Application Note

FESTO

How to start with the Festo AX Controls Python App

This documents provides a guide to use the Festo AX Controls Python App

GSBE-PY1

100827

Title ..... How to start with the Festo AX Controls Python App
Version 1.10
Document no. 100827
Original ..... en
Author ..... Festo

Last saved 20.04.2026

General information:

This document is intended for qualified, trained and instructed professionals. The data provided in this document are no guaranteed specifications, in particular with regard to functionality, condition or quality in the legal sense. The information in this document is intended only as simple indications for the implementation of a specific, hypothetical application, and in no way as a substitute for the operating instructions of the respective manufacturers or the design and testing of the respective application by the user. The respective operating instructions for Festo products can be found under www.festo.com. The user of this document must ensure that each function described herein works properly in his application. The user remains solely responsible for his or her own use of this document, even by studying this document and using the information mentioned therein. This also applies to any software (in source code and/or object code) that is made available to the user as an appendix to this document.

Rights of use of the software:

If software is made available to the user as an appendix to this document, the user is granted a simple and unlimited right of use hereto. This right also includes the processing and distribution of the software to third parties in edited or unedited form. The User is not permitted to use the name "Festo" to endorse or promote products derived from the Software without express prior written permission.

Software is provided to the user "as is". Festo does not assume any warranty or guarantee with regard to software. Festo's liability for damages of any kind arising from the use of the software is limited to intent and gross negligence.

Legal Notices:

Table of contents

1 Components/Software used

Type/Name	Version Software/Firmware	Date of manufacture
CEPE	25.9.0	05/2025
Festo Python	1.1.1 or later	

Table 1.1: Components/Software used

2 Overview

The Festo Python App provides a Python Runtime to allow programming in a modern scripting language directly on the PLC. It can be used for simple scripting tasks or to enhance the PLC application with functionality written in Python.

The App offers a range of key features designed to enhance the development experience. It provides an online web-based IDE, allowing users to access a fully functional integrated development environment directly through their web browser, enabling easy coding from anywhere.

One of the standout features is the remote development capability, that allows developers to connect a local IDE to the app, enabling them to work directly on applications running on the target device. This connection utilizes a pre-configured reverse tunnel, providing access to the host PC's network and facilitating seamless interaction with network resources as long as the IDE connection is active.

Additionally, the app includes a built-in sample Python project that features example scripts and a customizable main.py script, which runs automatically each time the app starts. This setup allows developers to quickly get started with coding and testing. Additionally, the app comes with a pre-installed Python virtual environment that includes all necessary libraries.

For version control, the app offers built-in Git support, making it easier to manage code and collaborate with others. Users also have the flexibility to install further VS Code extensions and Python libraries if the device is connected to the internet or using the remote development feature.

Furthermore, the app provides a configurable port for optional incoming socket connections, allowing for flexible integration with other services and applications.

The app is designed to make it easy for developers to experiment, try, and test things out directly on the device quickly, without the need for extensive learning or complicated setups. With its intuitive features and streamlined processes, developers can focus on coding and innovation, enhancing their productivity and efficiency.

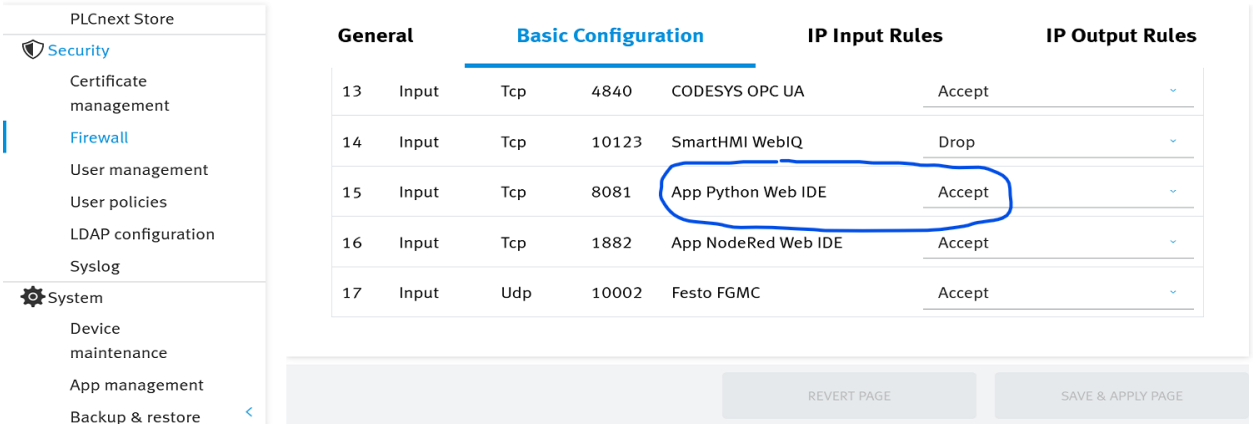
3 Setup and usage

3.1 Using the integrated IDE

3.1.1 Open Integrated IDE

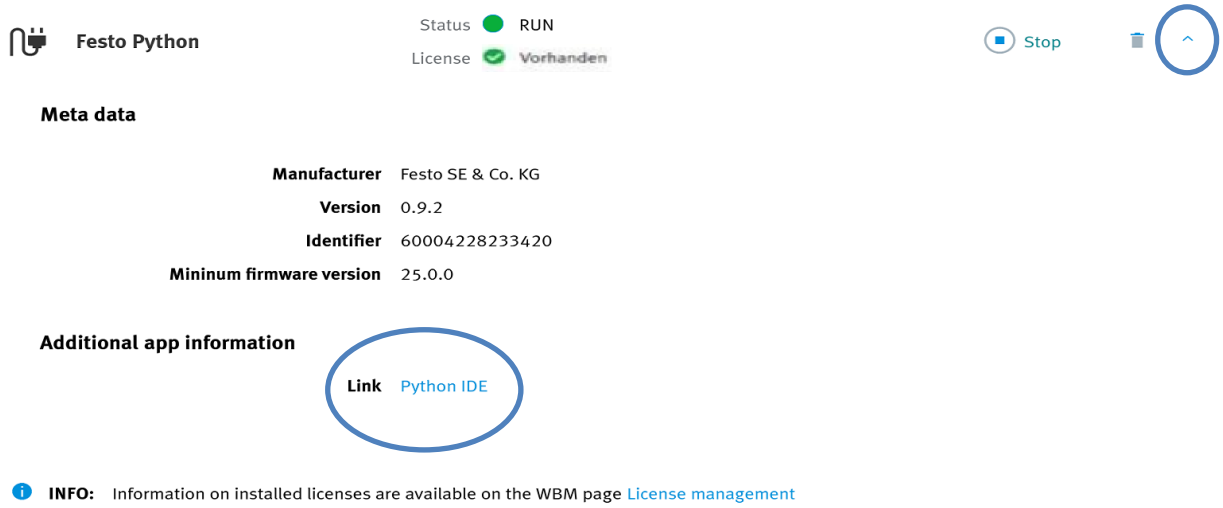
By default the port to access the Integrated IDE is blocked by the firewall of the device-. You need to manually configure it before usage by opening the firewall menu in Web Based Manager (WBM-Device Web-Browser):

