlX Devices** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-REMOTE*AG**-APRS-EN-P • R911403785

No.	Documentation
13	**Oscilloscope App - Oscilloscope Function for ctrlX Devices** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-OSCI*******-APRS-EN-P • R911409806
14	**IDE App - Integrated Development Environment** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-IDE********-APRS-EN-P • R911410625
15	**Node RED App - Graphic Programming for ctrlX CORE** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-NODE*RED***-APRS-EN-P • R911403789
16	**3D Viewer App - Browser-based 3D Kinematic Simulation for ctrlX CORE** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-3D*VIEWER**-APRS-EN-P • R911416124
17	**Telegraf App - Server Agent for Collecting Data in the Data Layer** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-TELEGRAF***-AP01-EN-P • R911416836
18	**PROFINET device App - PROFINET device for ctrlX CORE** Application Manual ➥ Web documentation link Bestellinformationen: • DOK-XCORE*-PROFINET***-AP01-EN-P • R911417857

13 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts provide you with advice and assistance. You can contact us **24/7**.

Service Germany

Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	**+49 9352 40 5060**
Fax:	**+49 9352 18 4941**
Email:	➦service.svc@boschrexroth.de
Internet:	➦http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide

Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information

To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

14 Index

rexroth
A Bosch Company

Bosch Rexroth AG
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr a.Main
Germany
Tel. +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics

R911410625