Safety Education Kit

Virtual Safety with CODESYS and A4P

Project Documentation

„Virtuelle Safety und ein A4P-Safety-Education-Kit“

Authors:

Lukas Fischer
Patrick Leitner
Jakob Güttinger
Leopold Weber

Institution:
Hochschule Kempten

Partner:
GROB, Magnet-Schultz, Christ Electronics, etc.

Contact:
Aaron Windmüller (CODESYS)
a.windmueller@codesys.com

June 20, 2025

Contents

1 Introduction

The project "Virtual Safety and A4P Safety Education Kit" aims to integrate the "Virtual Safety" functionality of CODESYS, evaluate its application within the "Allgäu 4 Production (A4P)" platform, and develop a Safety Education Kit.

2 Virtual Safety Control and System Requirements

2.1 Concept of Virtual Safety Control

A central aspect of this project is the use of *CODESYS Virtual Safe Control*, a revolutionary technology that enables functional safety according to IEC 61508 (SIL3) on standard industrial PCs or servers without the need for specialized, redundant safety hardware.

The core concept is based on fully implementing the traditionally hardware-based dual-channel safety architecture in software. Instead of using two separate, mutually monitoring processors, a standard CPU is utilized, on which the safety application runs in a virtualized environment. This is achieved through the TÜV SÜD-certified method of "Diversified Encoding" (based on Coded Processing):

- **Two logical software channels:** The safety application is executed sequentially on a single CPU core in two logical channels.
  - **1st Channel:** Executes the application in its original form.
  - **2nd Channel:** Executes the application with encoded algorithms. This encoding enables the detection of execution errors.
- **Continuous comparison:** The results of both channels are constantly compared. In case of discrepancies, a safe state is assumed. Additionally, the control flow in the encoded channel is continuously monitored.

This purely software-based approach eliminates the need for physical safety PLCs and allows control and safety functions to be flexibly consolidated on a single hardware platform. The advantages include significant cost and space savings, reduced dependency on specific hardware manufacturers, and high scalability, as safe control instances can be easily created virtually as needed.

3 System Requirements for Host Device

This section outlines the software and hardware requirements necessary for the host device to ensure seamless installation and operation of the CODESYS packages.

3.1 Software Requirements

The following subsections detail the software prerequisites for the host device, including architecture compatibility, operating system specifications, and real-time capabilities.

- **CODESYS Safe Control XS** (License ID: 230000001)
- **CODESYS PROFIsafe V2.6** (License ID: 230000006)

- **CODESYS Control Performance M** (License ID: 230000003)

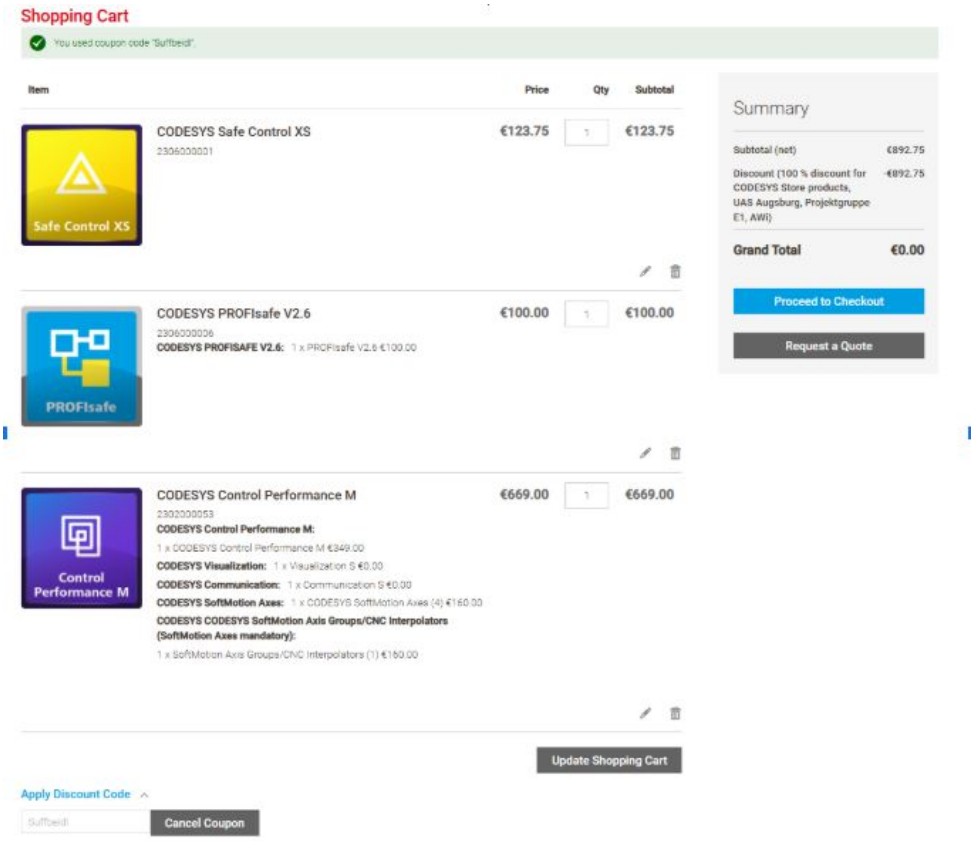


Figure 1: CODESYS Software Bundle

3.1.1 Architecture and Compatibility

To support the CODESYS environment, the host device must meet the following architectural requirements:

- **64-bit CPU (AMD64 architecture)**: Required for compatibility with modern CODESYS components.
- **Support for i386 (32-bit) architecture**: Essential when using `CODESYS Control for Linux SL - AMD64` in conjunction with `Safe Control`, which relies on 32-bit libraries.
- **Enable 32-bit support on Debian-based systems**:

```
sudo dpkg --add-architecture i386
sudo apt update
sudo apt install libc6-i386
```

3.1.2 Operating System Requirements

The host device must run an operating system with the following characteristics:

- **Debian-based distribution**: Compatible with the `dpkg` package manager (e.g., Debian, Ubuntu).
- **RPM-based systems**: From version **4.15.0.0** onward, the Linux Deploy Tool allows manual installation of RPM packages, supporting distributions like Red Hat.

3.1.3 Real-Time Capability

For time-critical applications, real-time support is necessary:

- **Real-time Linux kernel**: Required to ensure deterministic system behavior.
- **Optimization guidance**: Refer to the official CODESYS documentation for tuning and setup: `https://content.helpmecodesys.com/de/CODESYS%20Control/_rtsl_performance_optimization_linux.html`

3.2 Hardware Requirements

The following hardware components are necessary to set up the project environment:

- **1 Router**: Ensures reliable network communication between hosts.
- **2 Debian Linux Computers**: Ideally each with two LAN interfaces. If unavailable, USB-to-Ethernet adapters may be used.
- **1 Windows PC**: Used for configuration, monitoring, or optional supervisory tasks.

3.2.1 Minimum Hardware Specifications

To ensure reliable performance under real-time and industrial conditions, the following hardware setups are the minimum recommended specifications. Systems with lower specs may not support the required CODESYS components effectively.

Host 1:

- AMD A6-7310 CPU
- 4 GB RAM
- 32 GB SSD
- Gigabit LAN + WLAN AC/N

Host 2:

- Intel Celeron N3150 CPU
- 4 GB RAM
- 32 GB SSD
- Gigabit LAN + WLAN AC/N

These configurations have been validated for use in the project. Performance or compatibility may degrade on systems with slower processors or less memory/storage.

4 Debian Installation

The following steps describe the installation of Debian:

4.1 Preparing the Installation Medium

1. Download the Netinst CD image for amd64 from `https://www.debian.org/CD/netinst/`.
2. Create a bootable USB stick using Rufus (`https://rufus.ie/en/`). Download the portable .exe file and use the default settings to flash.

4.2 BIOS Configuration and Starting the Installer

1. Connect a network, keyboard, and monitor to the IPC/PLC.
2. Boot into BIOS and set the USB stick as the primary boot device.
3. Press F10 and confirm settings.
4. The IPC should boot into the graphical installer.

4.3 Performing the Installation

1. The "Debian Net Installer" starts.
2. Select "Graphical Installer" (requires internet connection).
3. Follow the installation (language, keyboard layout, country, etc.).
4. Set up passwords and user accounts.

4.4 Partitioning the Hard Drive

1. Select "Guided - use entire disk".
2. Select "No separate partition".
3. Choose the Ext4 file system.
4. Click "Continue", "Continue", "Continue" $\rightarrow$ "Finish".

4.5 Mirror Selection

1. Select all suggested mirror servers.
2. Do not use a proxy.

4.6 Software Selection

1. Select "SSH Server".
2. Select "Standard System Utilities".
3. Deselect "Desktop Environment" as it is not required.

4. Click "Finish".

4.7 Reboot and First Steps

1. Reboot the system (without the USB stick!).
2. Log in as ROOT user (username: root, password: the one set during installation).

4.8 Creating a SUDO User

1. Install sudo and add the "codesys" user to the sudo group:

```
apt install sudo && adduser codesys sudo
```

2. Save and close Nano with Ctrl+X.

4.9 SSH Configuration

1. Open the SSH configuration file:

```
nano /etc/ssh/sshd_config
```

2. Uncomment the line by removing the #:

```
PermitRootLogin yes
```

4.10 Logging Out the ROOT User

1. Log out the ROOT user:

```
exit
```

2. Log in with the regular user.

4.11 Installing Python 3

1. Install Python 3:

```
sudo apt install python3
```

2. Log out:

```
exit
```

4.12 SSH Connection and Docker Installation

1. Log in via SSH (open PowerShell in Windows and use the following command) or use Putty and follow the Docker install guide:

```
ssh codesys@hostname
```

2. Follow the link to install Docker: `https://docs.docker.com/engine/install/debian/`

5 Making Debian Real-Time Capable

1. Install the real-time kernel and test tools:

```
sudo apt install linux-image-rt-amd64 rt-tests
```

2. Open the GRUB configuration:

```
sudo nano /etc/default/grub
```

3. Edit the line `GRUB_CMDLINE_LINUX_DEFAULT` depending on CPU type:

   For Intel Systems:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet igb.EEE=0 processor.max_cstate=0 \
processor_idle.max_cstate=0 intel_idle.max_cstate=0 clocksource=tsc
    tsc=reliable \
nmi_watchdog=0 nosoftlockup intel_pstate=disable idle=poll noht
    rcu_nocb_poll \
hugepages=1024 i915.enable_dc=0 i915.disable_power_well=0 mce=off
    hpet=disable \
numa_balancing=disable efi=runtime"
```

   For AMD Systems:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet idle=poll clocksource=tsc tsc=reliable \
nmi_watchdog=0 nosoftlockup hugepages=1024 rcu_nocb_poll \
numa_balancing=disable efi=runtime"
```

4. Optional: Set boot delay to 0:

```
GRUB_TIMEOUT=0
```

5. Save the file and regenerate GRUB:

```
sudo update-grub
```

6. Reboot the system:

```
sudo reboot
```

7. Check if the real-time kernel is active:

```
uname -a
```

   The output should contain something like `PREEMPT_RT`.

8. Test the real-time performance:

```
sudo cyclictest -p 99 -t -m
```

   Note: A latency under 100 µs is a good value.

9. Further optimization tips for real-time capabilities can be found at: `https://confluence.codesys.com/x/AoNZEQ`

Explanation of Key GRUB Parameters:

- `idle=poll` – Prevents CPU sleep states (for lower latency).
- `clocksource=tsc tsc=reliable` – Uses a stable time source.
- `rcu_nocb_poll` – Decouples RCU interrupts from certain CPUs.
- `hugepages=1024` – Reserves large memory pages.
- `nosoftlockup, nmi_watchdog=0` – Prevents interrupt issues.
- `intel_pstate=disable, noht, processor.max_cstate=0` – Relevant for Intel CPUs only.
- `i915.enable_dc=0, i915.disable_power_well=0` – Only relevant for Intel graphics.

