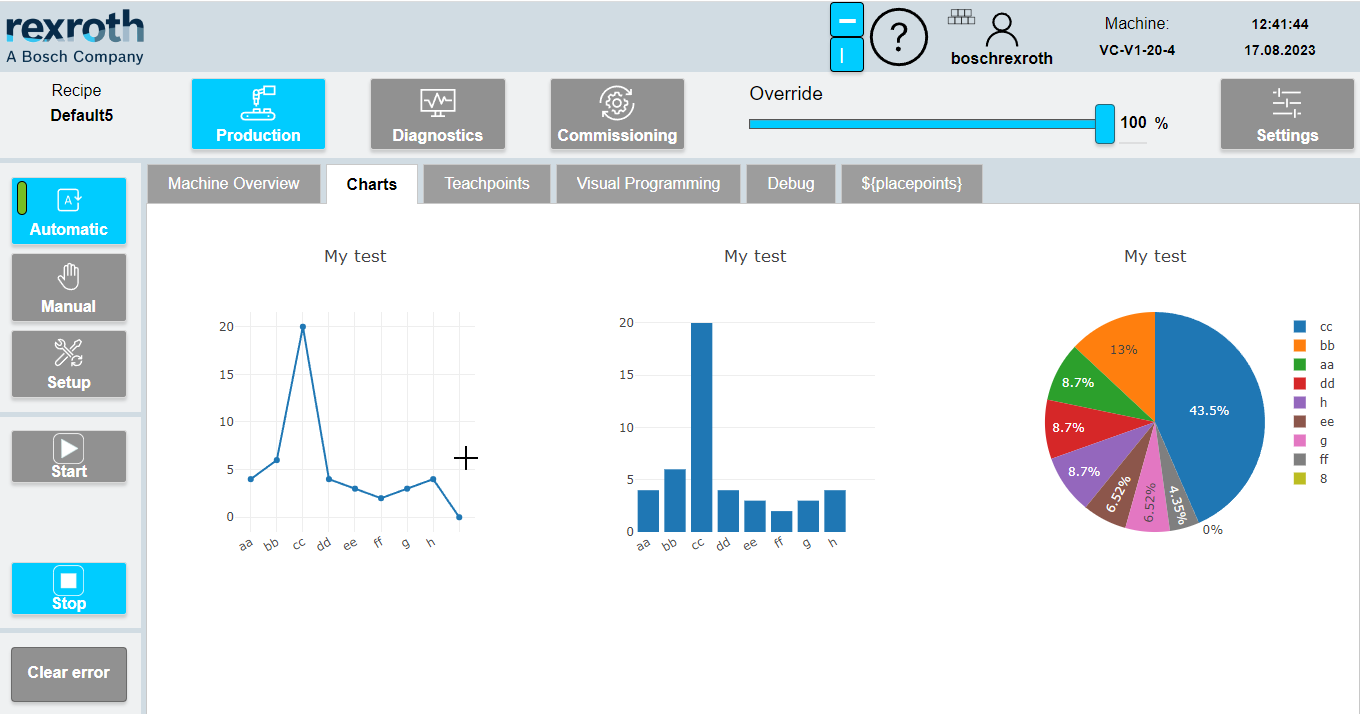


**Handling template for**

**ctrlX AUTOMATION**



Titel

Basic-Hints-Template-Handling

Documentation type

User description

Docu-Type

Internal filing note

Application description

Purpose of the documentation?

Changes

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Note** |
| V00 | 30.08.2022 | Frederik Rueb | Creation |
| V01 | 06.12.2023 | Stefan Baron | Updates for v1.20.0 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of contents**

[1. Change history 5](#_Toc156922764)

[V0.3 March 18, 2022 5](#_Toc156922765)

[V0.4 April 14, 2022 5](#_Toc156922766)

[V1.0 June 21, 2022 5](#_Toc156922767)

[V1.20 December 2023 5](#_Toc156922768)

[2. Introduction 6](#_Toc156922769)

[2.1 General hints 6](#_Toc156922770)

[3. Requirements 7](#_Toc156922771)

[3.1 Apps 7](#_Toc156922772)

[3.2 Libraries 7](#_Toc156922773)

[4. First Startup 8](#_Toc156922774)

[4.1 ctrlX CORE 8](#_Toc156922775)

[4.2 PLC Engineering 8](#_Toc156922776)

[4.3 WebIQ 9](#_Toc156922777)

[4.3.1 Load WebIQ project 9](#_Toc156922778)

[4.3.2 Edit WebIQ project 10](#_Toc156922779)

[5. Known bugs 12](#_Toc156922780)

[5.1 Motion App 12](#_Toc156922781)

[5.2 HMI 12](#_Toc156922782)

[5.3 PLC 12](#_Toc156922783)

[5.4 Restrictions 12](#_Toc156922784)

[6. Features 13](#_Toc156922785)

[7. Features in detail 14](#_Toc156922786)

[7.1 Choose method for power on via fieldbus 14](#_Toc156922787)

[7.2 Adapt axis in ref signal 14](#_Toc156922788)

[7.3 Adapt axis & kinematic configuration 15](#_Toc156922789)

[7.4 Adapt axis and kinematic limits 16](#_Toc156922790)

[7.5 Configure kinematic switches 16](#_Toc156922791)

[7.6 Configure common features 16](#_Toc156922792)

[7.7 PLC state machine 17](#_Toc156922793)

[7.8 Code generation for inputs/outputs 18](#_Toc156922794)

[7.9 Display active step of PLC automatic program in HMI 20](#_Toc156922795)

[7.10 Virtual keypad 22](#_Toc156922796)

[7.10.1 Configuration of alphanumerical keypad 22](#_Toc156922797)

[7.11 Help documentation 23](#_Toc156922798)

[7.12 Charts 23](#_Toc156922799)

[7.13 Oscilloscope 24](#_Toc156922800)

[7.14 Teach point table configuration 24](#_Toc156922801)

[7.15 Alarm/Warning/Info Management 25](#_Toc156922802)

[7.16 Debug> Plc/Python 25](#_Toc156922803)

[7.17 Debug> Plc/Python 26](#_Toc156922804)

[8. WebIQ 27](#_Toc156922805)

[8.1 Documentation 27](#_Toc156922806)

[8.2 Setup development environment (reverse proxy) 27](#_Toc156922807)

[8.3 Package Manager 28](#_Toc156922808)

[8.4 Color Themes 28](#_Toc156922809)

[8.4.1 Color schema 1 Corporate Design Rexroth 28](#_Toc156922810)

[8.4.2 Color schema 2 Dark Theme 29](#_Toc156922811)

[8.5 Composites 30](#_Toc156922812)

# Change history

### V0.3 March 18, 2022

* PLC:
  + Bugfix in state machine. Now all errors can be cleared
  + Code optimization for stability and better maintainability
  + Step mode in PLC automatic program for debugging
  + Configuration of motion data like max. Jog velocity, Kinematic limits
  + Usage of Safe Areas in ctrlX CORE
  + Code generator for Input/Outputs force mechanism
  + EuroMap variables are moved to variables **ix: ST\_INPUT** and **qx: ST\_OUTPUT**
* HMI:
  + Revision of GUI design
  + Update of changed OPC UA variable names
  + Display hint, when WebIQ screen is no editable, because of wrong op mode or user.

### V0.4 April 14, 2022

* PLC:
  + Improvements in state machine
  + Code optimization for stability and better maintainability
  + Update of internal visualization
  + Additional checks in code generator for Input/Output force mechanism
  + There is still a bug in KinInterface library, that some errors could not be deleted.