- Security -> Firewall -> Basic Configuration
- then set port 8081 – App Python Web IDE to accept



To start the Integrated IDE of the Python App click on the link in the Additional App information area in the App management of the WBM. This information can be displayed by clicking on the little arrow on the right of the App.

Then click on the link to open the Integrated IDE:

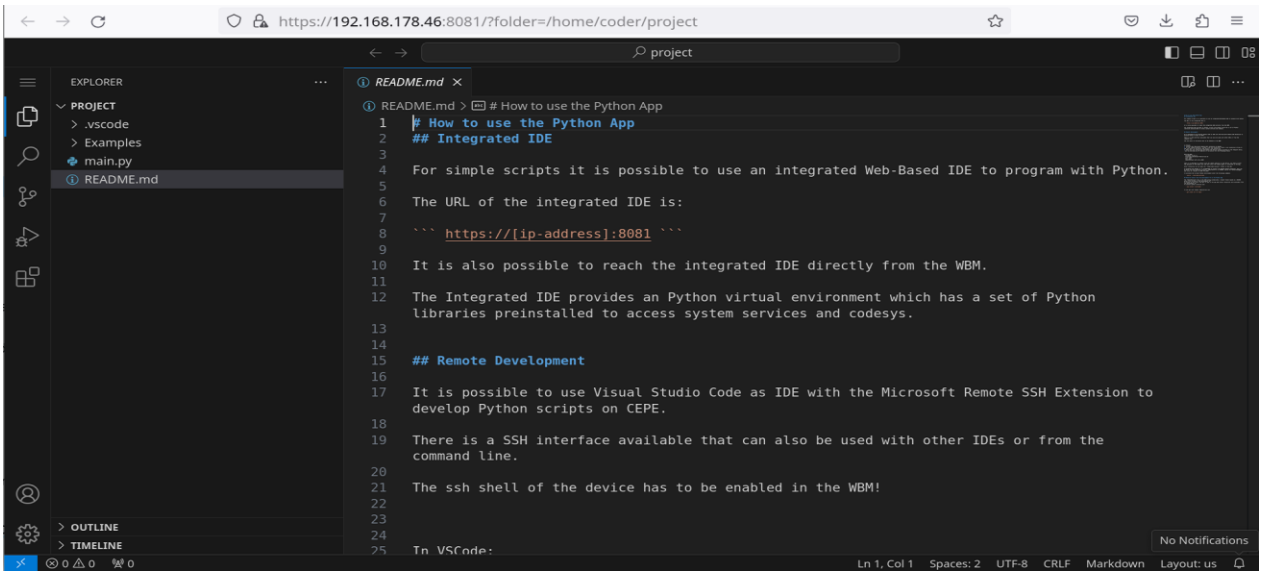


It is also possible to open the Integrated IDE by entering its URL in the address bar of your browser:

```
https://[ip-address]:8081
```

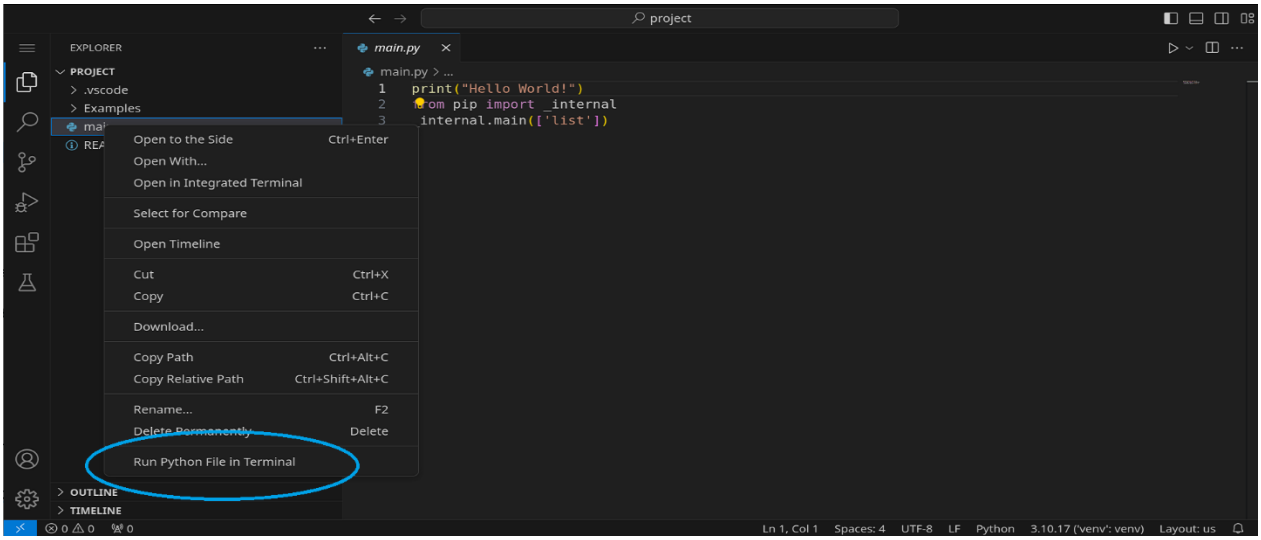
with [ip-address] as the IP address of the device

This opens the Integrated IDE which you can use to develop python scripts. The integrated IDE is configured to open the sample project automatically the first time it starts. To learn more about the sample project please refer to chapter “Sample Project and Examples”.