6 For Native Linux SL Installation (a SoftSPS)

Now switch to the Windows PC with CODESYS installed. Install the following packages with the CODESYS installer 

, which provides an overview of the CODESYS Package Manager where you can search for the required packages in the search bar. The following packages are required to run CODESYS Control directly on a Linux host:

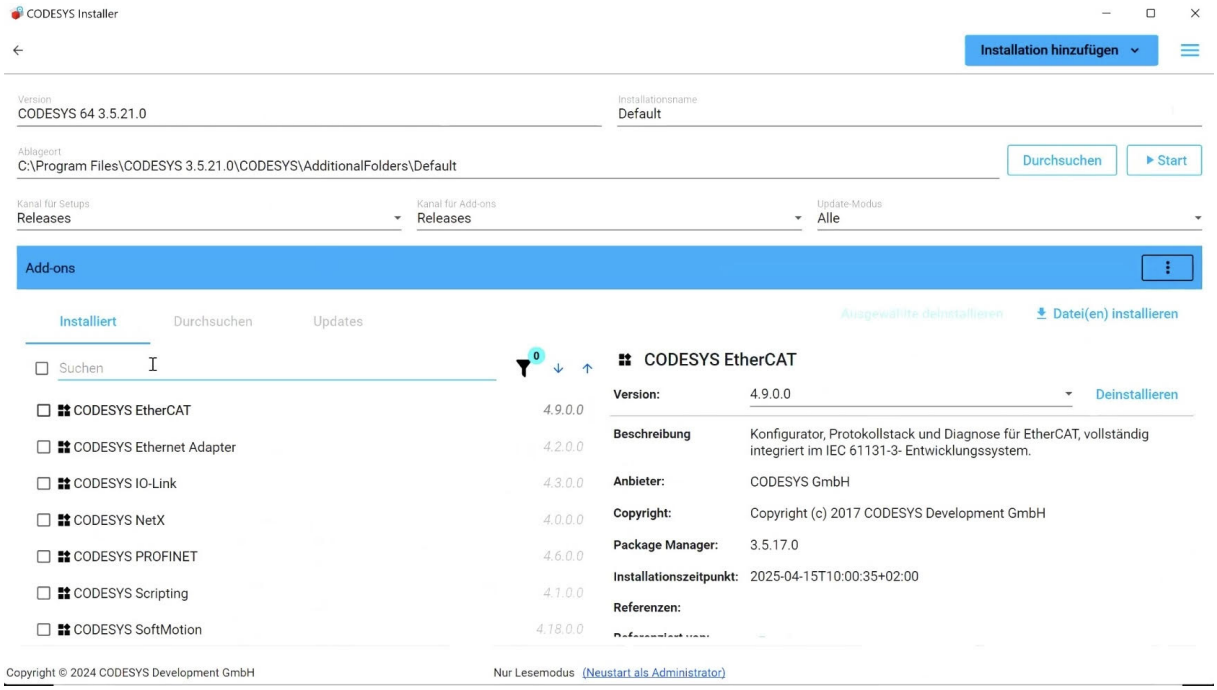


Figure 2: Overview of CODESYS package Manager. Search for the following Packages in the Search Bar

- **CODESYS Control for Linux SL**
  Note: ARM devices are currently not supported.
- **CODESYS Control SL Deploy Tool**
- **CODESYS Edge Gateway for Linux SL**

6.1 For Running Multiple Controllers with a Container Engine (e.g. Docker)

When running multiple SoftSPS instances on the same host:

- **CODESYS Virtual Control for Linux SL**
- **CODESYS Virtual Safe Control SL**

6.2 For Safety Features with CODESYS

If safety-relevant controllers are to be used, you additionally need:

- **CODESYS Safety Extension**

- **CODESYS Safe Control Service**
  Only required for systems with Safe Control Core.
- To integrate PROFINET-based fieldbus systems: CODESYS PROFINET

7 Network Configuration

For stable operation and correct communication with CODESYS and the Timeprovider, the following network configuration is recommended:

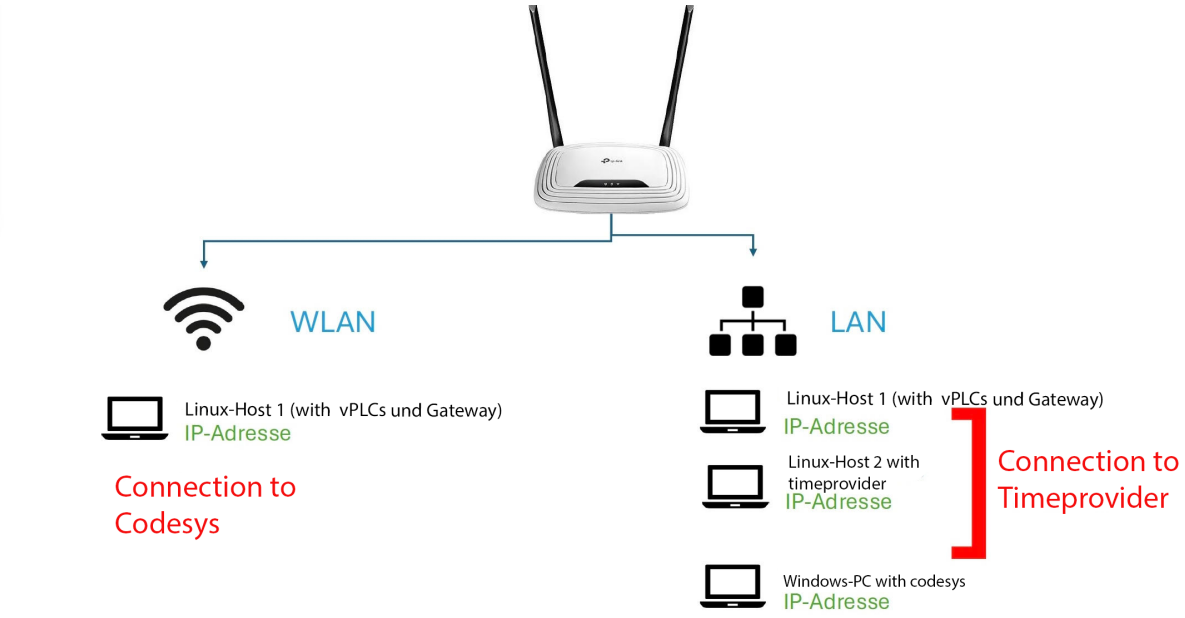


Figure 3: Recommended network configuration with WLAN and LAN devices

Network Structure

The network consists of a central router that provides both WLAN and wired LAN connections. The individual devices in the network are connected as follows:

- **WLAN:**
  - A Windows PC with CODESYS is connected via WLAN.
  - This device has an assigned IP address and establishes the connection to the CODESYS controller.
  - PCs connected via WLAN can also be connected via LAN if needed. (WLAN was used in this setup due to a lack of available LAN ports.)
- **LAN:**
  - All LAN devices have static IP addresses.
  - These devices are connected via LAN to a **Timeprovider** to provide a precise time source within the network.
  - Devices connected via LAN must use wired connections.

Two dedicated systems are required to ensure certification and thus security.

8 Installing Runtime Systems

With the CODESYS Deploy Tool, a connection can be established to the Linux host where the necessary package-based or container-based runtime systems are to be installed. Navigate to: `Tools → Deploy Control SL → Connect to the Linux target system via SSH connection.`

In the **Deployment** tab, all available package- and container-based runtime system versions and components that can be installed on the device are listed. Here, the appropriate package (or container) and its version can be selected and installed on the Linux device.

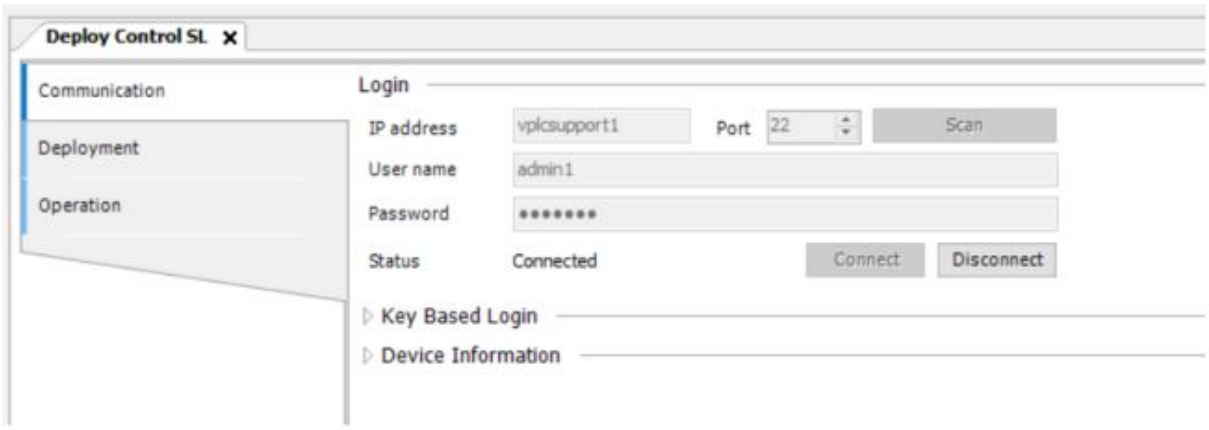


Figure 4: Open Deploy Control: shows target device's IP address (WLAN), username ("with root access"), and Linux user password input

9 Deployment to use Control Linux SL (without Docker)

For the usage of CODESYS Control for Linux SL you have to install the following Packages:

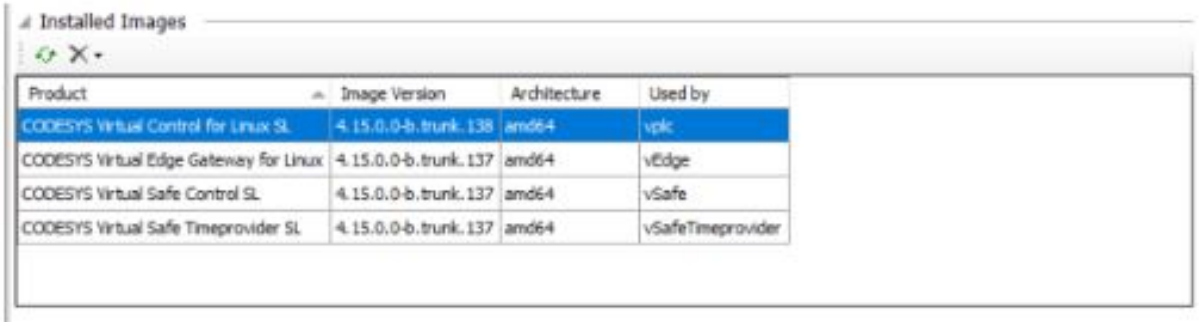


Figure 5: What you see under "Deployment"

10 Installing CODESYS Virtual Control for Linux SL

This section provides a comprehensive guide for deploying the CODESYS Virtual Control for Linux SL runtime environment, including both a virtual standard PLC (vPLC) and a virtual safety PLC (vSafePLC) with a dedicated Timeprovider. The deployment is split across two separate Linux systems for real-time safety certification and architectural integrity.

10.1 Deployment Overview (Using Docker Containers)

To implement this scenario, the following setup is required:

- **Linux Host PC 1** – Hosts the vPLC and vSafePLC runtime systems.
- **Linux Host PC 2** – Hosts the Safety Timeprovider service.

An internet connection and Docker must be available on both systems. SSH should be configured to allow remote connection via the CODESYS Deploy Tool.

Refer to the official CODESYS documentation for background: `https://content.helpme-codesys.com/en/CODESYS%20Control/_rtsl_scenario_safe_house.html`

Step 1: Install the vSafe Timeprovider on Linux Host PC 2

1. Launch the CODESYS Deploy Tool.
2. Connect to **Linux Host PC 2** via `Tools → Deploy Control SL → Communication tab`.
3. Switch to the **Deployment** tab.
4. Select `CODESYS Virtual Safe Time Provider SL` from the product list.
5. Choose the latest available version and click `Install`.
6. Switch to the **Operation** tab and click the + button to add a new instance.
7. Name the instance `timeprovider`, select **Safety Timeprovider** from the filter list, and use the latest version.
8. In the instance settings:
    - Set `TARGET_IP` to the IP address of **Linux Host PC 1**.
    - Ensure `TARGET_PORT` is `60000`.
    - Set `Autostart` to `Yes`.
9. Click `Save`, then `Start` the instance.

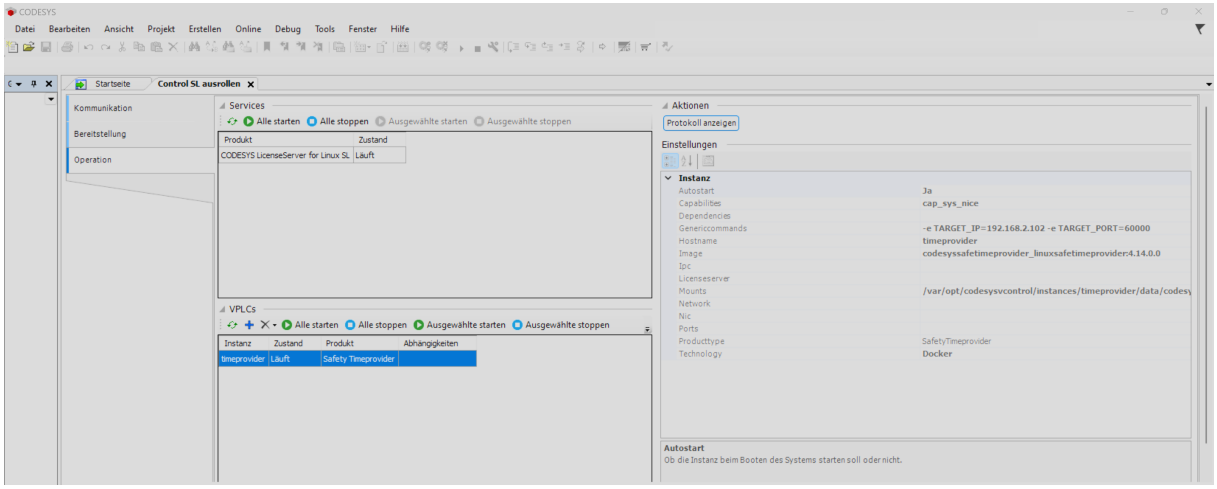


Figure 6: Deploy Tool – Configuration of Timeprovider instance on Linux Host PC 2.

Step 2: Install vPLC and vSafePLC on Linux Host PC 1

1. Connect to **Linux Host PC 1** via the CODESYS Deploy Tool.
2. In the **Deployment** tab, install:
   - `CODESYS Virtual Control SL` (vPLC)
   - `CODESYS Virtual Safe Control SL` (vSafePLC)
3. Go to the **Operation** tab and click + to add new instances:
   - Create `vPLC` – Select **Runtime System**.
   - Create `vSafePLC` – Select **Safety Runtime System**.
4. Configure the `vSafePLC` instance:
   - `Ports`: `60000:60000/udp`
   - `IPC`: `container:vPLC`
   - `Dependencies`: `vPLC` must start first
5. Configure the `vPLC` instance:
   - Enable `shareable` IPC namespace
6. Click `Start All` to run both instances.