### V1.0 June 21, 2022

* PLC:
  + Implement State machine in ST, this allows customer specific code extensions
  + Workaround for Start/Pause in mode Automatic
  + Code optimization for stability and better maintainability
* HMI:
  + Revision of GUI design

### V1.20 December 2023

* PLC:
  + Separate Base and Customer code/variables
  + Use original CXA\_MOTIONINTERFACE library, to avoid additional effort on updating
  + Optimize source code, Restructure, and rename some variables
  + GVL\_CFG as central place to enable/configure features
* HMI:
  + Add Color Theme manager
  + IDE Visual Coding via iframe
  + System information: Axes, kinematics, Apps, Licenses & Host info
  + Replace composite widgets by package widgets
  + Add charts and oscilloscope feature (controlled by PLC code)
  + Display active step of PLC automatic program in HMI (controlled by PLC code)
  + config.js as central place to enable/configure features

# Introduction

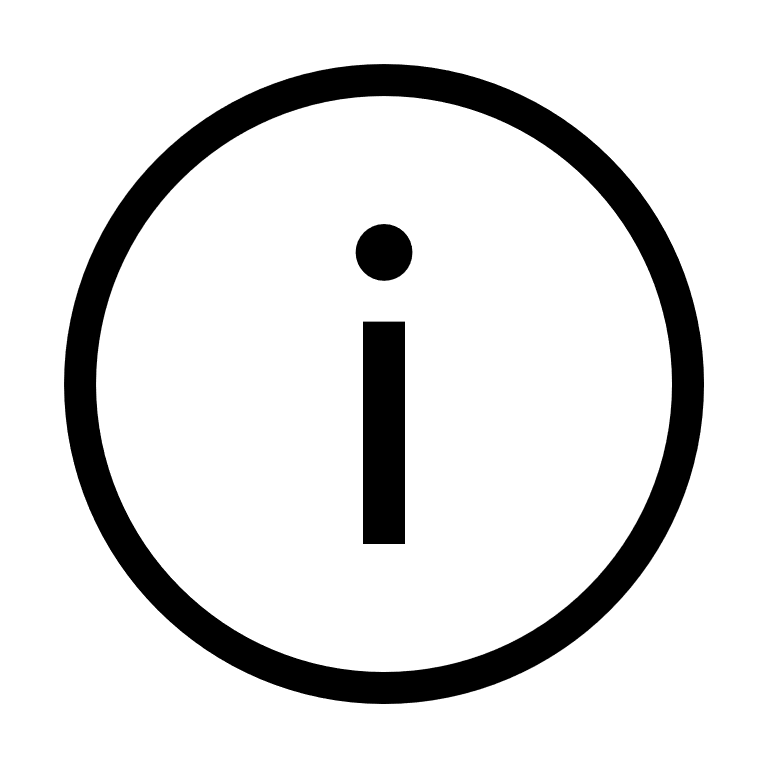
The following documentation describes how to use the HMI-Template (for Handling). It helps to setup the template on your ctrlX CORE or ctrlX COREvirtual. Furthermore, it explains how the template is created and gives tips how to modify or create your own HMI Solution. Start at this point if you want to install the template for your first time.

## General hints

This template is based on an example of a customer’s project. Therefore, keep in mind the following things:

* Example code uses specific axes, kinematic names.
* **FB\_Machine\_Base** is protected, to prevent any changes of the code.

The template is created with ctrlX CORE version 1.20.5. There is no guarantee that the code will compile successfully in newer ctrlX CORE versions without adaptation.

To understand this documentation, you **must** be familiar with the basics of ctrlX AUTOMATION and WebIQ.

See How-To section in ctrlX AUTOAMATION Community: <https://developer.community.boschrexroth.com/t5/Store-and-How-to/Collection-of-how-to-videos-blogs-and-examples-for-ctrlX/ba-p/12343>

This entry is a collection of very helpful videos and documentation for wide range of know how.

See PLC Engineering manual

<https://docs.automation.boschrexroth.com/doc/154626822/ctrlx-plc-engineering-system/latest/en/>

See WebIQ manual  
<https://www.smart-hmi.de/user/download/deliver/docs/documentation-manual-webiq-designer-2.13-e58c/index.html>

ctrlX CORE - IDE appTextual and Visual programming

[Episode #10 PYTHON Programming](https://youtu.be/vuNJXB8HP2c)

[Episode #10 PYTHON Programming](https://youtu.be/vuNJXB8HP2c)

Graphical and Textual Python Programming using IDE App [[DOCU]](https://developer.community.boschrexroth.com/t5/App-Zone-and-How-to/Graphical-and-Textual-Python-Programming-for-Pick-amp-Place/ba-p/25399) - [[VIDEO]](https://youtu.be/ljooYMHs-qo)

# Requirements

## Apps

The following apps are necessary to run all features of the provided example. You need the licenses for all apps. For WebIQ Server you need a license with 500 items or more.

**Mandatory**

* EtherCAT Master (Get access to I/O’s & drives via EtherCAT fieldbus)
* Motion (Set up, commission, and move axes and kinematics)
* OPC UA server (communication to WebIQ)
* PLC (develop and run PLC applications)
* WebIQ Server App (HMI server 2.15.6 or newer)   
  (To edit HMI project you need to install WebIQ designer on your PC)

**Optional**

* 3D Viewer (Display 3D model of moving kinematics)
* IDE (Integrated develop environment for python with file manager)
* Python Runtime (to execute python scripts)

## Libraries

These following 2 libraries are included in the project archive of this Template, because they are not part of the ctrlX PLC Engineering installation:

* App\_CommonData
* App\_Tools

# First Startup

## ctrlX CORE

* Install apps and licenses (see chapter [3.1 Apps](#_Apps))
* To keep template simple, it uses the default user and password **boschrexroth**
* In a real project, for security reasons, you should create a user only with OpcUA rights for OpcUA communication.
* The same applies for the user used for the REST API communication.

To load axis, kinematics definition and python scripts:

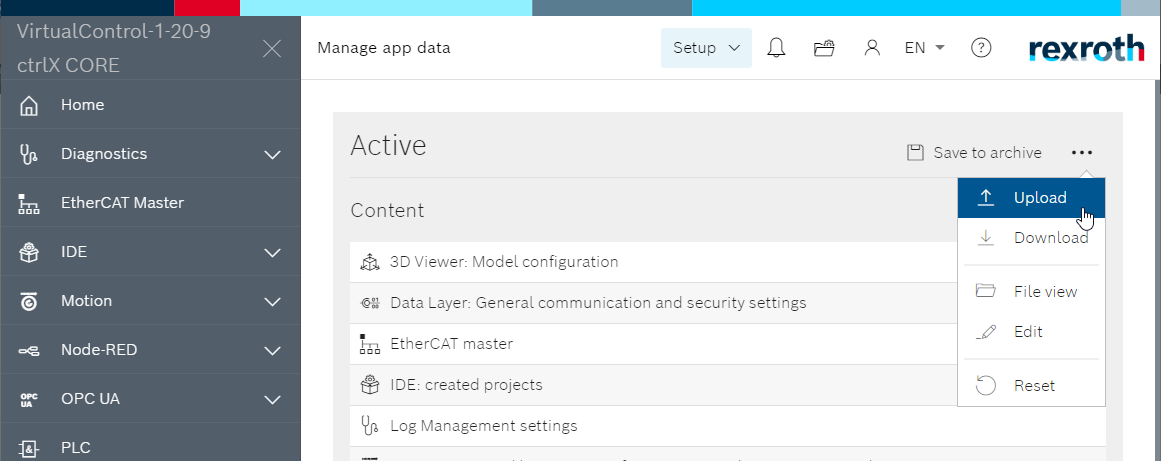
* Switch system state to “Setup” (1)
* Click “Manage App Data” (2)
* Click “…” -> “Upload” (3) and select **AppDataTemplate-10Ax2Kin**. The archive contains 10 axes and 2 kinematics.
* Open branch **Motion** to view the imported axes and kinematics.