3.1.2 Start a Python script

The easiest way to run a python script is by a right click on the script and selecting “Run Python File in Terminal”. This way the script gets automatically executed using the preinstalled virtual environment, that has been preconfigured in the in the project settings:



3.1.3 Installing libraries

If the Device is connected to a network that has access to a library repository (e.g. pypi.org) it is possible to just install libraries by using the pip tool in the usual ways.

In case there is no sufficient network connection available it is possible to copy the libraries from a host via drag and drop.

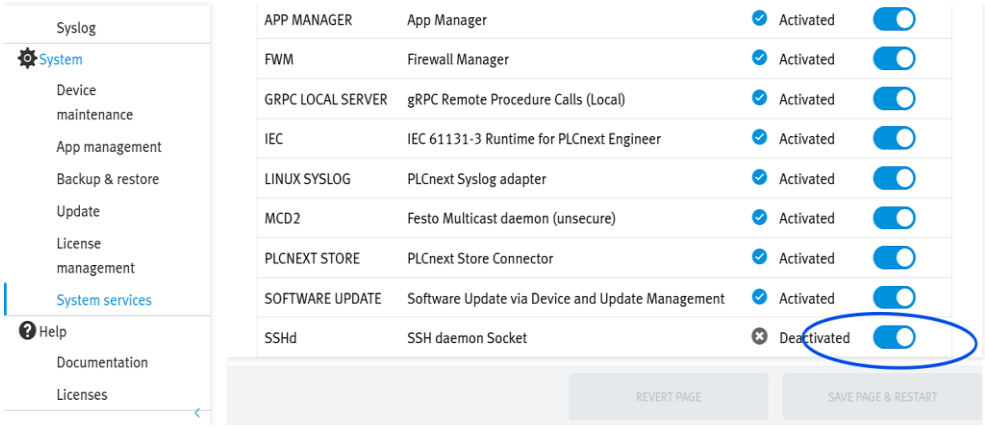
3.2 Remote Development

The Remote development feature, allows developers to connect a local Visual Studio Code (VS Code) instance to the app, enabling them to write, debug, test and run code directly inside the Python App on the target device.

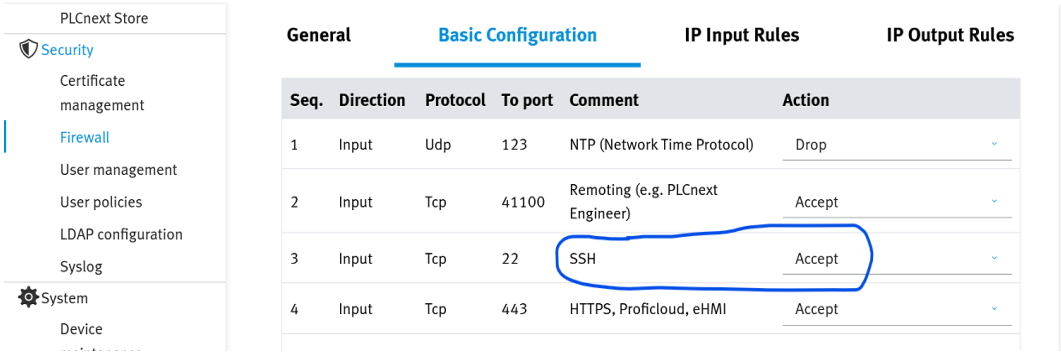
This feature also leverages a reverse tunnel to the host PC, allowing users to interact with the host PC's network infrastructure for internet connectivity or accessing source code repositories on the device while connected, even if the device does not have direct access to this infrastructure.

3.2.1 Connect local IDE to the App

- Make sure ssh connection is enabled on your CEPE
  - System -> System Services -> enable ssh connection

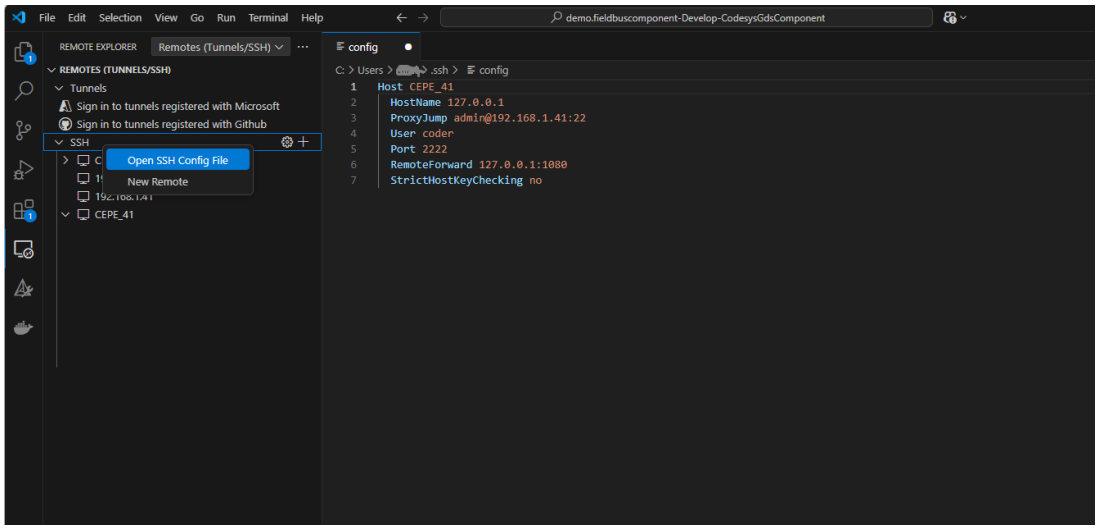


- Allow ssh access in the firewall configuration
  - Security -> Firewall -> Basic Configuration -> set port 22 - SSH to accept