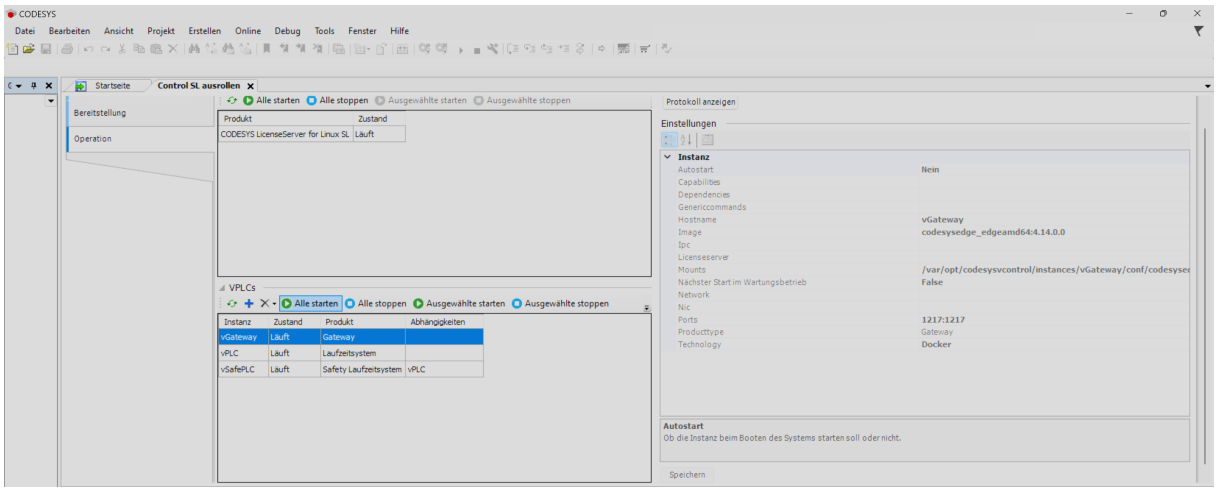


Figure 7: VGateway Configurations. Feel free to copy the Mounts

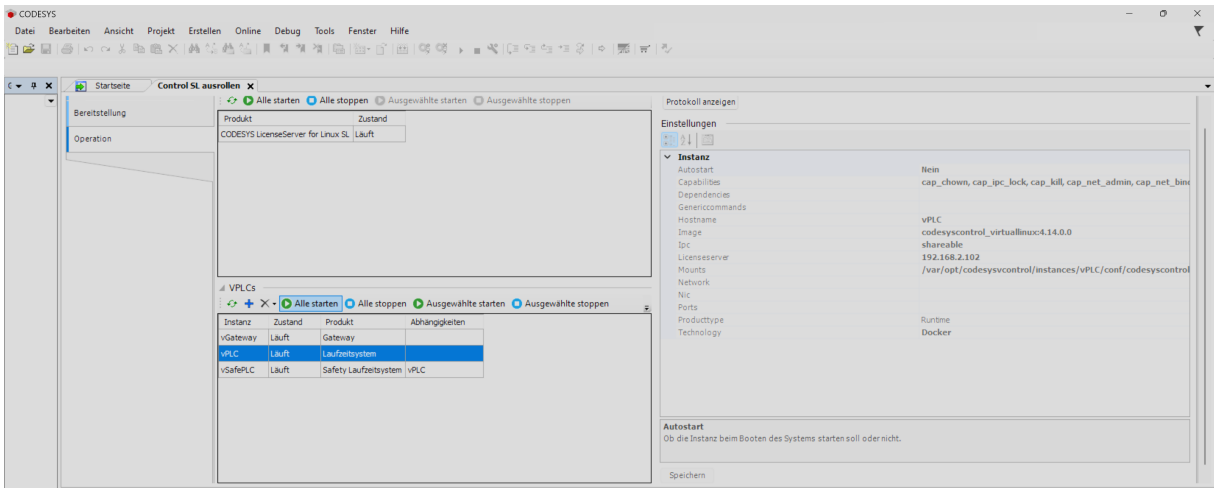


Figure 8: VPLC Configurations

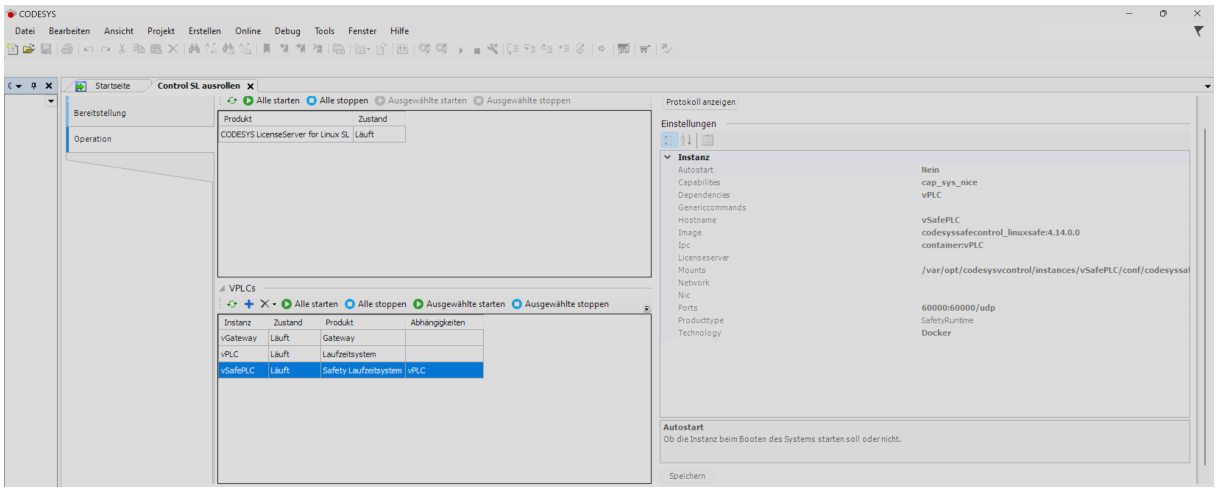


Figure 9: Use the same Mounts

Step 3: Verify the Time Synchronization

To confirm proper communication between the Timeprovider and vSafePLC:

- In the Deploy Tool, select the `vSafePLC` instance.
- Open the log via the `Show Log` action in the top-right corner.
- Check for the message: `External Time Provider found`.

If this message is present, the safety time synchronization is functioning correctly.

All runtime systems are now installed and running. You can proceed to create your project and integrate PROFIsafe.

11 CODESYS Project for Using CODESYS Safe Control

11.1 Importing Devices

To automatically generate the logical safety devices in the device tree, a specific import option must be enabled in the PROFINET plugin.

After enabling this setting, re-import the GSDML file for the corresponding PROFINET fieldbus devices.

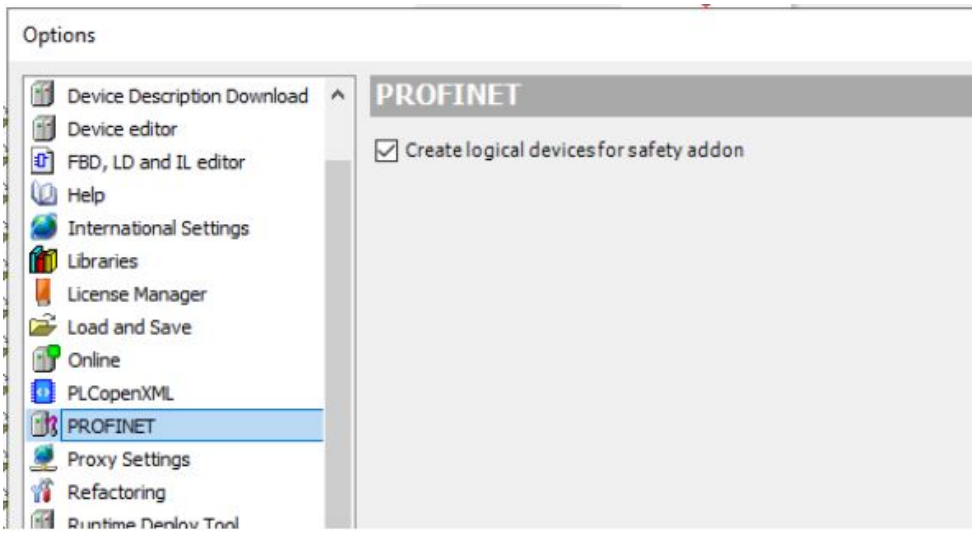


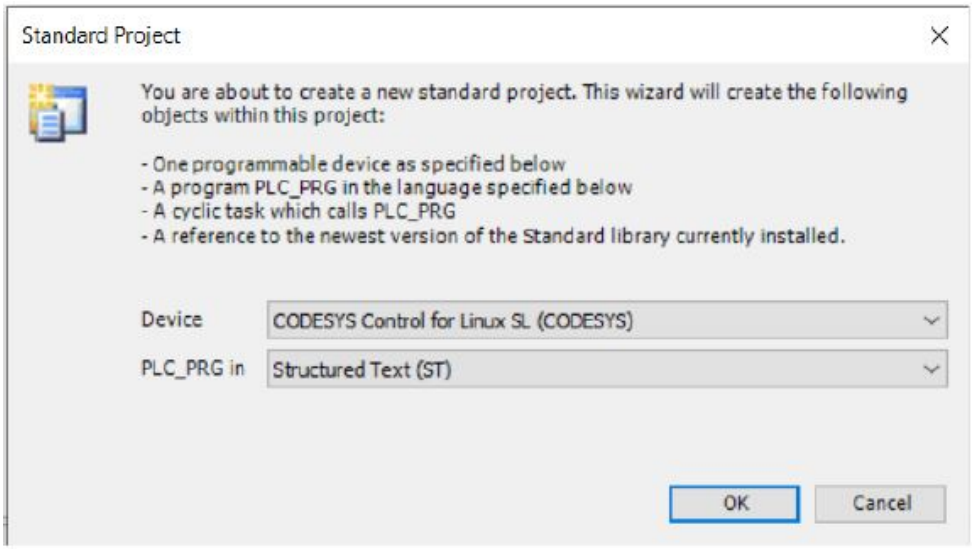
Figure 10: VPLC Configurations

Note: Ensure the following option is enabled:
`Options → PROFINET → Create logical devices for safety addon`

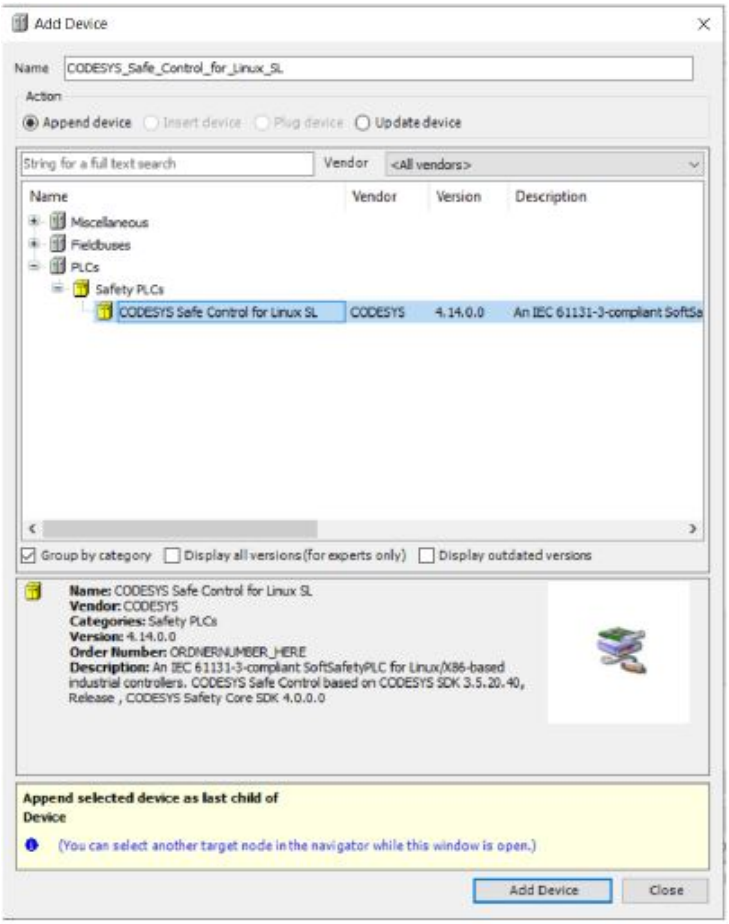
11.2 Creating a New Project

Note: Using the *Empty Safety Project* template will automatically enable user management for the safety project.

Alternatively, you can use the *Empty Project* or *Standard Project* templates. If using the *Standard Project*, select the following device: `CODESYS Control for Linux SL (CODESYS)`



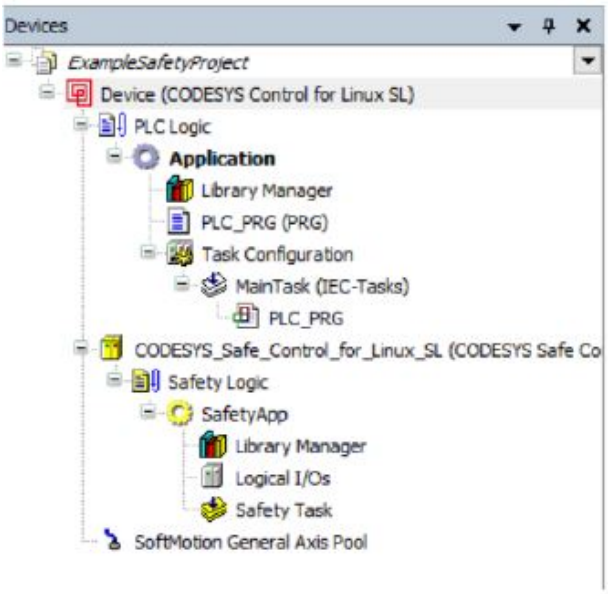
In the project tree, right-click on the device and select: `Add Device → PLCs`, then choose the appropriate `CODESYS Safe Control` device to add it to your project.