Figure : Manage App Data

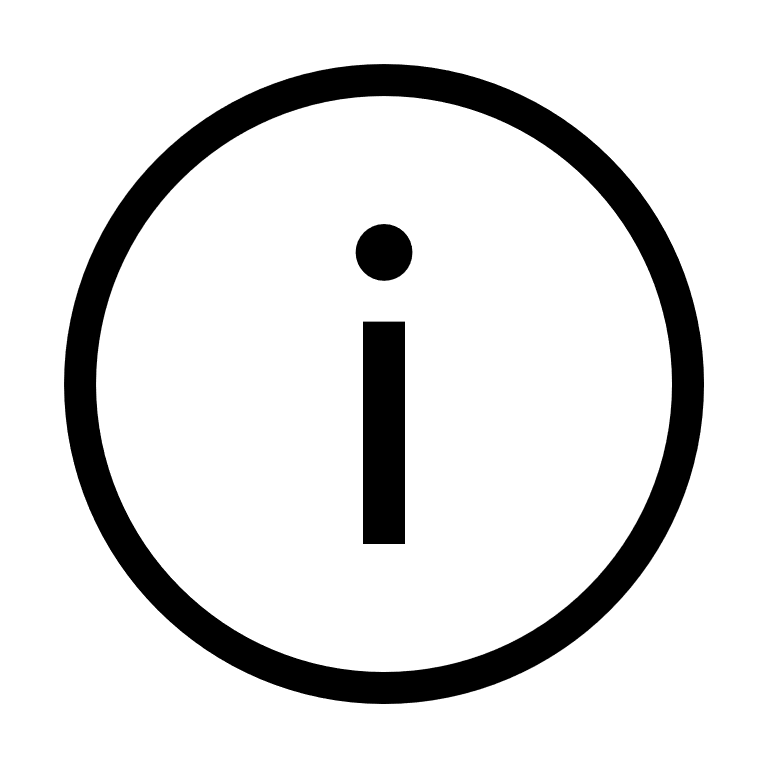
## PLC Engineering

When you are not familiar with PLC Engineering, see chapter 2.1 General hints

* Unzip Base-Template-Handling-V1.20.0.zip and open PLC project archive with ctrlX PLC Engineering.
* Open project: File> Open Project…
* Check if project device matches your device (virtual or real ctrlX CORE)
  + Right click ctrlX CORE… in the project tree
  + Select Update device
  + Select device you are using
  + Click on button **Update device**
  + Click on button **Close**
* Restriction for ctrlX COREvirtual**:** Only WebIQ Runtime is supported
* Check IP settings**:**
  + Right click ctrlX CORE… in the project tree
  + Select **Communication settings…**
  + Enter IP or select it from selection box
  + Click on button **Apply** to verify connection
* Login to ctrlX CORE
  + Online> Login (Provide user & password)
* Start PLC program
  + Debug> Start

**Hint:**

* Dependent on your program requirements, you might have to adapt Task Configuration.
* ctrlX COREvirtual has no real time operating system, therefore the cycle time and watchdog must be big, or disabled to avoid PLC stops caused by a watchdog.
* Only necessary with real axes:
* Currently the power activation is hardware dependent
* Check which option mPower1()-mPower4() meets your setup
* Set PersistentVars.iPowerVars to 1-4 to select a power activation method
* When no option matches, create your own mPower5()

So far, no performance tests and analyses have been carried out. Therefore, no experience about shorter and longer cycle times is available.

## WebIQ

When you are not familiar with WebIQ see chapter 2.1 General hints

## Load WebIQ project

Login to ctrlX CORE web page and click on “WebIQ Runtime Manager”. The first time after installation of WebIQ on ctrlX CORE you must define a user and password for login (e.g., User: **admin** password: **boschrexroth1**). This user is valid for “Runtime Manager” and “Designer”.

* In menu item **Hmi Projects** **(1)** press **Upload (2)** button and select the WebIQ project
* Press **…** and select **Start (3)** to run the project
* Press **…** and select **Open (4)** to display the project in browser

**Hint**: When you forget the password, you must delete and reinstall the WebIQ app. This means all WebIQ projects on ctrlX CORE are deleted.

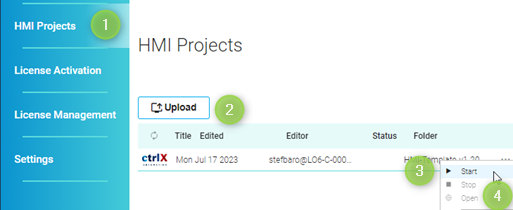


Figure : WebIQ Runtime Manager

## Edit WebIQ project

The WebIQ Designer is the Engineering tool of WebIQ. When you have installed a newer WebIQ version than you must update the project by loading, updating, and saving it.

To connect the WebIQ Designer to the ctrlX CORE, you must add some parameters. Create a new shortcut for the WebIQ Designer application and click on **Properties**. In this example the used IP address is “10.52.244.3“*.* At download page of WebIQ in ctrlX Store, there is a documentation which describes this in detail. There is even a script, doing the work for you.

You must add:

“<WebIQ\_Installation\_path>\Designer\webiq-designer.exe" --ws-host=10.52.244.3 --wsport=10123 to the target input field and apply the changes.

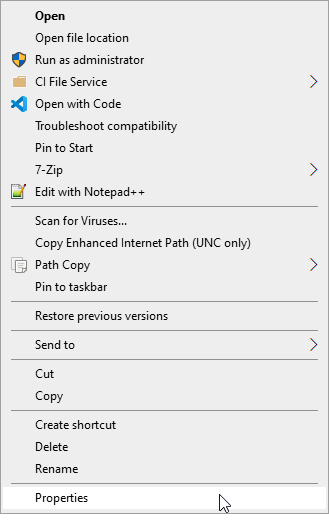
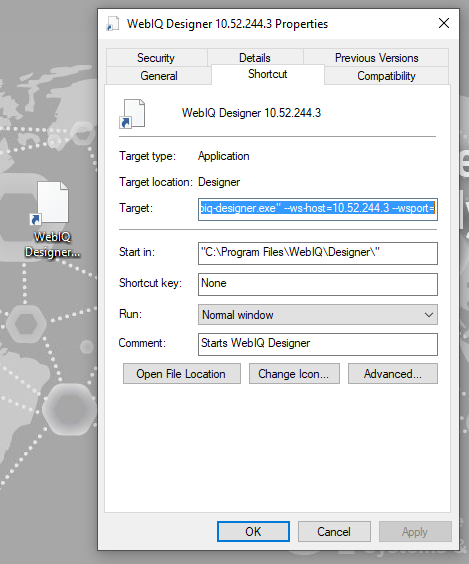
 

Figure : Shortcut Properties

Start WebIQ Designer with the shortcut and login.

* The first time after installation of WebIQ on ctrlX CORE you must define a user and password for login. Th user is used by “Runtime Manager” and “Designer”.
* Import Project HMI-Template.zip
* Load Project
* Install packages (containing additional features) via WebIQ Package Manager.
* Publish Project
* Start program (see 4.3.1 Load WebIQ project)



To establish the OPC UA communication the OPC UA client certificate of SmartHMI must be trusted in ctrlX CORE. This must be done only during the first startup.

To trust the certificate, open ctrlX CORE Web page, go into Settings > Certificates & Keys > OPC UA Server. Click on “**…**” of the Smart HMI certificate and select “Trust”.

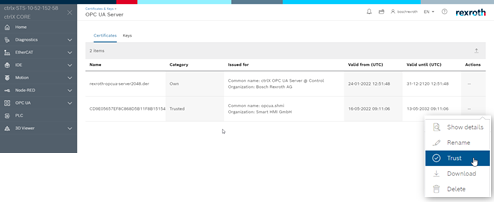


Figure : Trusted OPC UA Certificates

Start the machine and test the different functionalities of the Template. For example, you could modify the Teachpoints. Keep in mind that the most inputs are only possible in mode manual or setup. To start the movement, switch to automatic mode and press the **Start** button.

# Known bugs

## Motion App

* For further information see release notes of ctrlX CORE.

## HMI

* Unicode characters (e.g., special Asian chars) can’t be exchanged via OPC UA.   
  (Bug in OpcUA server). Workaround for file names implemented.
* Some custom widgets are not yet using the theme manager colors.