- Open VS Code on your PC
- Install the extension “Remote SSH” from the Extensions side panel
- Open the Remote Explorer from the side bar
- Open your system ssh config by a right click on “REMOTES(Tunnels/SSH)/SSH” and selecting “Open SSH Config File”
- Add the following configuration and save it (replacing the place holder with the ip address of your device):

```
Host MY_CEPE
  HostName 127.0.0.1
  ProxyJump admin@<your-device-ip>:22
  User coder
  Port 2222
  RemoteForward 127.0.0.1:1080
```



- Right click on the newly available MY_CEPE connection in the Remote Explorer and select “Connect in a New Window...”
- A new editor window opens, select “Linux” as the platform of the remote host and enter your credentials each time prompted
- VS Code installs the necessary software and extensions on the target device and after a successful connection MY_CEPE should be green and in the state “connected” in the “Remote Explorer” side bar
- Install the extensions “Python”, “Python Debugger”, optionally “GitLens” and any other extension of your choice from the Extensions side panel while connected. While the connection is active the extensions panel lists the extensions for the host and the device side separately (see lists “Local - Installed” and “SSH: MY_CEPE”). If an extension has been previously already installed on the host side, click on “Install in SSH: MY_CEPE” to install it on the CEPE as well. If a new extension is installed while the connection is active, it gets installed on both the host pc and the CEPE side.
- Open the sample project by clicking on “Open Folder” and selecting the folder “/home/coder/project”
- Edit, Debug and Run python scripts and run VS Code tasks the regular way while connected
- The connection can be closed by clicking on the blue Remote SSH extension symbol on the lower left corner (reads “SSH: MY_CEPE”) and selecting “Close Remote Connection” on the appearing drop down menu or simply by closing the VS Code Window

3.2.2 Using git for version management

The Python App is configured to use a reverse tunnel through the host PC while connected making it possible to use git inside the App to interact with any repositories the host PC has access to. Git can be used per command line in the VS Code Terminal or via any extension of your choice (e.g. GitLens)

3.2.3 Install Python libraries

The Python App is configured to use a reverse tunnel through the host PC while connected making it possible to install any pip packages from pypi or any other repositories the host PC has access to:

- Open a VS Code Terminal
- Activate the virtual environment of the sample project:

  source /home/coder/venv/bin/activate
- Install python packages:

  pip install <python package>

3.3 Connecting with plain SSH

The app allows users to configure and utilize a plain SSH connection to connect directly to it. Through this connection, users can browse the files and folders within the app, facilitating easy navigation and management of data, copy data between the host and the app using SCP or any other preferred method, providing flexibility in data transfer and file handling or use it for any other tasks the user sees fit, enhancing the overall functionality of the app.

3.3.1 Creating an SSH Connection:

- Open the system ssh configuration file (<user home>\.ssh\config)
- Add the following new connection and save it (replacing the place holder with the ip address of your device):

  Host MY_CEPE
  HostName 127.0.0.1
  ProxyJump admin@<your-device-ip>:22
  User coder
  Port 2222
  RemoteForward 127.0.0.1:1080

- Connect to MY_CEPE with your ssh client of your choice or manually by entering in a terminal or command prompt:

  ssh MY_CEPE

3.4 SSH Connection Trouble Shooting

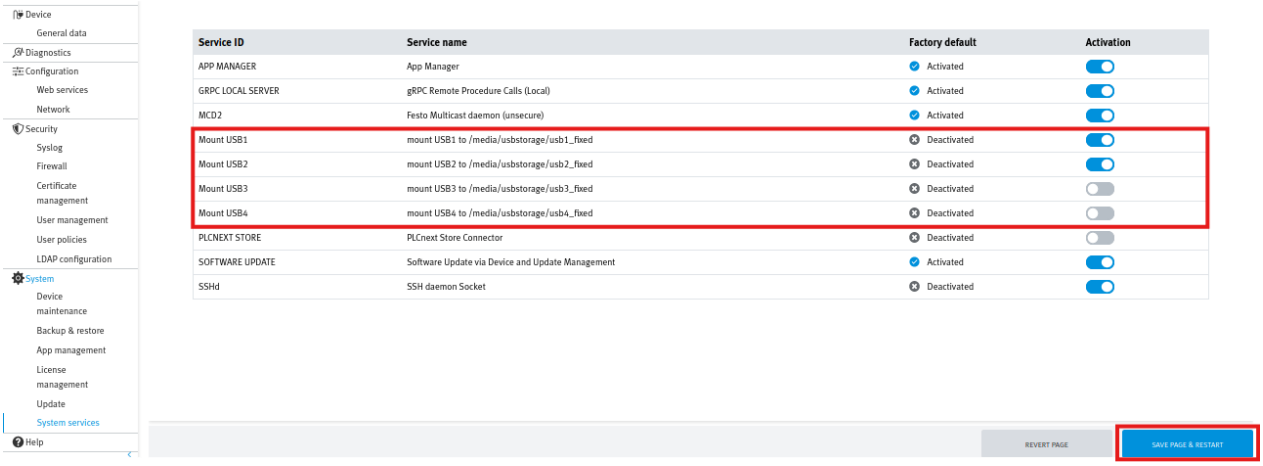
Make sure your PC is connected to the same network as your device or there is a valid network route from the PC to the device. You can list multiple jump hosts in the ssh configuration if needed the following way (make sure to add all the intermediate jump hosts preceding “admin@<your-device-ip>:22”):

- ProxyJump <jump_user_name>@<jumphost-ip>:<port>, admin@<your-device-ip>:22
- Make sure ssh connection is enabled on your device (System -> System Services -> enable ssh connection)
- Remove any host keys previously associated with 127.0.0.1:2222 by entering the following command in a terminal and try to connect again:
- ssh-keygen -f '<your user home folder>/.ssh/known_hosts' -R '[127.0.0.1]:2222'
- Try to manually open an ssh connection to the device by entering “ssh MY_CEPE” in a terminal or command prompt

3.5 Using USB Mass Storage Drives

The app allows users to access USB Mass Storage drives that are connected to the device the following way:

- Make sure the USB storage device is formatted with the format VFAT or FAT32
- Enable the USB mass storage device support on you CEPE
    - System -> System Services -> enable USB mass storage drives



- The USB Storage device will be available inside the python app under the path:
  /media/usbstorage

4 Sample Project and Examples

The app comes with a sample Python project located at the path /home/coder/project. This sample project is specifically designed to be used with the pre-installed virtual environment located at /home/coder/venv. The project is configured to automatically utilize this virtual environment for running and debugging Python scripts, ensuring that all necessary dependencies are available.