11.3 Project Tree Overview

After adding the `CODESYS Safe Control for Linux SL` controller, the project tree now includes:

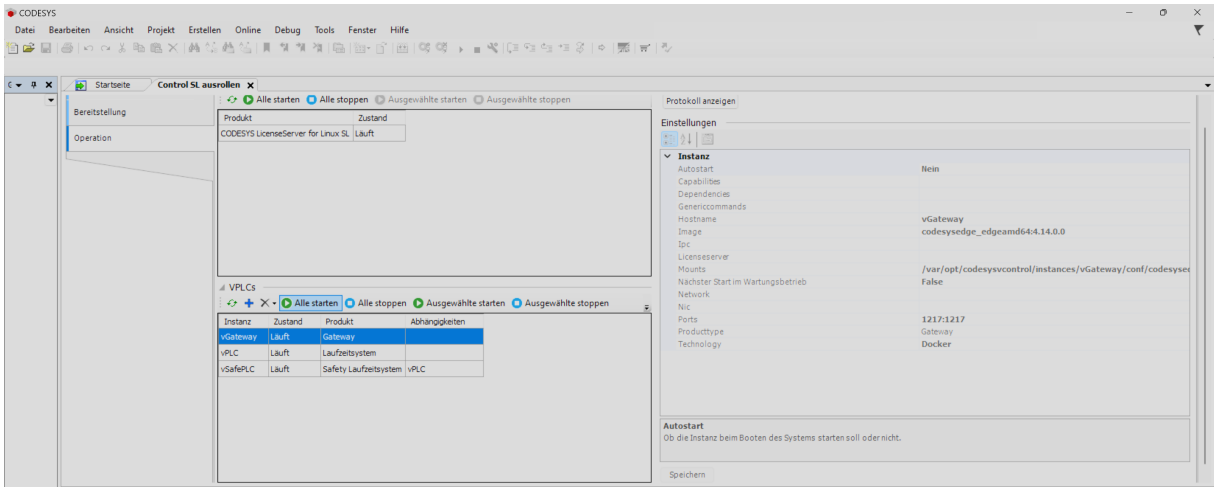
- A dedicated **Safety Logic**
- A **Safety Application**
- A separate **Library Manager**
- Logical **I/Os**
- A defined **Safety Task**



These components provide the foundation for developing and managing the safety-related aspects of your application in CODESYS.

11.4 Communication with Linux Runtime Systems

To establish communication between the CODESYS development environment and the Linux runtime system, a gateway is required. This can either be a local gateway or a **CODESYS Edge Gateway for Linux** installed directly on the target device.



Initial setup instructions for installing the gateway and establishing communication with a Linux runtime can be found here:

`https://content.helpme-codesys.com/en/CODESYS%20Control/_rtsl_load_and_start_application.html`

11.5 Installing a Safe Timeprovider

A **Safe Timeprovider** is always required for the operation of hardware-independent Safe Control runtime systems. This component is available as a separate software package and provides essential timing information for safety-related applications.

For improved fault detection, the timeprovider should run on a **second device** with a different CPU. It periodically sends a timestamp to the device running the safety controller, allowing it to compare and detect discrepancies in CPU clock timing—an important aspect of safety certification.

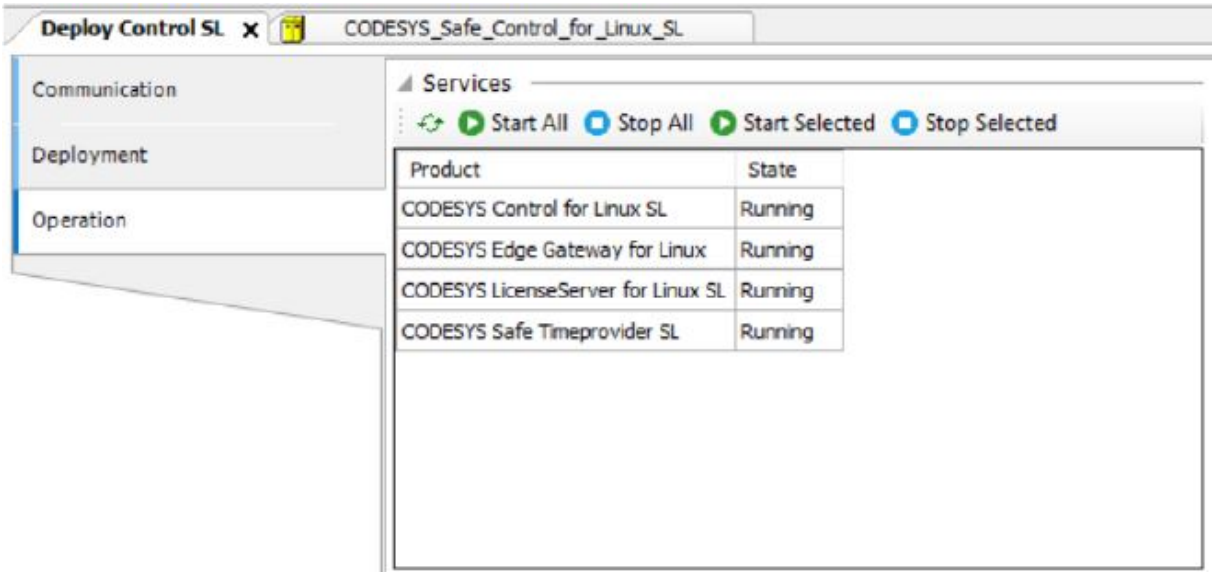
Development and Testing: For initial testing, offline programming, or virtual commissioning, the timeprovider may also be installed on the **same device** as the safety controller.

Production Systems: For deployment in a certified application, the timeprovider **must be installed on separate hardware** to ensure safety integrity.

If you are using a package-based controller, the `Safe Timeprovider SL` can be deployed via the **CODESYS Deploy Tool**.

Same-Device Configuration: If the Safe Timeprovider runs on the same machine as the `CODESYS Safe Control`, no additional environment variable is required. The time signal is sent by default to:

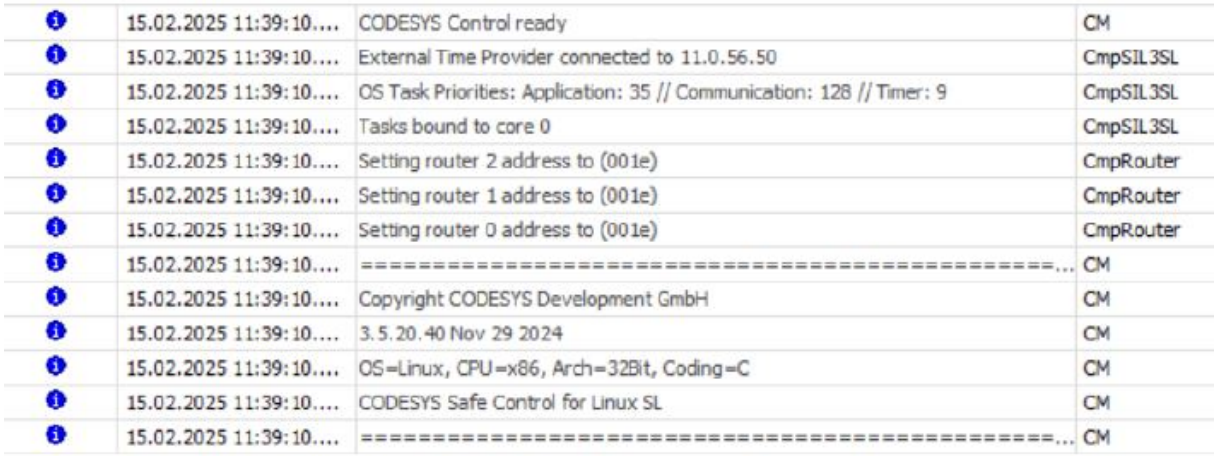
- `localhost` on port `9000`



Two-Device Configuration: If the Safe Timeprovider runs on a separate Linux device, you must set the target IP address as an environment variable on that system:

```
export TARGET_IP=XXXX.XXXX.XXX.XXXX
```

In the `Safe Control` log, a successful connection to the timeprovider should be visible before performing a download. This indicates that synchronization is working correctly.



ℹ	15.02.2025 11:39:10....	CODESYS Control ready	CM
ℹ	15.02.2025 11:39:10....	External Time Provider connected to 11.0.56.50	CmpSIL3SL
ℹ	15.02.2025 11:39:10....	OS Task Priorities: Application: 35 // Communication: 128 // Timer: 9	CmpSIL3SL
ℹ	15.02.2025 11:39:10....	Tasks bound to core 0	CmpSIL3SL
ℹ	15.02.2025 11:39:10....	Setting router 2 address to (001e)	CmpRouter
ℹ	15.02.2025 11:39:10....	Setting router 1 address to (001e)	CmpRouter
ℹ	15.02.2025 11:39:10....	Setting router 0 address to (001e)	CmpRouter
ℹ	15.02.2025 11:39:10....	==...	CM
ℹ	15.02.2025 11:39:10....	Copyright CODESYS Development GmbH	CM
ℹ	15.02.2025 11:39:10....	3.5.20.40 Nov 29 2024	CM
ℹ	15.02.2025 11:39:10....	OS=Linux, CPU=x86, Arch=32Bit, Coding=C	CM
ℹ	15.02.2025 11:39:10....	CODESYS Safe Control for Linux SL	CM
ℹ	15.02.2025 11:39:10....	==...	CM

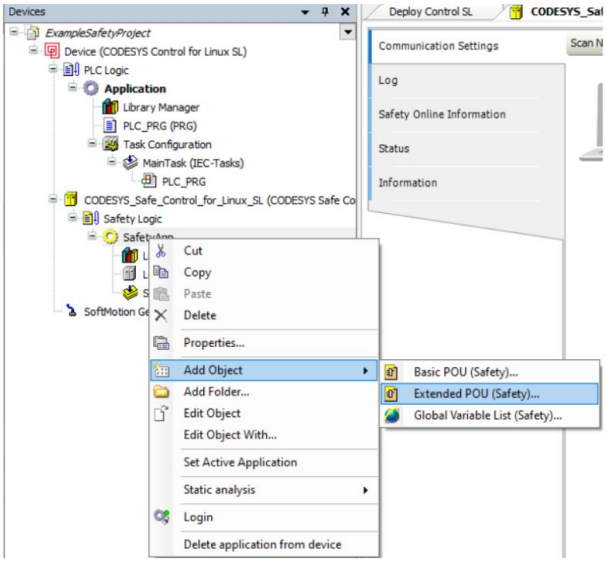
Changing the Time Cycle: The default sending cycle is `10 ms`. You can override this setting using the `CYCLETIME` environment variable. For example, to change the cycle time to `3 ms`:

```
export CYCLETIME=3
```

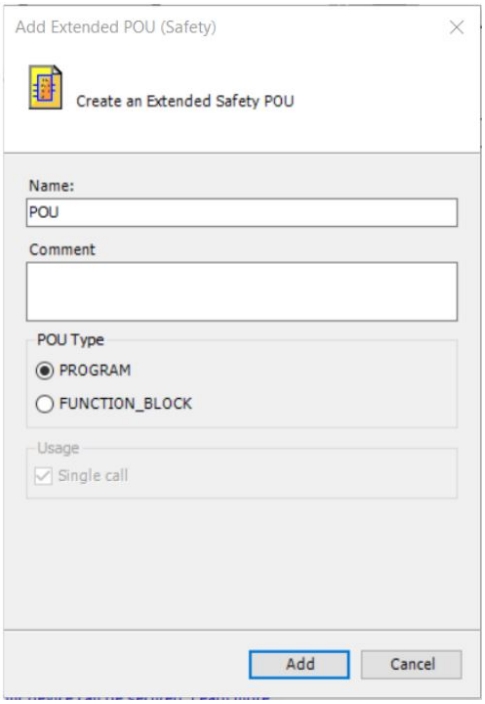
11.6 Example: Adding a Safe Application Program

As an introductory example, this section illustrates how a simple counter can be created within a safe application.

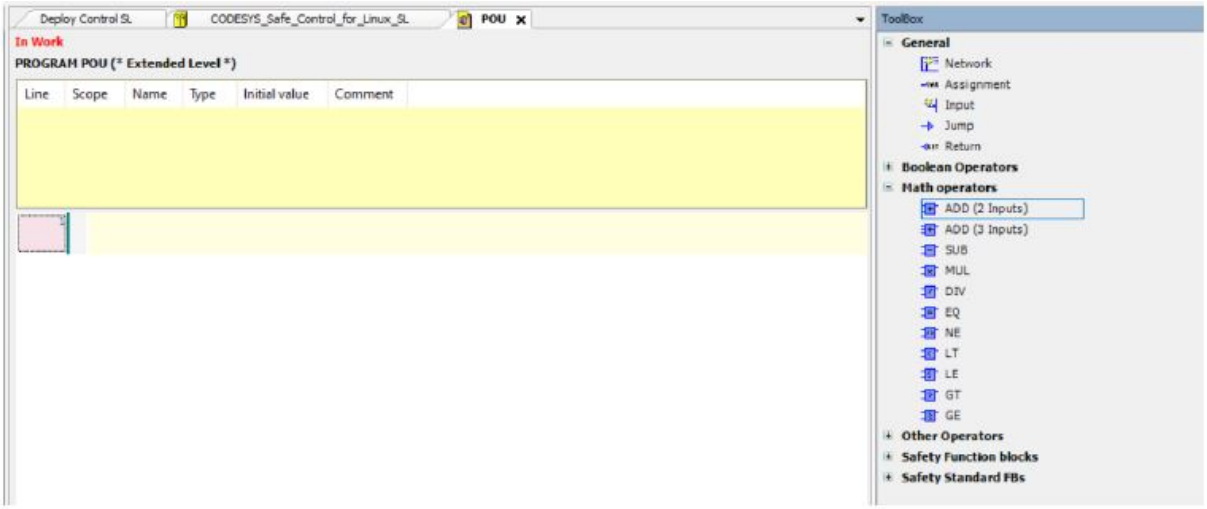
By right-clicking on the `SafetyApp` and selecting `Add Object` → `Extended POU (Safety)`, a new program can be added.