## PLC

* Sometime errors can only be cleared by resetting the PLC program
* Very rarely, activating an OP fails and must be selected a second time
* Very rarely, override value is not used correct (slow movement). Change override to fix.
* The PLC internal visualization is not up to date; therefore, some screens/functions are not working.

## Restrictions

* In mode Automatic and Manual, the Safe Areas are used
* In mode Setup, the Safe Areas are not used.

# Features

* Axis/Kin configuration checking at startup of PLC program. Problems are shown in HMI.
* Automatic load DC bus and switch drives to Ab
* Last active recipe is automatically loaded at ctrlX CORE startup
* Jog widget for single axes and kinematic (HMI menu item: Commissioning> Jog)
* Display and force digital IO’s via HMI (Commissioning> Inputs/Outputs)
* Template can be used without a kinematic (only with axes), too.
* Table for editing teach points (Recipe/Teachpoints)
* Predefined HMI languages: German, English and Turkish (Settings>General)
* Communication check: HMI is locked if not connected to PLC  
  (reasons: 1. security 2. some features are only working when PLC is running)
* Alarm handling: Active, history and ctrlX CORE diagnostics. (Diagnostics)
* HMI User Manager which allows to add, edit, delete… users (Settings>User Manager)
* Responsive design (Min. requested resolution tablet)
* PLC example code moving between 3 different points   
  (Pos-Home, Pos-Mold-Up, Pos-User-10)

See FB\_Machine\_Customer.mAutoPlc()

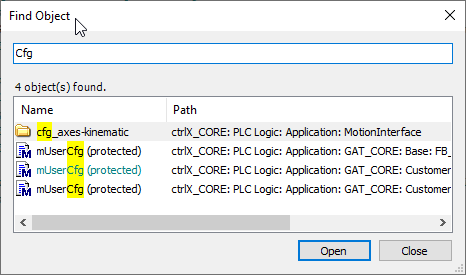
* PLC example code to manage Python interpreter (create instance, start script, …)  
  See FB\_Machine\_Customer.mAutoPython()
* Recipe handling on PLC with Open, Save, SaveAs and Delete in mode Manual & Setup(Recipe) (PLC recipe structure stRecipe)
* 3D viewer showing the moving kinematics
* IDE Coding in iframe
* Oscilloscope and chart graphics

# Features in detail

The following describes how to use some specially implemented features in more detail.

There is a powerful feature in PLC Engineering, how you can find any item in the PLC project tree, by the following steps:

* Click anywhere in the project tree, to active the window
* As soon, as you type the first character, a window with matching elements is opening



## Choose method for power on via fieldbus

When real ctrlX DRIVEs are connected to the fieldbus the DC bus can be loaded and drives status changed from bb to Ab via the fieldbus. Therefore, example programs in the mCyclic\_Active of the FB\_Machine\_Customer can be chosen via variable iPowerOpt. Each is representing a special hardware topology.

* mPower1  
  Power supply must be a single axis XCS and two more slaves must be attached
* mPower2  
  Power supply must be a double axis XCD and one more slave must be attached
* mPower3  
  Power supply must be a single axis XCS and 6 more slaves must be attached
* mPower4  
  Power supply must be a single axis XCS and 3 more slaves must be attached

## Adapt axis in ref signal

In all mPower methods the checking if axes are in ref is implemented. For virtual axes it will automatically be set to true. As for real axes the information is missing for now in the PLC, customer will have to extend the cyclic data of all axes with parameter S-0-0403 and use each Bit 0 in the code.

*//Check if each single axis is in reference  
arAxExt\_gb[AxisX.AxisNo].InRef :=* ***FALSE*** *OR\_ELSE NOT bRealAxis;* ***// Exchange FALSE to S-0-0403 Bit0 of X\_Axis***

## Adapt axis & kinematic configuration

To get familiar with the template, you should take the axis settings. But later you might have to add axes or change their names.

**Hint**: When you change axis names, you must adapt them in 3D viewer model as well.

The Init procedure of the PLC program verifies if the Motion configuration matches the PLC. If a configuration error is detected, the HMI shows an information. To run the program, you must fix the error first. For example, you have defined an axis name that does not exist.

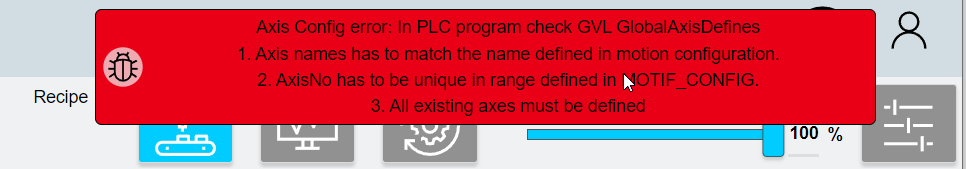


Figure : Config error

When a config error happens, check the following:

|  |  |
| --- | --- |
| **Axis configuration** | **Kinematic configuration** |
| 1. AxisName must match the name defined in motion configuration.  2. AxisNo must be unique in range defined in MOTIF\_CONFIG.  3. All existing axes must be defined here | 1. KinName must match the name defined in motion configuration.  2. GroupNo must be unique in range defined in MOTIF\_CONFIG.  3. Axes assigned to kinematic must exist |

After fixing the problem, do a **complete download** of the PLC program and **reload** the HMI in the browser.

|  |  |
| --- | --- |
| Config Error: | TE\_AxisInterfaceMainProg |
| Definition: | MotionInterface/cfg\_axes-kinematic/GlobalAxisDefines/ |
|  | |

Figure : Config error

|  |  |
| --- | --- |
| Config Error: | TE\_KinInterfaceMainProg |
| Definition: | MotionInterface/cfg\_axes-kinematic/GlobalKinematicsDefines/ |
|  | |

Figure : Config error

## Adapt axis and kinematic limits

In FB\_Machine\_Customer/00\_Init/mUserCfg settings for axes and kinematics can be adapted to application needs. E.g., Set max. jog velocity for each axis is set to 10% of the max. axis velocity.

arAxExt\_gb[uiAxNo].Lim.lrVelJogMax :=   
 MIN(arAxExt\_gb [uiAxNo].Lim.lrVelPos, arAxExt\_gb [uiAxNo].Lim.lrVelNeg) \* 0.1;

**Hint**: Instead of the library **CXA\_MotionInterfaceUser** the template now uses the internal structure arAxExt\_gb[]. This avoids problems with updated libraries of new ctrlX PLC Engineering versions.

## Configure kinematic switches

In FB\_Machine\_Customer/00\_Init/mUserCfg switches can be configured. These switches are displayed in the jog widget of the HMI, e.g., to activate compressed air or gripper in OP mode Manual. To display a switch set a reference to a BOOL variable and define the label for the HMI button.

## Configure common features

PLC: Open **GVL\_CFG** in Plc engineering to configure PLC features. Here some examples:

|  |  |
| --- | --- |
| Flag | Description |
| bDisplayAxesPos | Display axes position instead of kinematic position, while kinematic position is invalid, when not grouped |
| bTeachPointEditInAutoMode | Allow/Disable editing teachpoints in OP mode automatic |
| bHoldJogButton | jog button must be hold until distance reached |

WebIQ: Open local script **cfg** in Code Manger of WebIQ Designer. Here some examples:

|  |  |
| --- | --- |
| Flag | Description |
| bCfgHideHelp | Hide help icon if no help defined |
| bCfgHideFullscreen | Hide toggle button for fullscreen |
| bCfgHideCtrlxName | Hide name of ctrlX CORE |
| bCfgHideVirtualKeypad | Hide virtual keypad (e.g. when keyboard is plugged) |
| iCfgHeartbeatInterval | To disable heartbeat connection check for debugging set interval = 0. |
| URL\_3D\_VIEWER | Url to display 3D viewer |
| URL\_TIGER | Url to display IDE |
| arTabUserGroupsShow cssHideTabs | Now the WebIQ feature to hide a tabulator dependent on a condition is not implemented. You can use these two variables to configure which tabs are hidden for certain user groups. |
| arIdeUserGroupsShow  cssHideIdeCtrls | Now there is no feature to hide specific controls in IDE app. You can use these two variables to configure which controls are hidden for certain user groups. |