Inside the sample project, users will find example scripts that demonstrate how to interact with the available device services using Python. These examples serve as a useful reference for developers looking to leverage the capabilities of the device in their own applications.

4.1 Autostart Script

A key component of the sample project is the main.py script, which is automatically started each time the app launches. This script runs within the pre-installed virtual environment and serves as the entry point for users' programs. Developers are encouraged to modify main.py to suit their specific needs and to build upon the provided examples.

4.2 Stop Task

Additionally, the project includes a predefined task that can stop main.py if it is still running in the background. It is important to note that the debug task is set up to automatically execute this stop task before initiating the debugging process. If this behaviour is not desired, users can easily edit the debug task to remove this feature.

To manually trigger the stop task open the menu, click on Terminal, select “Run Task...” end select “StopBackgroundScript” from the drop down menu.

4.3 Using the Terminal

Users also have the flexibility to run scripts manually or install additional libraries using the terminal. However, it is essential to manually activate the virtual environment beforehand to ensure that the correct dependencies are used:

- Open the menu, click on “Terminal” and select "New Terminal”
- Activate the virtual environment of the sample project:
  source /home/coder/venv/bin/activate
- Install python packages:
  pip install <python package>
- Execute python scripts:
  python main.py

4.4 Examples

The sample project comes with examples demonstrating how to effectively communicate with the device using its built-in services. The device features a Codesys OPC UA server, which facilitates communication between the Python application and the Codesys program, enabling seamless data exchange and control.

Additionally, the device is equipped with a gRPC interface that exposes a variety of internal services. Detailed documentation for interfaces can be found on the following website . To utilize this gRPC interface within Python, the examples leverage a library located in the pxc_grpc folder. This library was generated from the gRPC interface .proto files, which are available in the following official GitHub repository.

These examples aim to provide developers with practical insights into using the device's services, making it easier to integrate and interact with the device in their applications.

4.4.1 Reading System Information

The *grpc_read_system_info.py* example demonstrates how to read system information, device status, and diagnostic information over the gRPC interface. This script provides a straightforward way to retrieve essential data about the device, allowing developers to monitor its operational state and diagnose any issues effectively.

4.4.2 Sending Device Notifications

The *grpc_send_device_notification.py* example illustrates how to utilize the device's notification system via gRPC. This example shows how notifications are sent and logged, which can then be accessed through the device's user interface. It enables developers to implement real-time alerts and notifications in their applications, enhancing the overall user experience.

4.4.3 Checking License Status

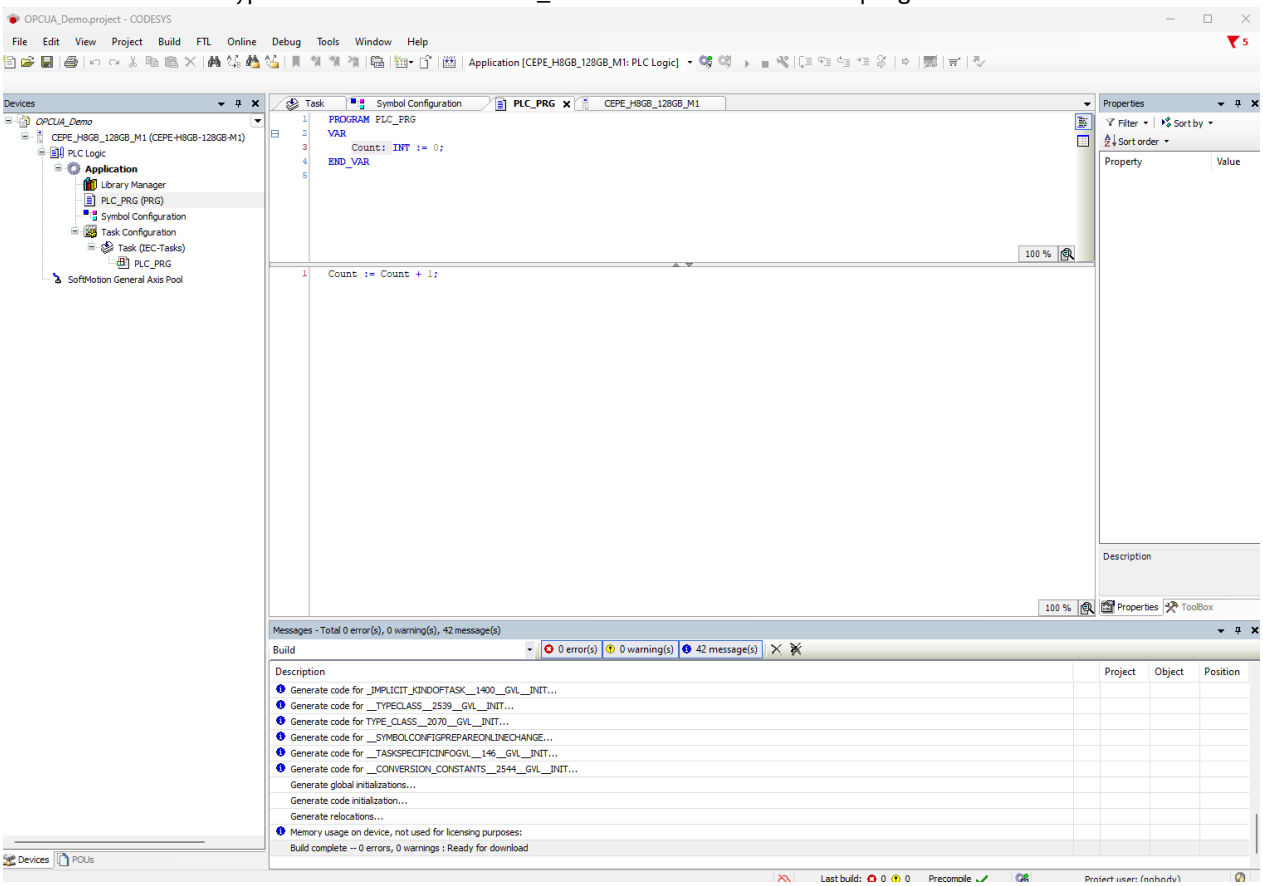
The *grpc_check_license.py* example offers a simple demonstration of how to communicate with the built-in Wibu license manager over gRPC using Python. This script allows users to verify license status and manage licensing information, ensuring that the application complies with licensing requirements.