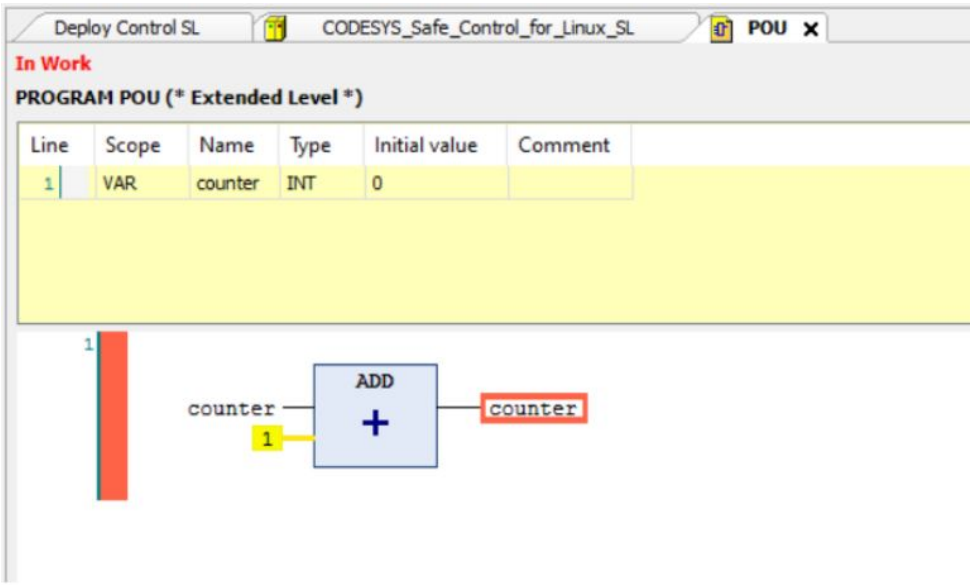
For the POU, a selection can be made between a program or a function block.



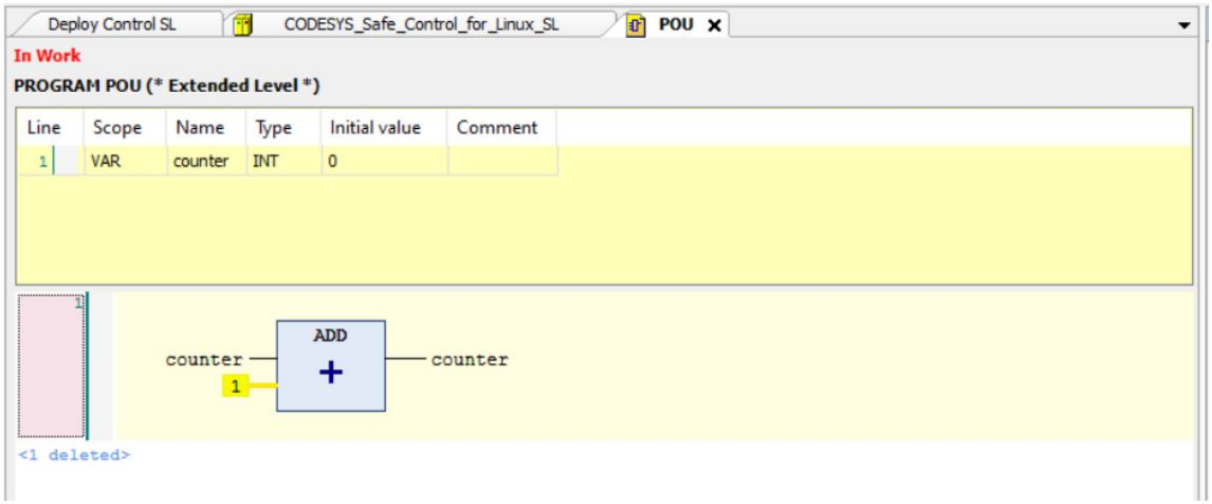
Within the program, a network is already defined. Using the toolbox, an `ADD` adder block with two inputs can be added.



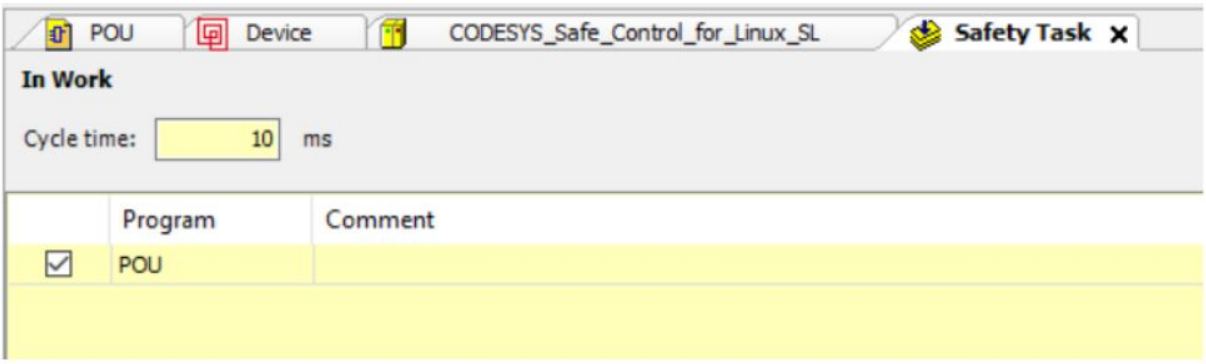
A variable named `counter` can be added to one input. The second summand can be added to the second input.



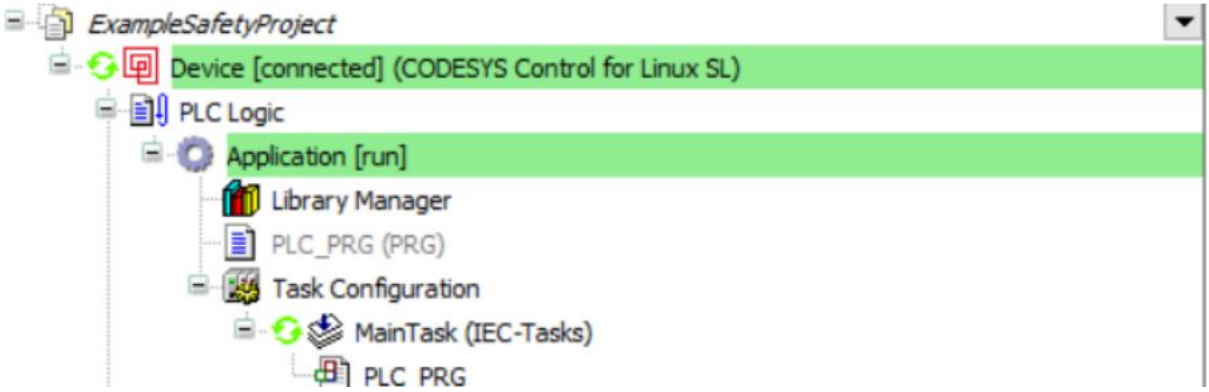
Changes in a Safe POU are marked in red, and this does not indicate an error. Finally, a network can be added and subsequently deleted, ensuring the POU is no longer marked in red.



After adding the POU to the project tree, it is automatically assigned to the Safety Task.

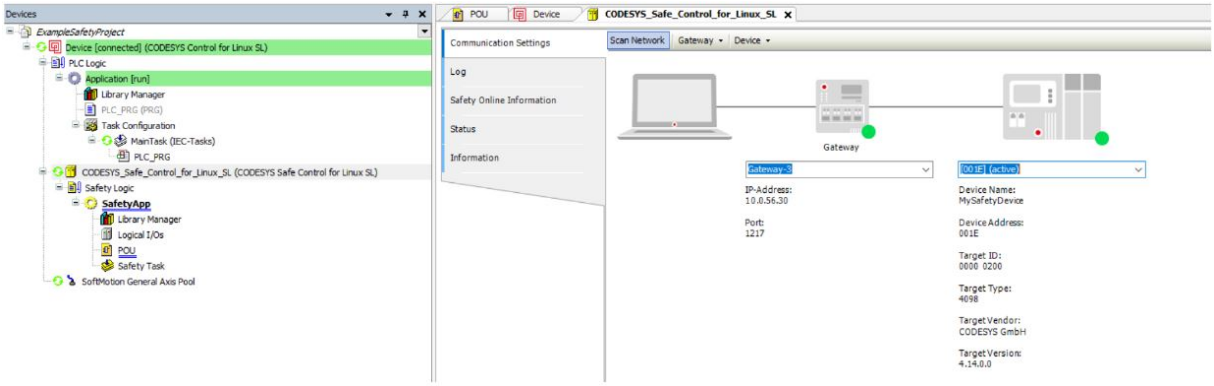


Subsequently, you can connect to the Linux runtime system and download the project to the target device.

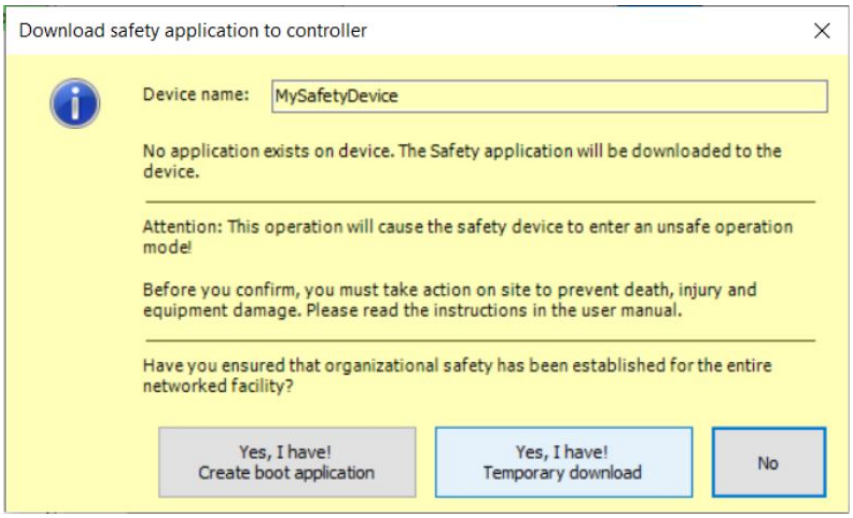


As previously described, the Safe Control log can be checked to confirm the connection to the timer.

The `Safe Control SafetyApp` can then be selected as the active application. Via the communication dialog of the device, you can log in to the safety controller.

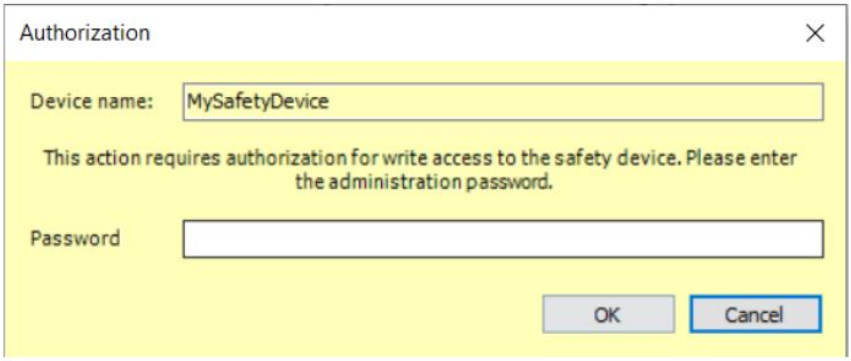


After logging in, a yellow dialog window will prompt for the type of download.

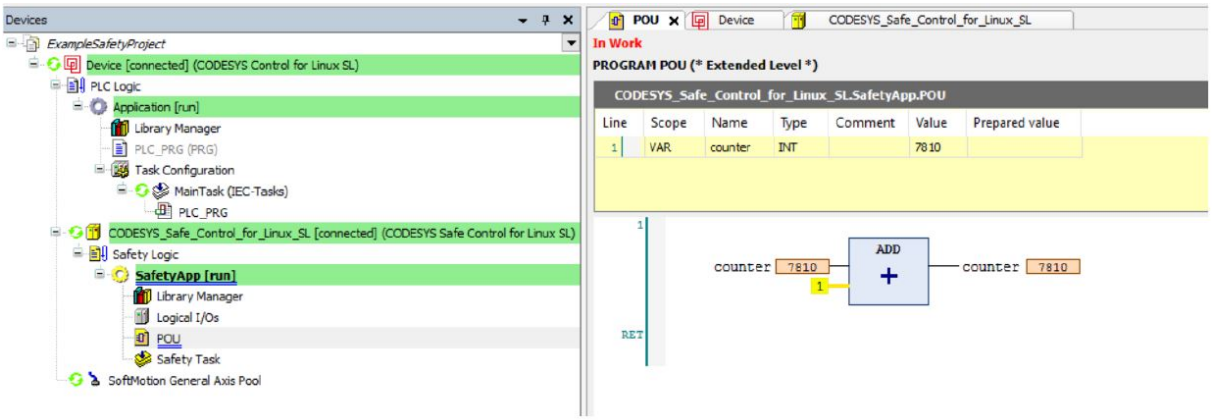


For a brief test of the counter, a temporary download can be performed. A window will then open, requesting a password.