## PLC state machine

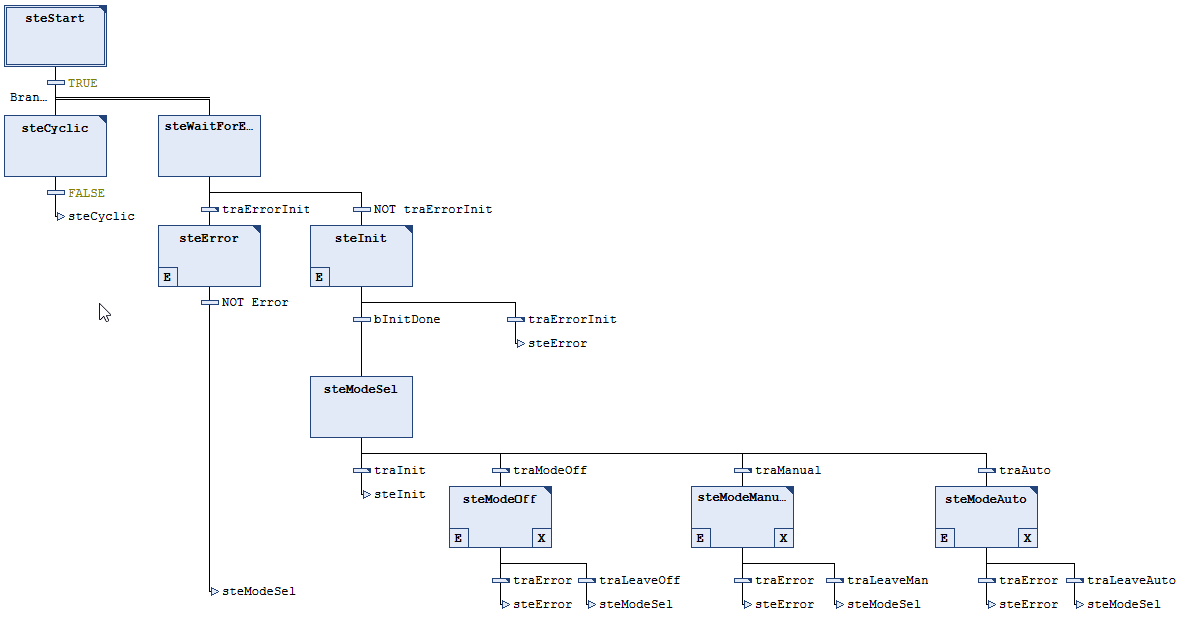
At the beginning the GAT compact status machine was copied 1:1 from the MLC. Below you see the logic of the state machine.

Figure : PLC state machine

Since the original graphical implementation could not separate basis from user code, we changed to Structured Text. This results in the following advantages:

* Separation of base and user code. => Easy version update
* Entry and exit methods can be active for several PLC cycles instead of only 1 cycle before.

Although the inheritance of object orientation is used here, the required knowledge to use it, is very basic. All necessary informations are included as a comment in the source code. All entry, active and exit methods have the same structure.

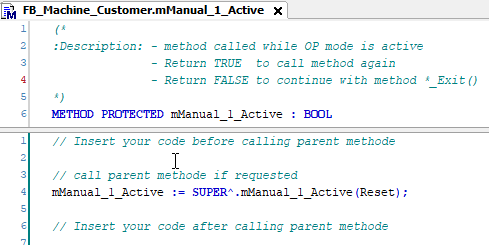


Figure : PLC example mManual\_1\_Active

## Code generation for inputs/outputs

The template includes a mechanism that allows forcing of inputs and outputs via the HMI. The code required for this feature is generated by a Python script in PLC Engineering. No additional work, like adding OPC UA variables or HMI screens, is required. So, you can save a lot of implementation time.

By default, digital IO is automatically detected, and the code is generated. Alternatively, you can still manually export the digital IO configuration.

You must accomplish these 5 steps to use this mechanism:

Step 1: Copy python code into a file e.g., makeCode\_ForceIo.py (only once)

Step 2: Definition of the variables for the digital inputs/outputs

Step 3: **Only manual export:** Export every single digital IO into a separate csv file

Step 4: Start the Python script

Step 5: Adapt **bDeadmenSwitch** in **MachinePrg** (only once)

**Step 1: Copy python code into a file**

Creating the script file must be done only once. Afterwards the script can be used again.

* Open the POU **makeCode\_ForceIo** in PLC engineering
* Copy complete python code into a text file (e.g., makeCode\_ForceIo.py)
* Optional:
  + Set variable **iMapAllIOs**.   
    0=do not map IOs without name   
    1=map all IOs even the unnamed IOs. When no IO name is defined the bit address is used as IO name. e.g., IX50\_0

Manual export of CSV files:

* To enable manual export in script: Set variable **iAutoDetectIO**=0
* Set the variable **strCsvDir** to the directory where you saved the CSV files. Therefor no dialog asks you to select the files.

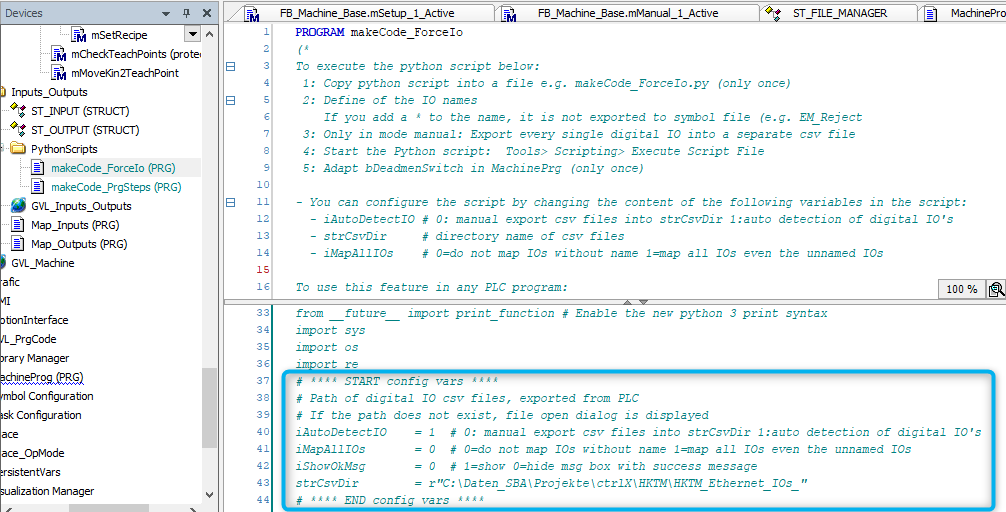


Figure : PLC engineering Python script

**Step 2****: Definition of the variables** **for the digital inputs/outputs**

Open the configuration mapping page. Enter the variable name of inputs/outputs into the description field of the input/output page. Keep in mind to create valid PLC variable names. If you add a \* to the name (e.g., EM\_Reject\*), it is not exported to the symbol file.

The column Variable must not be filled in. This prevents writing to the IOs via 2 different paths.

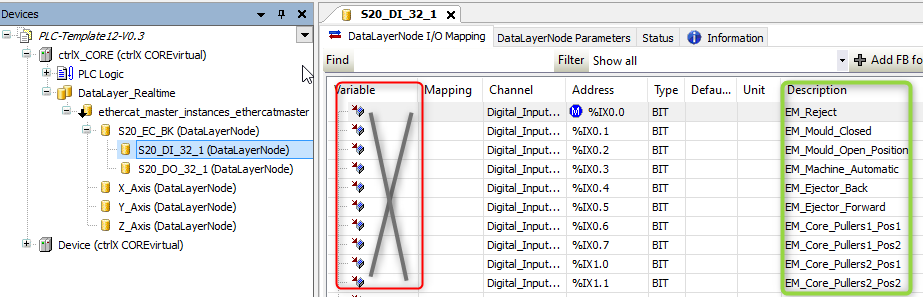


Figure : Names of I/O

**Step 3: Only manual export: Export every single digital IO into a separate csv file**

This step is not necessary in default mode of script, where the digital IOs are automatically recognized.