4.4.4 Accessing the Codesys OPC UA Server

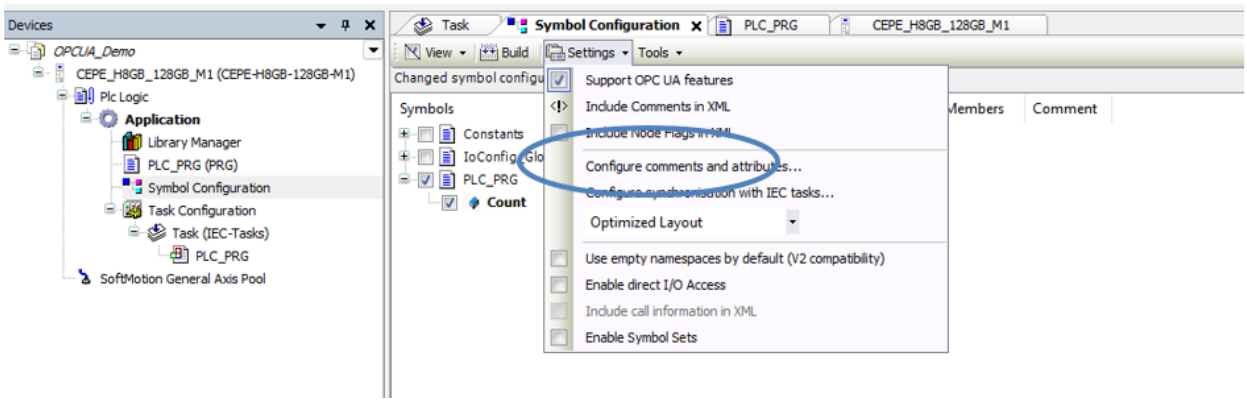
The *codesys_opcua_access.py* example showcases how to connect to the Codesys OPC UA server and read or write values using Python, while also highlighting how the Codesys naming convention translates to OPC UA addresses. To run this project, users need to follow these steps:

1. Create a CEPE Codesys project with a program POU PLC_PRG implemented in structured text

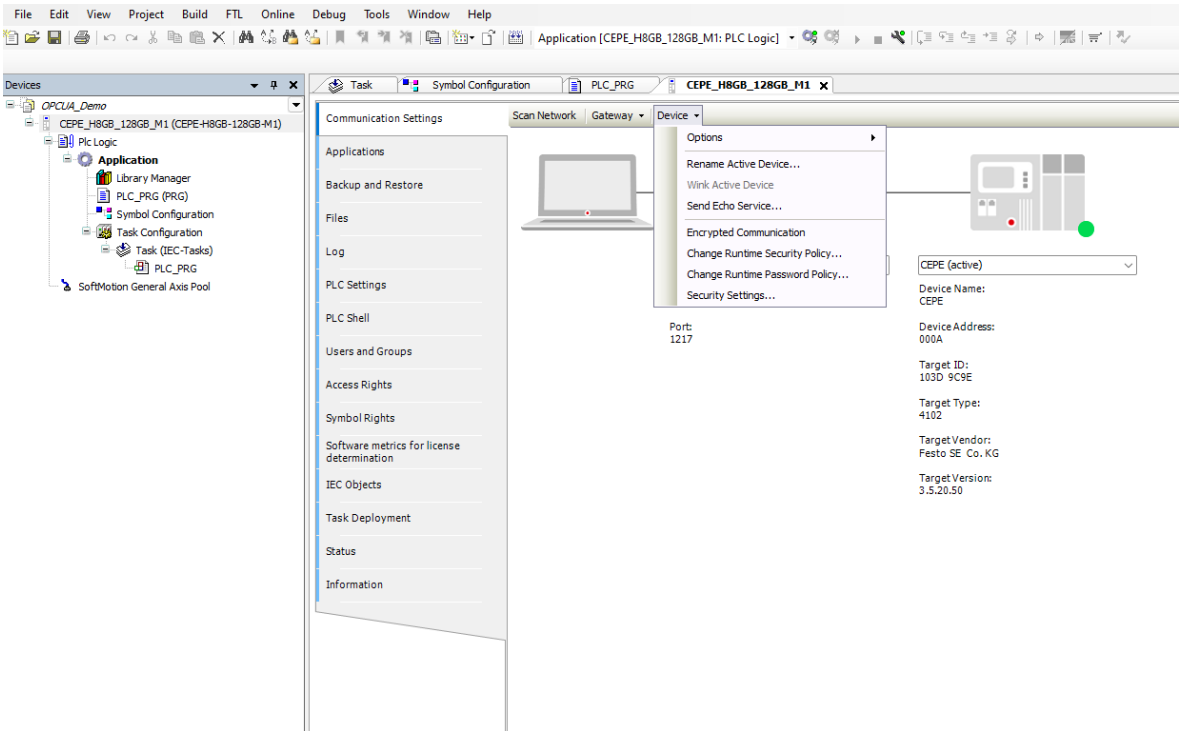
2. Define a variable of type INT called Count in PLC_PRG and increment it in the program