Since no password has been defined on the controller yet, the field can be left empty and confirmed by clicking `OK`.

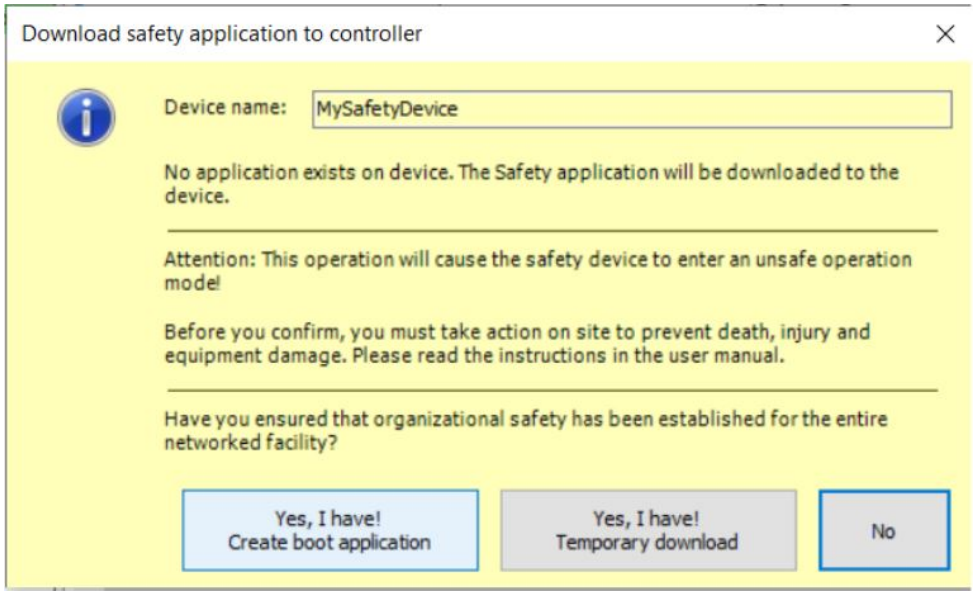


The application is then downloaded and in the `Stop` state. After starting the application, the `counter` variable in the POU should cyclically increment.

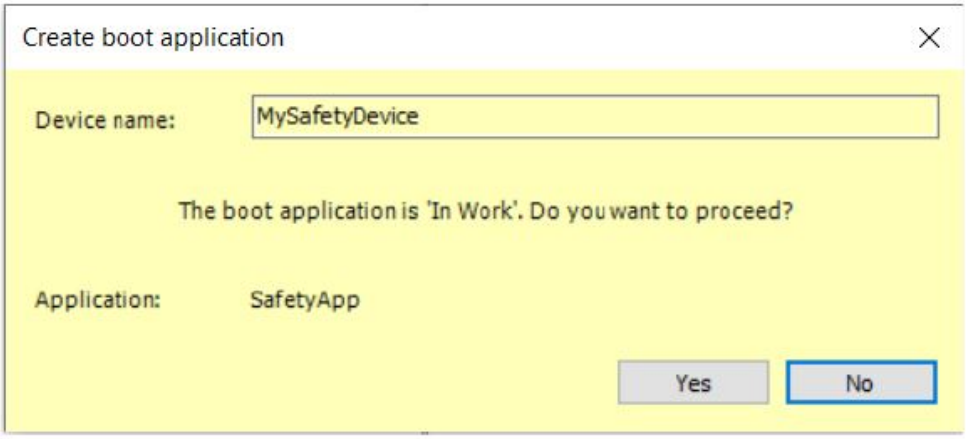


During a normal download (DL), the program still runs in the unsafe state (DL).

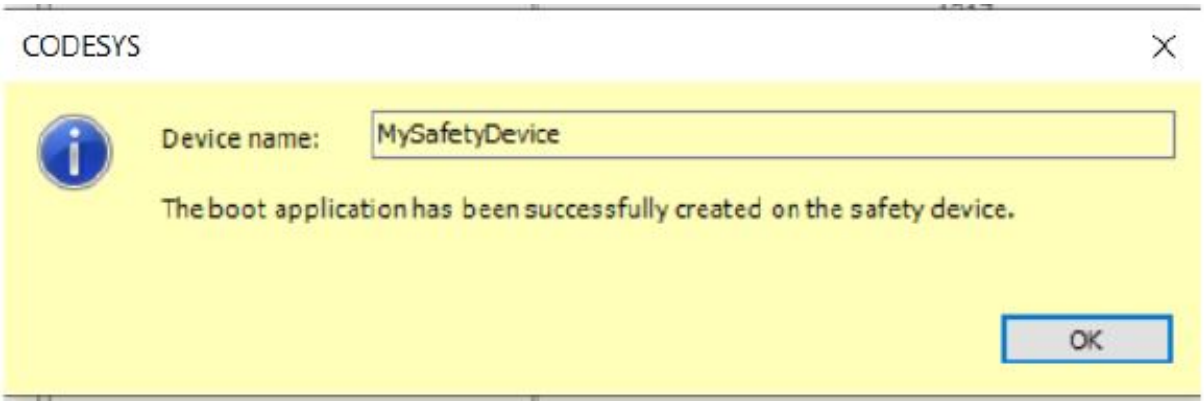
To run the program in a safe state, a boot application must be created on the safety controller. Creating the boot application requires logging out of the safety controller, which unloads the download.



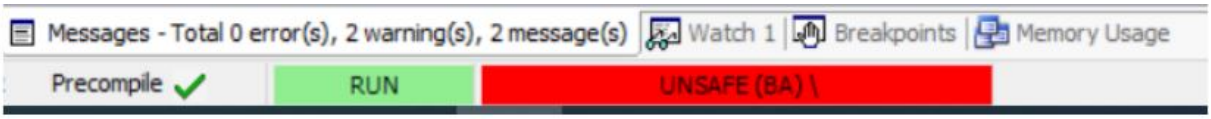
The boot application is then selected. You will be prompted to confirm whether you want to create the boot application, which must be confirmed with `Yes`.



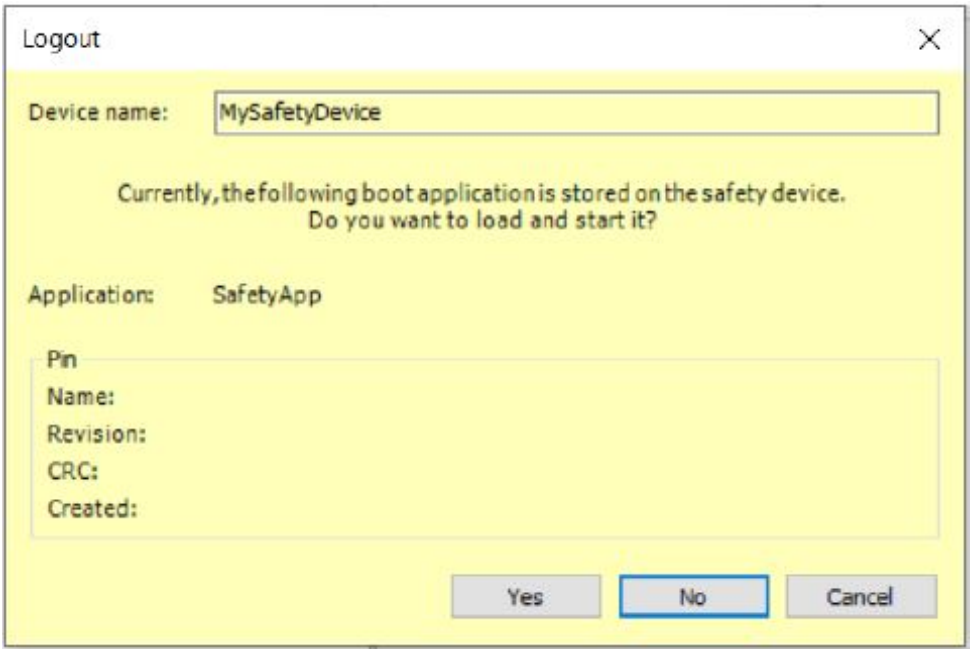
Subsequently, you will receive confirmation that the boot application has been created on the device. The application is now in the `Stop` state again.



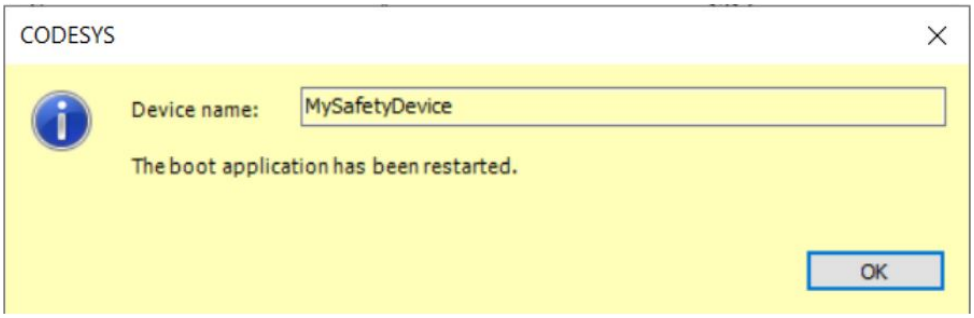
After starting the application, it remains in the unsafe state but with a boot application



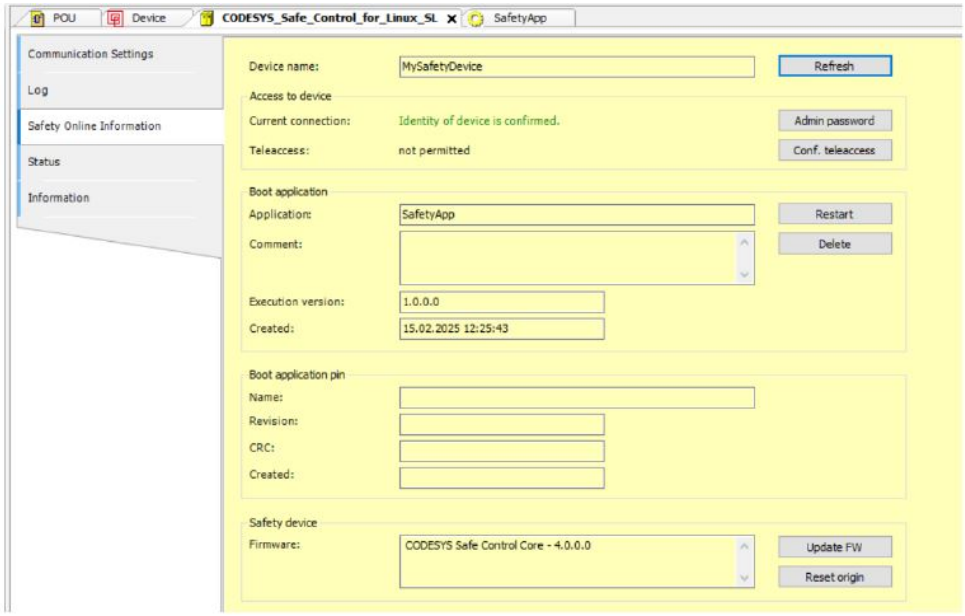
To enter the safe state during the boot application, a logout is necessary along with the confirmation of the restart of the boot application.



Now the application has been restarted.



To view the status of the boot app, the Safety Online Information dialog from the Safe Control can be accessed.

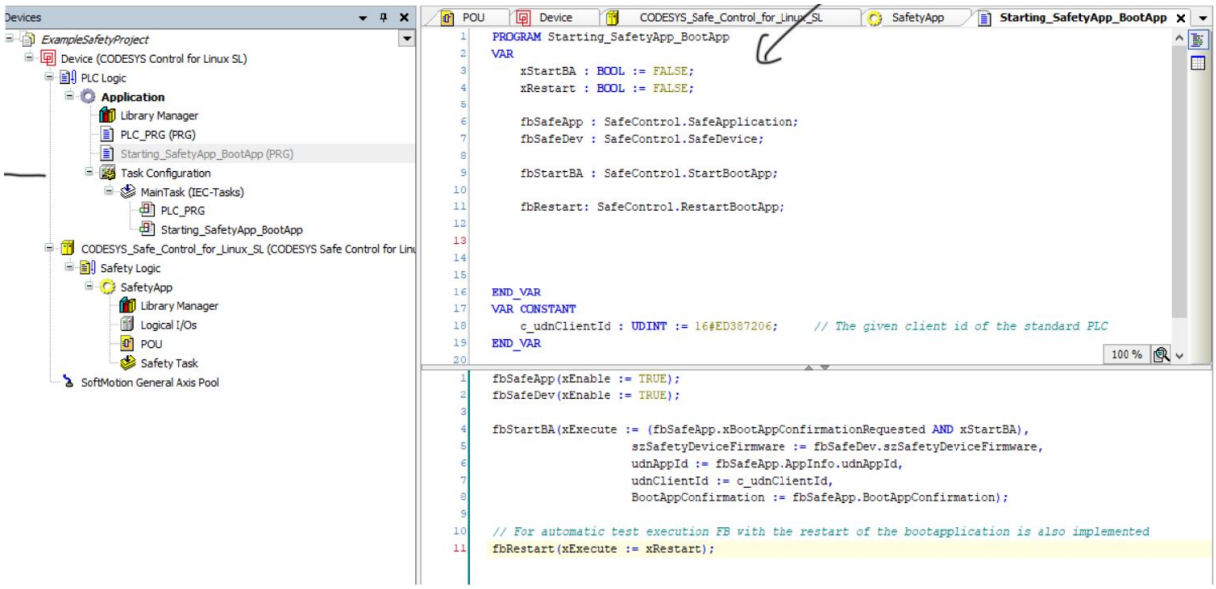


After logging in again to the safety controller, a confirmation to start the boot app is requested in the log messages from the Safety Runtime.