Right click digital IO, select Export mappings to CSV… and provide a valid file name.

|  |  |
| --- | --- |
| Figure : Digital IO nodes | Hint: All csv files must be in the same folder  Do not export any other module than digital IO’s. |

**Step 4: Start the Python script**

In PLC Engineering

* Click on Tools/Scripting/Execute Script File

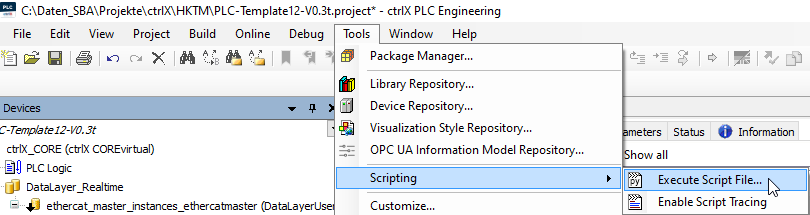


Figure : Execute script file

* Start the python script, by selecting it within the file browser
* Manual export only:   
  If you haven’t set **strCsvDir**, select any of the exported csv files and confirm with Open
* In the message window, check whether the code generation was successful and which DUTs, GVLs and POUs were updated.
* Compile the program. When compile errors are thrown, fix the invalid variable names. Start creation process again.

**Step 5: Adapt bDeadmenSwitch in MachinePrg**

Since not every machine uses a dead men switch or the signal is taken from a any source, you must define the source of the signal in MachinePrg().

|  |
| --- |
| // activate, when there is NO deadman input  //bDeadmenSwitch := TRUE;  // activite, when there is a deadman digital input  bDeadmenSwitch := ix.Deadmen\_Switch; |

## Display active step of PLC automatic program in HMI

The template includes a mechanism to display the program steps in a table. The current step is highlighted. The displayed text is extracted from the ST program code.

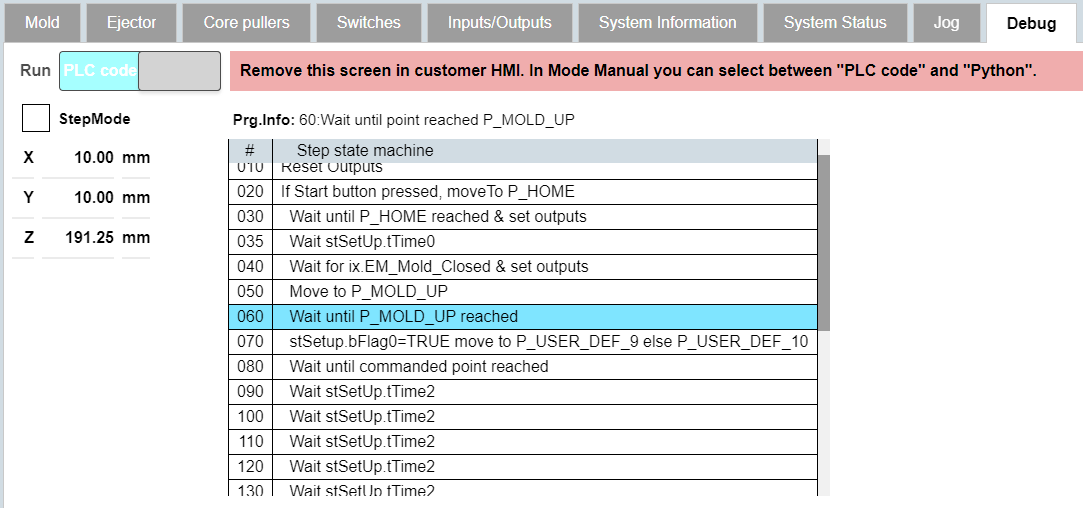


Figure : Display program steps

You must accomplish these 5 steps to use this mechanism:

Step 1: Copy python code into a file e.g., makeCode\_PrgSteps.py (only once)

Step 2: Define a text for every main step of state machine

Step 3: Start the Python script

Step 4: Login with Download

**Step 1: Copy python code into a file**

Creating the script file must be done only once. Afterwards the script can be used again.

* Open the POU **makeCode\_PrgSteps** in PLC engineering
* Copy complete python code and into a text file (e.g., makeCode\_PrgSteps.py)

**Step 2: Define the text for every main step of state machine**

* Open the POU **FB\_Machine\_Customer\_EuroMap.mAutoPlc**
* Add you comment starting with **//PRG** in the same line as the **CASE** value of **iAutoStep**

|  |
| --- |
|  |

Figure : Edit program steps

**Step 3: Start the Python script**

See chapter 7.8 Code generation for inputs/outputs Step 4: Start the Python script

The script generates some variables in **GVL\_PrgCode**

**Step 4: Login with Download**

“Login with Online change” is not sufficient to update the information on HMI.

## Virtual keypad

The template uses a virtual keypad from WebIQ package cx-tools. Only the widget **cx-input**, uses this keypad. It has the following advantages:

* It can be enabled/disabled at runtime (Settings> General Settings)
* Min./Max. Limits can use variables which are calculated in PLC at runtime
* Max. input length for text
* Numpad or alphanumeric keypad automatically displayed, dependent on item type
* Easy configuration of additional languages

## Configuration of alphanumerical keypad

Features of alphanumerical keypad:

* Letter page (**char**)
  + Each row contains **lower** and **upper**
  + Do not define more than 5 rows, to avoid small keys
  + Characters are separated by at least one space
* Number special characters page (**number**)
  + Same setup as **char**
  + Can be empty when not used/displayed
* Normal and password input
* Function keys

|  |  |
| --- | --- |
| #cancel cancel key  #ok     ok key  #|< move cursor to start  #>| move cursor to end  #<<     move cursor to previous word  #>>     move cursor to next word  #<       move cursor to previous char  #> move cursor to next char | #abc display keypad character page  #123 display keypad number page  #shift shift key  #<del delete key  #clear clear complete input  #space space key  #+-     +/- key |

* Predefined languages (en/de) can be overwritten by adding **cxt.keypadAddLanguage("de", myDE)** in local script **start-up**

The keys of the virtual keypad are defined within a json object.

|  |
| --- |
| myDE: {      char: [          {              lower: "° 1 2 3 4 5 6 7 8 9 0 ß",              upper: '^ ! " § $ % & / ( ) = ?'          },  // you can add more array items (=keypad rows)          {              lower: "#shift < y x c v b n m , . -",              upper: "#shift > Y X C V B N M ; : \_"          }      ],      number: [  // same format as section char      ],      header: { lower: "#|< #<< #< #clear #> #>> #>|" },      footer: { lower: "#cancel #abc #space #123 #ok" }  }, |

## Help documentation

The template support displaying help information in multiple languages for all screens (Click “?” in header of template). If help of active language is not available, the help of default language is displayed, if available. (button can be hidden see chapter 7.6 Configure common features)

Naming convention**: help<main-item>-<sub-item>.<ext>**   
e.g. main-item=0 sub-item=0 => help0-0.htm or help0-0.pdf  
When both files exists, the pdf file is displayed.

Folder: **%appdata%\webiq-designer\workspace\docs\help\<LanguageFolder>**

The **<LanguageFolder>** strings, and the default language are defined in local script **config.js**

The default data directory name **webiq-designer** used in the example, must be adapted, when you configured a different data directory. You can verify your data directory in the “About WebIQ Designer” screen.

## Charts

To display charts in WebIQ, the library **plotly.js** is used. The following features are implemented:

* Data creation in PLC (number of items variable at runtime)
* Chart updates when data changes

See remarks in PLC example **PRG\_Chart\_Demo()** & WebIQ screen **Production>Charts** for more details.