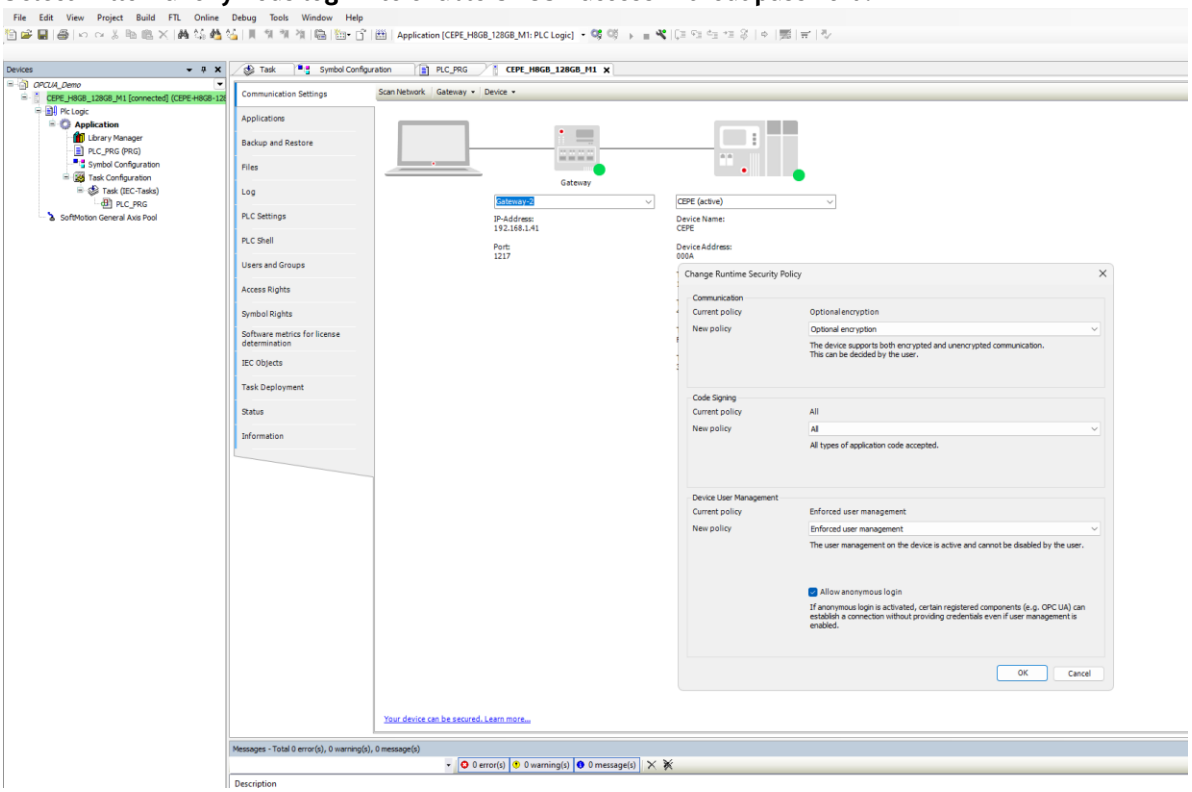
3. To communicate with the Codesys App via OPCUA it needs to be enabled in the the Codesys Application.
The simplest approach is to add a “Symbol Configuration” object to the Application.
In the editor tab of the this object you can now select the variables to export over OPCUA (here: “Count”.
Ensure also that OPCUA is enabled in the “Settings” menu.



4. By default the OPCUA server does not have a user and does not allow anonymous login. For demo projects you can enable anonymous login in the following configuration dialog:
   Device -> Communication Settings -> Device -> Change Runtime Security Policy



Select “Allow anonymous login” to enable OPCUA access without password.



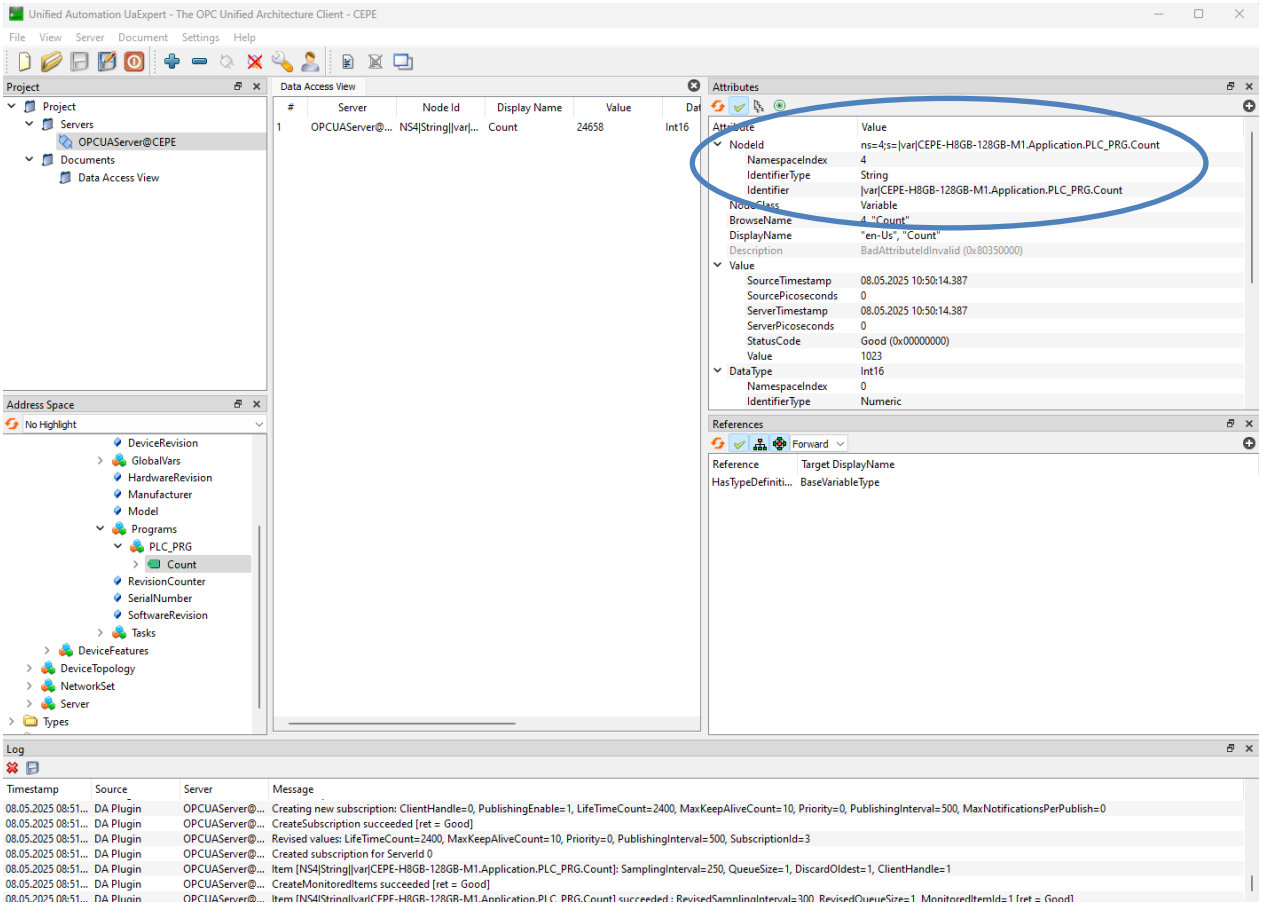
Note

Do not use this for production setups! Please refer to the Codesys documentation for instruction to setup a secure OPCUA Server