	15.02.2025 12:31:20....	Confirmation to start the boot application is requested	CmpSIL3SL
	15.02.2025 12:31:20....	Restart boot application by user [redacted]	CmpSIL3SL
	15.02.2025 12:26:46....	Create boot application by user [redacted]	CmpSIL3SL
	15.02.2025 12:26:46....	Activate boot application on device	CmpSIL3SL
	15.02.2025 12:26:45....	Delete boot application by user [redacted]	CmpSIL3SL
	15.02.2025 12:26:45....	Delete boot application on device	CmpSIL3SL
	15.02.2025 12:22:58....	Delete boot application by user [redacted]	CmpSIL3SL
	15.02.2025 12:22:58....	Delete boot application on device	CmpSIL3SL
	15.02.2025 12:22:38....	Create boot application by user [redacted]	CmpSIL3SL
	15.02.2025 12:22:38....	Activate boot application on device	CmpSIL3SL
	15.02.2025 12:22:38....	Delete boot application by user [redacted]	CmpSIL3SL
	15.02.2025 12:22:38....	Delete boot application on device	CmpSIL3SL
	15.02.2025 11:46:28....	Delete boot application by user [redacted]	CmpSIL3SL

The application is now in the safe state but has not yet started.

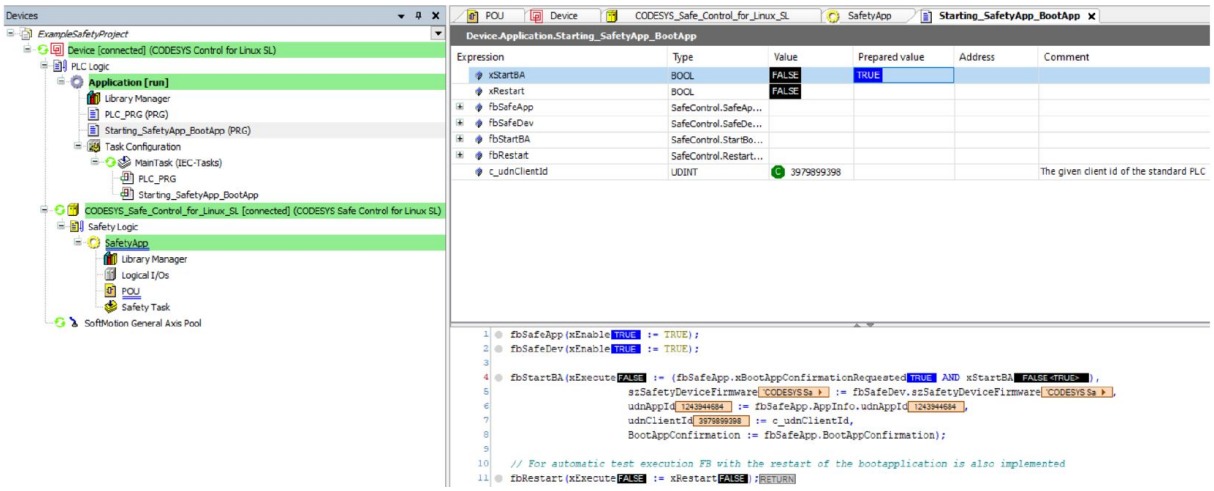
The start of the boot app is performed via the non-safe standard controller. For this purpose, a program was created that implements the `SafeControl.StartBootApp` FB.



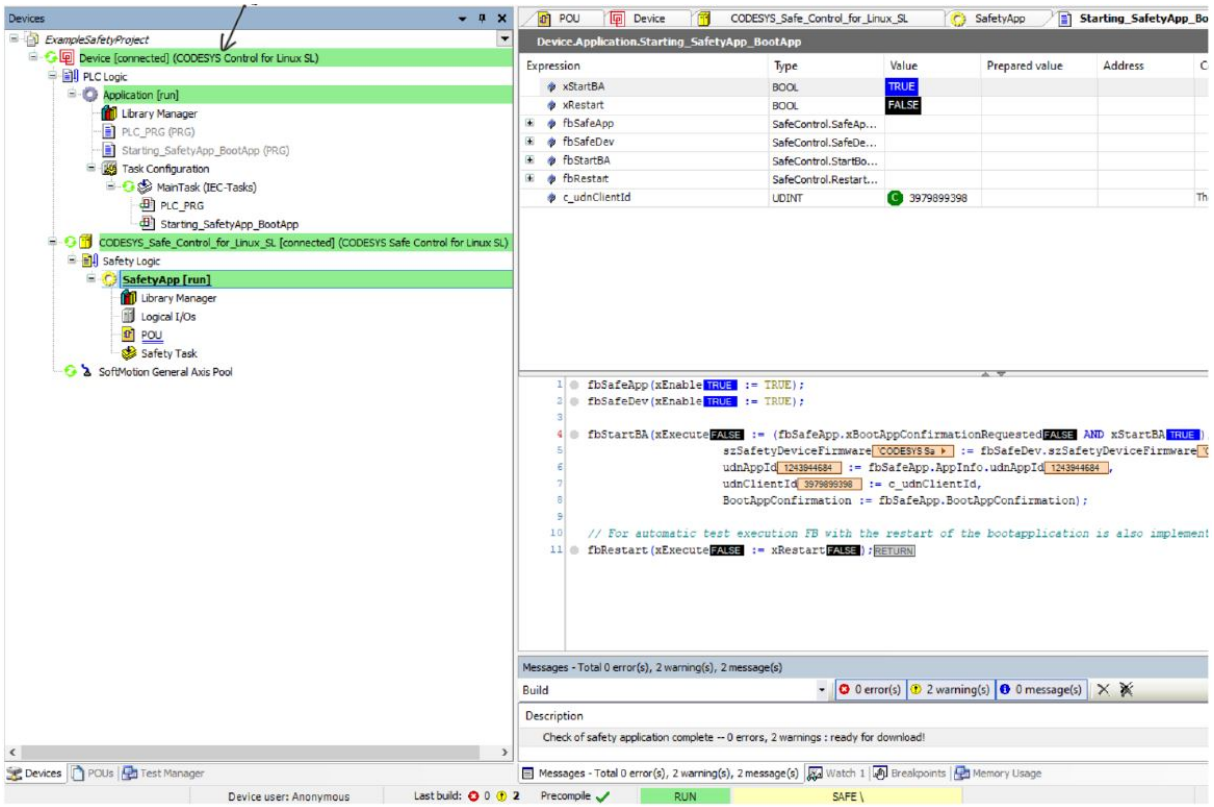
Code:

```
PROGRAM Starting_SafetyApp_BootApp
VAR
xStartBA : BOOL := FALSE;
xRestart : BOOL := FALSE;

fbSafeApp  : SafeControl.SafeApplication;
fbSafeDev  : SafeControl.SafeDevice;

fbStartBA  : SafeControl.StartBootApp;
fbRestart  : SafeControl.RestartBootApp;
END_VAR

-----------------------------------------------------------------

VAR CONSTANT
c_udnClientId : UDINT := 16#ED387206;   // The given client id of the standard PLC
END_VAR

fbSafeApp(xEnable := TRUE);
fbSafeDev(xEnable := TRUE);

fbStartBA(xExecute := (fbSafeApp.xBootAppConfirmationRequested AND xStartBA),
szSafetyDeviceFirmware := fbSafeDev.szSafetyDeviceFirmware,
udnAppId := fbSafeApp.AppInfo.udnAppId,
udnClientId := c_udnClientId,
BootAppConfirmation := fbSafeApp.BootAppConfirmation);

// For automatic test execution FB with the restart of the bootapplication is also i
fbRestart(xExecute := xRestart);
```

With the variable `xStartBA`, the boot application can be manually activated.



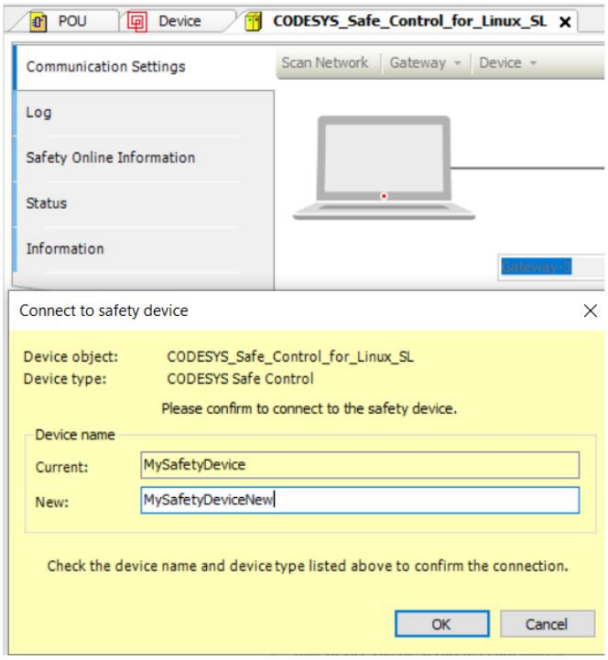
Subsequently, the Safe Control is started and is in the safe state (BA).



11.7 Renaming the Safe Control Device

Renaming the safety controller is possible via the communication settings:

`Device → Active Device → Rename`

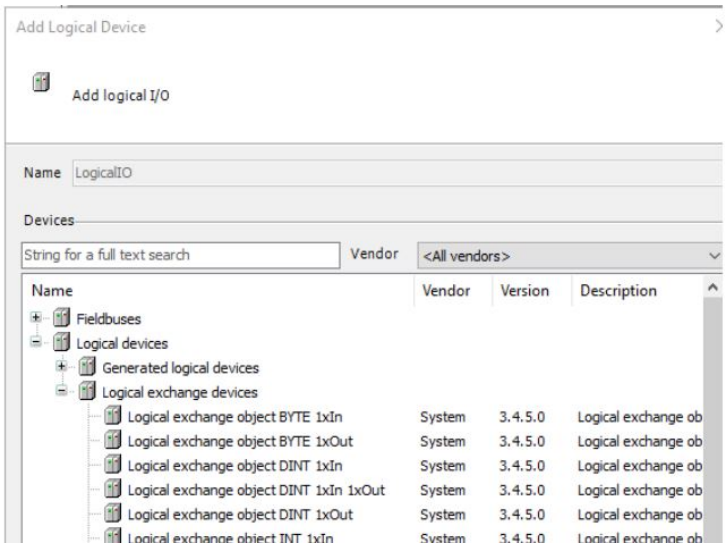


12 Logical Devices

12.1 Data Exchange Between Safety and Standard Controller

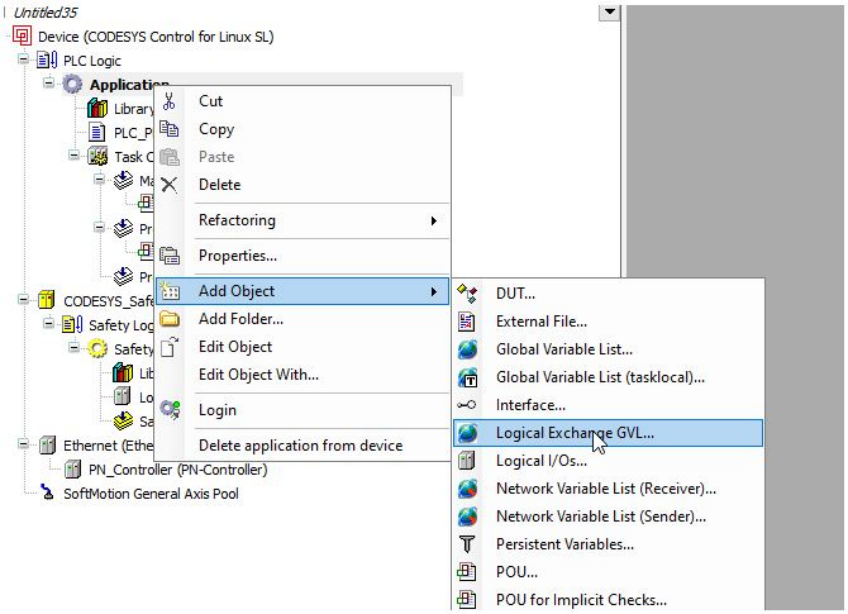
To set up data exchange between the safety and standard controller, follow these steps:

1. In the safety controller:
   - Navigate to `<Safety-Device> → Safety Logic → SafetyApp → Logical I/Os`.
   - Open the context menu and select the command **Add Logical Device**.
   - In the dialog, select an object from the **Logical Exchange Devices** node.



2. In the standard controller:
   - Navigate to `<Standard-Device> → PLC Logic → Application`.

- Open the context menu and select **Add Object** → **Logical Exchange GVL**.
- Give the object an appropriate name and add it to the project tree.