## Oscilloscope

WebIQ contains a trend element which allows to show trend charts. The smallest recording time is limited by the smallest communication update time. Therefore, with this mechanism you can’t display data which changes in a few msec or even less. With this example you get an oscilloscope functionality, where the PLC program:

* Records data (in every PLC cycle)
* Transfers configuration & data via CSV & JSON strings
* Sends it via OpcUA to WebIQ.
* Communication is like a REST API, via command & status strings

In WebIQ the plotly.js library is used to display the data.

See remarks in PLC example **PRG\_Osci\_Demo()**, WebIQ screen **Diagnostic>Oscilloscope** for more details (e.g. axes titles, name of csv file).

## Teach point table configuration

The package cxVisuals, contains the widget **cx-table**, which is used to display and edit the teach points. This table is controlled by **FB\_TeachPointMgr** which extends **FB\_TableMgr**. The features are:

* Display/edit any PLC structure array
* Filter/Sort/Clear table columns
* Copy/Paste table rows
* Format table cells via JS functions defined in WebIQ (e.g. 1 => 001)

The FB provides a manager for arbitrary PLC array of structures. To use the FB for a specific structure, the following must be observed:

* FB\_TeachPointMgr

|  |
| --- |
| // Init struct array used for table  mTable\_Init(IdFirst:=TABLE\_ID\_MIN, IdLast:=TABLE\_ID\_MAX, Size:=SIZEOF(arTeachPoints[TABLE\_ID\_MIN]));  // Init custom specific data used anywhere in the FB  \_Robot := Robot;  // Call table state machine  SUPER^.FB\_Main(); |

* Methods in folder **Adapt** must be implemented to fit used array of structure
* See description & source code of the methods for usage and examples
* **mDefineColumns()**  
  Define table columns to display with title, alignment, editable,…
* **mClearAllRows()**

Overwrite all struct array elements with default values defined here

## Alarm/Warning/Info Management

We have implemented a multiplexing channel for alarm/warning/info handling with the following features:

* The number of different alarms/warnings/info’s is unlimited (e.g., 1000, 4000, or more)
* Up to 30 alarms can be displayed simultaneously (method mAlarmSet())
* Up to 10 warnings can be displayed simultaneously (method mWarnSet())
* Up to 10 information’s can be displayed simultaneously (method mInfoSet())
* With increasing number of alarms/warnings/info’s, there is no need to add further communication variables
* If the translation not defined in WebIQ, only the error code is displayed

When you do not like this behavior, you can use the default implementation of WebIQ where a single variable for every message item is used.

Alarm IDs starts at 1000. This ensures a correct numeric sort in the alphabetically sorted Localization Manager list of WebIQ. This section shows the steps how to create a new error.

1. PLC: In EN\_APP\_ALARM add a new error enumeration item  
   e.g. APE\_APP\_SECURITY\_DOOR\_OPEN := 1015
2. PLC: To raise the error call method FB\_Machine\_Base.Tools.mAlarmSet()*.*  
   Beside the error enum you can provide 3 optional strings, which are displayed in HMI.

IF bDoorOpen THEN  
 mAlarmSet(enAppError:= APE\_APP\_SECURITY\_DOOR\_OPEN, '', '', '');

END\_IF

1. WebIQ: In Localization Manager create variable alm1015 if not exists and enter the alarm text for all languages.

## Debug> Plc/Python

For demonstration purpose the PLC template can either run the PLC method or any python script. Select mode Manual, to enable the execution of PLC code or python script.

The plc state machine contains a step mode for testing the program. It can be switched on/off while the program is running.

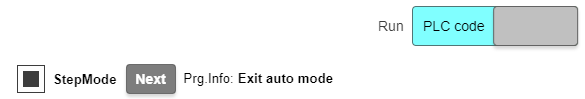


Figure : Debug screen

Python scripts can move axes and kinematics and read/write data layer variables. Also, the teach points of this PLC template. An example how to do this, will be shipped in a later version.

The example script only reads/writes variables from the data layer, which are shown in WebIQ HMI:

To use demo HMI page for python script, do the following steps:

* Install IDE & python app on ctrlX CORE
* The python scripts are installed. See section 4.1 ctrlX CORE
  + - **DemoDataLayer.py** Read & write of data layer variables, stop script
    - **DemoMoveKin.py** Move kinematic & write data layer variables

**In WebIQ HMI:**

* Select mode Manual or Setup
* Open page **Recipe/Setup**
  + Use button **Text** to open textual programs
  + Use button **Visual** to open visual programs
  + Save the recipe
* Open page **Production/Debug** select Run: **Python**
* “Start” the script in the Automatic Mode which changes the fields bFlag and Info

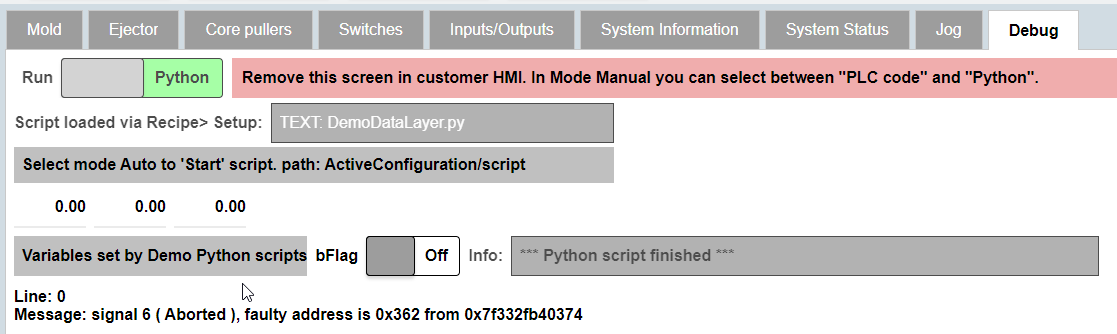


Figure : Debug screen

**Hint**: This screen is only a playground.

It is used to demonstrate the start of Python scripts from the PLC. In a customer project the handling of Python scripts must be implemented customer specific.

## Debug> Plc/Python

On this page some debug information’s are displayed. Especially the table feature is used to display and edit any PLC variable. To add a new variable, enter it in FB\_DebugVarsMgr.mDefineVars(). That is all, nothing to do in WebIQ.

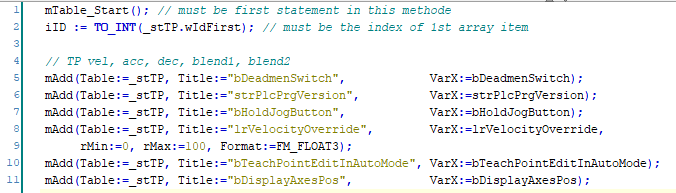


Figure : example mDefineVars()

# WebIQ

## Documentation

WebIQ is a HTML5 based visualization. It works totally different than our previous HMI system WinStudio. If you do not have any HTML5 know how, you should view the videos or read the documentation provided by WebIQ.

**Hint**: Especially the selection of tabulator panels in the Designer (=sub screens) takes some time to get familiar. At least check this video and in the ctrlX AUTOMATION Community.

|  |  |
| --- | --- |
| ctrlX AUTOMATION Community | <https://developer.community.boschrexroth.com/t5/ctrlX-AUTOMATION-Community/ct-p/dcdev_community> |
| WebIQ Forum (questions & answers) | <https://developer.community.boschrexroth.com/t5/Smart-HMI-WebIQ-Designer-and/bd-p/dcdev_community-dcae-smarthmi> |
| WebIQ Documentation | <https://docs.webiq.de/docs/webiq-designer-manual/> |
| WebIQ Videos | <https://www.smart-hmi.de/dokumentation/> |

## Setup development environment (reverse proxy)

The WebIQ template has some features that only work via a reverse proxy (=caddy). Neither of the two designer previews can be used directly to test the project during development. But there is an easy way to test the WebIQ with all features without publishing it every time.