5. Deploy and start the Codesys program.
6. Execute the codesys_opcua_access.py python example script to interact with Codesys. The python example script will read the value of the Count variable and then modify it with a new value, demonstrating how to effectively manage data within the Codesys environment using python.

4.4.5 How to test OPCUA

To test the operation of OPCUA you can use the tool “UaExpert” provided by Unified Automation for free.



4.4.6 How Codesys variables are addressed from OPCUA

OPCUA presents its available variables in a browsable tree structure. This is called a OPCUA address space and it is standardized how the variables of a PLC are organized. This tree can be browsed in UaExpert in the tree view on the left.

The variables can also get directly accessed by a **NodeId**. This is a unique string (or number) that references a specific variable. This is used the examples to read and write variable

For Codesys the following rules are used to create the NodeId:

`ns=4;s=|var|CEPE-H8GB-128GB-M1.Application.PLC_PRG.Count`

|-- first part defines the address space and type of NodeId

|------ Name of the PLC

|------ string describes hierarchy of objects in Codesys

5 Install additional libraries

Sometimes it is necessary to install additional libraries or other debian packages into the app e.g. as a prerequisite of Python libraries.

5.1 Persistence in the Python App

Only the /home/coder directory is persistent across device and app restarts. All other changes to the system are lost on restarts. The custom startup script allows you to recreate necessary changes automatically at each startup

5.2 Custom Startup Script

1. Create an executable script at /home/coder/custom_startup.sh:
   ```
   touch /home/coder/custom_startup.sh
   chmod +x /home/coder/custom_startup.sh
   ```

2. Example script:
   ```
   #!/bin/bash
   echo "Starting custom configuration..."

   # Install packages
   sudo -E apt-get update sudo -E apt-get install -y your-package-name

   # Install custom libraries
   sudo -E cp /home/coder/mylib.so /usr/lib/

   # Set environment variables
   export MY_VAR="value"
   echo "export MY_VAR=value" >> ~/.bashrc

   echo "Custom configuration completed."
   ```

5.3 Using Sudo Privileges

- Use sudo -E to preserve your environment variables when executing commands
- Passwordless sudo is only configured for the container user
- Example:
  ```
  sudo -E apt-get install -y build-essential
  ```

5.4 Installing Packages With and Without Internet

For environments without continuous internet access:

1. Download packages with dependencies while the CEPE is connected or connect to the Python App over ssh with the configured reverse tunnel (see recommended ssh configuration in this Application Note) and use the internet connection of your development PC:
   ```
   sudo -E apt-get download $(apt-cache depends --recurse --no-recommends --no-suggests \
    --no-conflicts --no-breaks --no-replaces --no-enhances libusb-1.0-0 usbutils libgomp1 \
    gcc g++ make unzip wget | grep "^\w" | sort -u)
   ```
2. Store the .deb files for example in /home/coder/my_packages
3. Add to your startup script:
   ```
   # Install downloaded packages
   sudo -E dpkg -i /home/coder/my_packages/*.deb
   ```

5.5 Best Practices

- Include error handling in your startup script
- Keep the script efficient to avoid startup delays
- Make your script idempotent (can run multiple times safely)
- Store configuration files, scripts, and custom applications in /home/coder

5.6 Troubleshooting

If your script isn't executing, check:

- Correct path: /home/coder/custom_startup.sh
- Execution permissions: chmod +x /home/coder/custom_startup.sh
- Test your script by running it manually inside the Python App in a terminal session window

6 Working with USB Devices

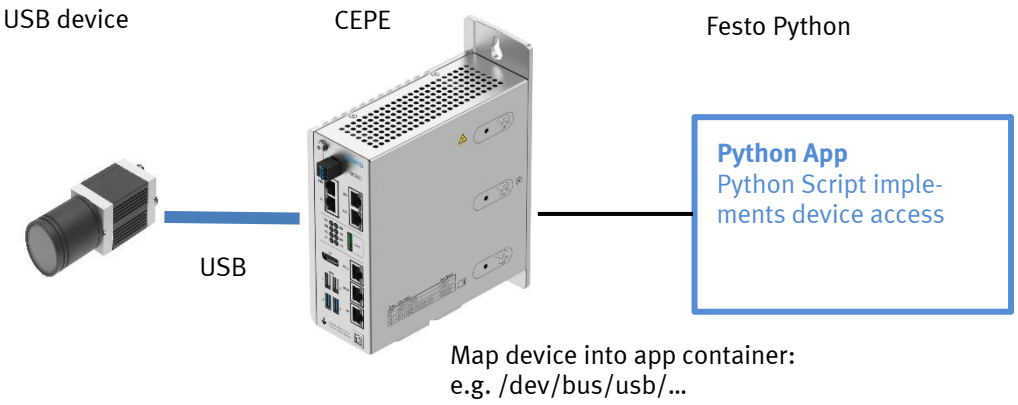
Certain USB3 Vision camera devices that are directly accessed under /dev/bus/usb are supported to be used from the Python App.
These device types have appropriate ownership mapping in the host system to make them accessible within the Python App:

- List USB devices from the built in terminal:

  lsusb

- Devices with the file owner “coder” are available for interaction:

  ls -la /dev/bus/usb/*



The techniques described in Install additional libraries can be used to install Camera drivers if supported by the SDK.