Open article [HowTo-view-content-from-multiple-web-servers-in-one-web-page](https://developer.community.boschrexroth.com/t5/Store-and-How-to/HowTo-view-content-from-multiple-web-servers-in-one-web-page/ba-p/69370) and follow instructions in section “Install and start caddy”. When you installed caddy you can test the WebIQ project by the link <https://localhost:8081/webiq/cx/>

**Hints**:

* After some hours, caddy stops working and it needs to be restarted.
* Enable “Auto Login” in “Hmi Project Settings” of the Designer, to avoid login prompt

## Package Manager

The HMI-Template uses some additional packages, which are included in the project file, for runtime. However, for editing the project, the packages must be installed within the Designer.

* Press (1) to open Package Manager
* Press (2) “Upload package” to select file
* Repeat previous step, for all other packages

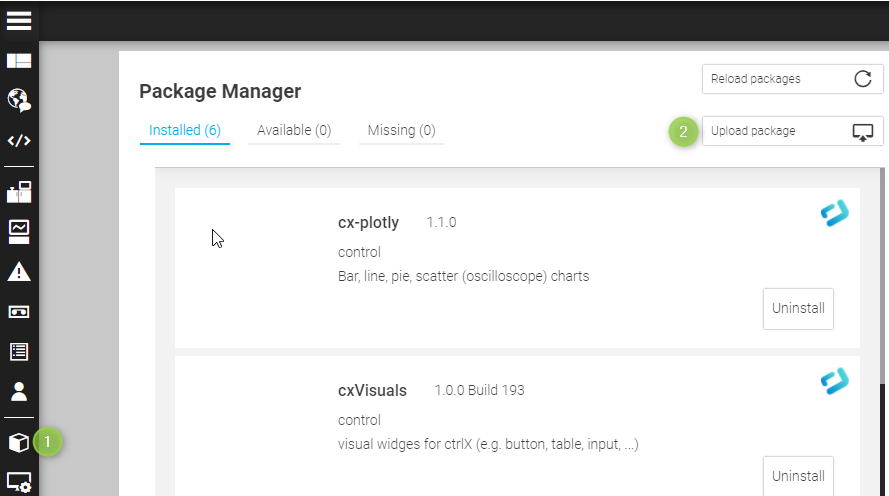


Figure : WebIQ Designer: Package Manager

## Color Themes

The user can choose between different color themes. The template offers a light Color Theme with Bosch Rexroth’s Corporate Identity Colors. In contrast to that also a dark Theme exists. The two other themes can be fully customized in the HMI settings. The functionality for this theme switching is done by custom code, which you can see in the Code Manager.

In the next chapters the different colors of the themes are listed in a table. The variables in the first column are the variables that are set in the globals.js file. To change the color of the icons the color is inverted.

### Color schema 1 Corporate Design Rexroth

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Hex Code** | **Color** | **Usage** |
| **Functional colors:** | | | |
| --col-bg-ok | #78BE20 |  | Messages, Green Leds |
| --col-bg-ok-label | #000000 |  | Label color for ok messages |
| --col-bg-warn | #FCAF17 |  | Used for Warnings |
| --col-bg-warn-label | #000000 |  | Label color for warn messages |
| --col-bg-error | #EA0016 |  | Used for Errors |
| --col-bg-error-label | #000000 |  | Label color for error messages |
| --col-bg-question: | #00CCFF |  | Used for Question Dialog boxes |
| --col-bg-question-label | #000000 |  | Label color for question messages |
| **Button colors:** | | | |
| --col-bg-btn-normal | #7D7D7D |  | Background color of Buttons |
| --col-bg-btn-active | #00CCFF |  | Background color of Buttons in active state |
| --col-btn-normal-border | #DADADA |  | Border color of Buttons |
| --col-btn-active-border | #91F0FF |  | Border color of Buttons in active state |
| --col-btn-normal-label | # FFFFFF |  | Label color of Buttons |
| --col-btn-active-label | #FFFFFF |  | Label color of Buttons in active state |
| --inv-btn-normal-img | 1 |  | Inverts the icons (0=black,1=white) |
| --inv-btn-active-img | 1 |  | Inverts the icons (0=black,1=white) |
| **Background colors:** | | | |
| --col-bg-content-window | #FFFFFF |  | Background color of content window |
| --col-bg-content-box | #EFF1F2 |  | Background color of a content box |
| --col-content-font | #000000 |  | Content font color |
| --inv-content-img | 0 |  | Inverts the icons (0=black,1=white) |
| --col-bg-sidebar | #EFF1F2 |  | Background color for the sidebar |
| --col-sidebar-border | #D1DCE3 |  | Color for sidebar border |
| --col-bg-table-hdr | #D1DCE3 |  | Color of the Header of the teachpoint table |
| --col-bg-table-row-active | #7FE5FF |  | Color of the active row in the teachpoint table |
| **Header colors:** | | | |
| --col-bg-header | #EFF1F2 |  | Background color of the header |
| --col-bg-rexroth-gif | #D1DCE3 |  | Background color of the Rexroth gif |
| --col-bg-header-top | #D1DCE3 |  | Background color of the top header |
| --col-header-label | #000000 |  | Font color of text in the header |
| --inv-header-img | 0 |  | Inverts the icons (0=black,1=white) |
| --col-tab-inactive | #9D9D9D |  | Color for the inactive tabs of the tab-panel |
| --col-tab-inactive-font | #FFFFFF |  | Font color for inactive tabs of the tab-panel |

### Color schema 2 Dark Theme

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Hex Code** | **Color** | **Usage** |
| **Functional colors:** | | | |
| --col-bg-ok | #78BE20 |  | Messages, Green Leds |
| --col-bg-ok-label | #000000 |  | Label color for ok messages |
| --col-bg-warn | #FCAF17 |  | Used for Warnings |
| --col-bg-warn-label | #000000 |  | Label color for warn messages |
| --col-bg-error | #EA0016 |  | Used for Errors |
| --col-bg-error-label | #000000 |  | Label color for error messages |
| --col-bg-question: | #00CCFF |  | Used for Question Dialog boxes |
| --col-bg-question-label | #000000 |  | Label color for question messages |
| **Button colors:** | | | |
| --col-bg-btn-normal | #969696 |  | Background color of Buttons |
| --col-bg-btn-active | #00CCFF |  | Background color of Buttons in active state |
| --col-btn-normal-border | #AAAAAA |  | Border color of Buttons |
| --col-btn-active-border | #91F0FF |  | Border color of Buttons in active state |
| --col-btn-normal-label | # FFFFFF |  | Label color of Buttons |
| --col-btn-active-label | #FFFFFF |  | Label color of Buttons in active state |
| --inv-btn-normal-img | 1 |  | Inverts the icons (0=black,1=white) |
| --inv-btn-active-img | 1 |  | Inverts the icons (0=black,1=white) |
| **Background colors:** | | | |
| --col-bg-content-window | #121212 |  | Background color of content window |
| --col-bg-content-box | #505050 |  | Background color of a content box |
| --col-content-font | #FAFAFA |  | Content font color |
| --inv-content-img | 1 |  | Inverts the icons (0=black,1=white) |
| --col-bg-sidebar | #646464 |  | Background color for the sidebar |
| --col-sidebar-border | #464646 |  | Color for sidebar border |
| --col-bg-table-hdr | #969696 |  | Color of the Header of the teachpoint table |
| --col-bg-table-row-active | #00CCFF |  | Color of the active row in the techpoint table |
| **Header colors:** | | | |
| --col-bg-header | #646464 |  | Background color of the header |
| --col-bg-rexroth-gif | #646464 |  | Background color of rexroth gif |
| --col-bg-header-top | #464646 |  | Background color of the top header |
| --col-header-label | #FAFAFA |  | Font color of text in the header |
| --inv-header-img | 1 |  | Inverts the icons (0=black,1=white) |
| --col-tab-inactive-font | #000000 |  | Font color for inactive tabs of the tab-panel |

## Composites

For easy maintenance most of the composite widgets from previous template version are replaced by custom widgets. The custom widget packages will be available soon in the ctrlX AUTOMATION Community with documentation and examples. While the documentation is not available, just check how the custom widgets are used in the template.