3. In the editor:
   - Link the logical exchange device with the GVL (Global Variable List).

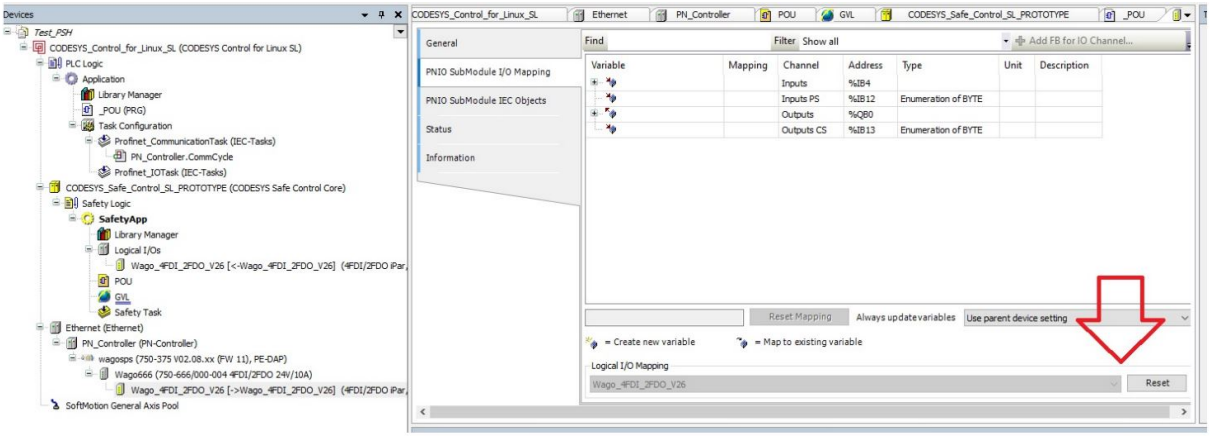
12.2 Exchange Fieldbus with Safe I/Os

The exchange of I/O data is also performed via logical devices. If the assignment is unambiguous (e.g., only one safety controller in the project tree), inserting the physical device automatically adds and links the logical device under `Logical I/Os` in the Safety Application.

The linkage is displayed:

- In the project tree, behind the name of both the physical and logical device as `[→]` or `[←]`.

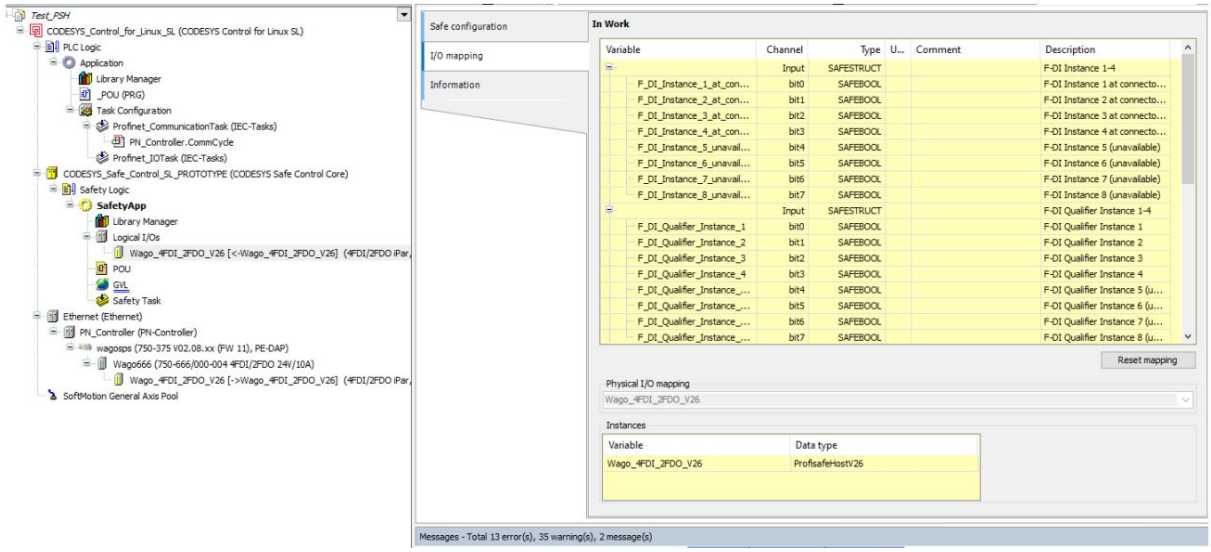
- In the Mapping Editor of the physical device.



In the Mapping Editor of the physical device, the linkage can be reset or re-established.

Product Marketing: Set up Debian + vPLC + vSafePLC from scratch.
CODESYS Project: For using CODESYS Safe Control – 33.

For the inserted logical device, a global function block (FB) instance is created in the Safety Application with the variable name and type corresponding to the logical device. The FB instance is uniquely defined in the Safety Mapping Editor of the logical device.



Note: When inserting, the logical device name may start with an underscore (_). However, the FB instance starts with `x_` (starting from SafetyVersion 4.2; previously, this caused a build error).

Fieldbuses supported for SIL3 with safe I/Os:

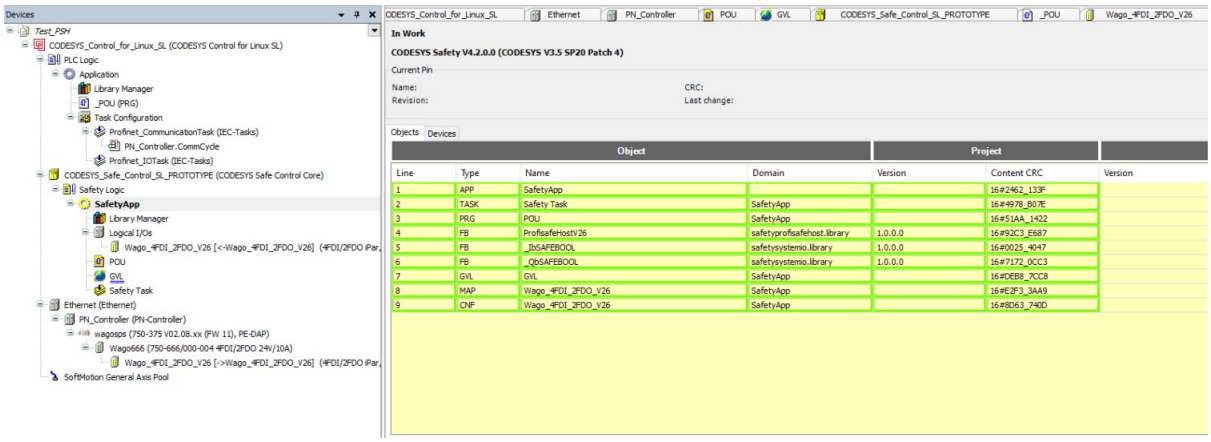
- PROFInet with PROFIsafe V2.4
- PROFInet with PROFIsafe V2.6 (only for Safe Control Core)
- EtherCAT with FSoE (planned for Safe Control Core)

12.3 PROFIsafe

Two versions are available: PROFIsafe V2.4 and V2.6. New F-Device devices must only support V2.6. The physical device defines the version, which is not switchable.

12.4 Safety Application

A list of objects.



Each object corresponds to a “yellow” editor. Every object editor stores the data in a 1:1 “Interpreter” format, which is loaded into the Safety Runtime during download. Additionally, the Safety Runtime expects information from the Safety Checker, which is executed during the “Build” or before the “Login with Download”.

12.5 Global Variable List (GVL)

Note: No namespace is used for the GVL. GVL variables are accessed via `VAR EXTERNAL`. This is not an issue since variable names in a Safety Application must be unique.

12.6 IO Mapping

Defines the variables mapped into the Safety Application. Variable definitions are implicitly of type Global and are used in the Safety Application’s POUs with `VAR_EXTERNAL`.

In the Mapping Editor, the linkage to the physical device and the global function block (FB) instance of the corresponding IO stack is displayed.

Note: In the Mapping Editor, a byte must be defined either as a single byte or as individual bits; mixing both is not allowed.

12.7 IO Configuration (F-Parameter)

Settings corresponding to the assigned physical safe I/O device. For PROFIsafe or FSoE F-devices, the F-Destination Address is set via DIP switches on the IO or assigned using a device-manufacturer-specific tool.

12.8 POU, FB

Two different object types:

Product Marketing: Set up Debian + vPLC + vSafePLC from scratch.
CODESYS Project: For using CODESYS Safe Control – 35.

- **Basic**: Only FBs with combinatorial logic of boolean operations using `AND` or `OR`. No `NOT`!
- **Extended**: Full range of supported operators (see Toolbox).

Note: Safety programming strictly distinguishes between logical and numerical data types. Numerical operations cannot be performed on logical data types, and logical operations cannot be performed on numerical data types.

- **Logical Data Types**: `BOOL`, `BYTE`, `WORD`, `DWORD`, etc., with operations `AND`, `OR`, `NOT`.
- **Numerical Data Types**: `INT`, `UINT`, `DINT`, etc., with operations `ADD`, `SUB`, `DIV`, `MUL`, `LE`, `GT`, etc.

Structures, arrays, enums, and pointers are not supported!

12.9 Safety Task

Only one cyclic task with a specified cycle time, default 10 ms. The selected programs in the list are executed in the order of the task list. The execution order can be controlled using `Up`, `Down`, or selection (`All`, `None`).

13 Diagnosis

13.1 Exchange Between Safety and Standard

Configuration ID via the module list of exchanged devices, `IoDrvSafetySP`. The device tree indicates whether the ID matches at the Safety Controller.

Note: Changes to the PROFIsafe configuration, e.g., `F_WD_Time`, also require a download of the Standard. The CRC of the F-Parameters is part of the Configuration ID, and a mismatch is displayed to the user in the project tree.

13.2 IO-Stack Instance

See online help: `https://content.helpme-codesys.com/en/CODESYS\%20Safety\%20Extension/sil3_field_buses.html`.

Meaning of the FB output Diagnosis:

- `0x8xxx`: OK, with `xxx` indicating status.
- `0xC0xx`: Initialization error `XX`, usually with a logbook entry, application terminated.
- `0xC1xx`: Self-detected error `XX`.
- `0xC2xx`: Error `XX` detected by the F-Device.

13.2.1 PROFIsafe

Different behavior regarding FB Diagnosis Output:

- **V2.4**: Diagnosis word is overwritten by higher-priority diagnosis, priority from `16#C0xx` to `16#C2xx`.
- **V2.6**: The first detected error remains as the diagnosis word at the FB output until acknowledged.

Product Marketing: Set up Debian + vPLC + vSafePLC from scratch.
CODESYS Project: For using CODESYS Safe Control – 36.

Note: The F-Host recognizes only two errors: Timeout or CRC error. The CRC error includes all possible variants of initialization and communication errors.

PROFIsafe Diagnosis on Standard (only Safe Control Core: F-Host Outputs are transferred to Standard). Example access:

Declaration:

```
uiID: UDINT;
FHostState: ProfinetCommon.F_HostStatus;
```

Implementation:

```
(* Next code line requires lib IoDrvProfinetBase! *)
uiID := IoDrvProfinetBase.GetID(Wago666);
ProfiNetCommon.GetFHostStatus(ID := uiID, F_Status := FHostState);
```

14 Download - Boot Application

Note: Handling of Download and Boot Application differs from Standard. With a logout, a running application is unloaded and no longer executed. If a Boot Application exists on the controller, it may or may not be started.

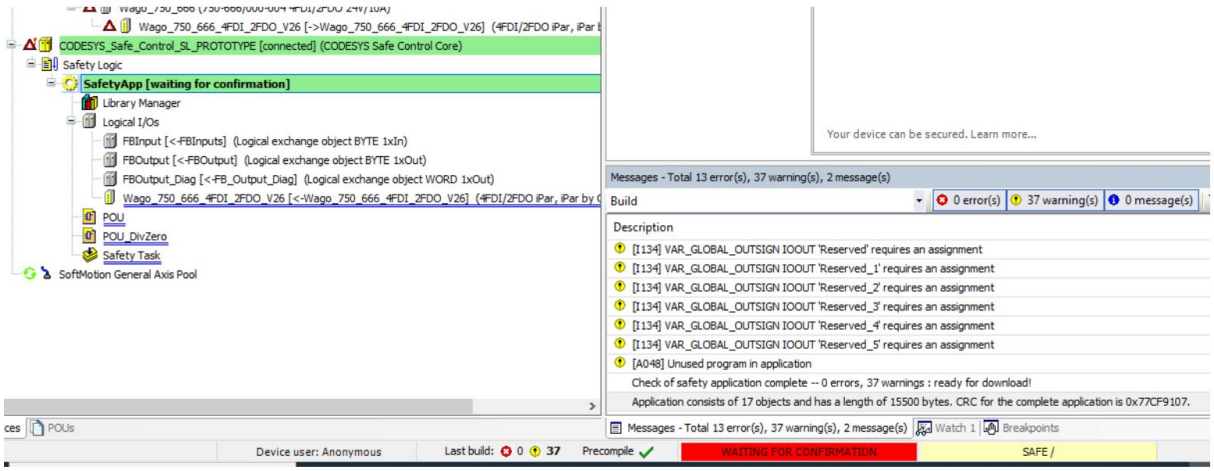
For Safety Controllers, the Login dialog requires a decision on whether to perform only a Download or a Download with Boot Application creation. Download with Boot Application combines two commands (Login and Create Boot Application), which can generally be executed independently.

A Download with Boot Application takes longer since the source code is loaded to the controller twice: once for the Download and once for the Boot Application. Therefore, it is recommended to perform a Download first and create a Boot Application only when the application is stable.

14.1 Start of the Boot Application for SafeControl Core

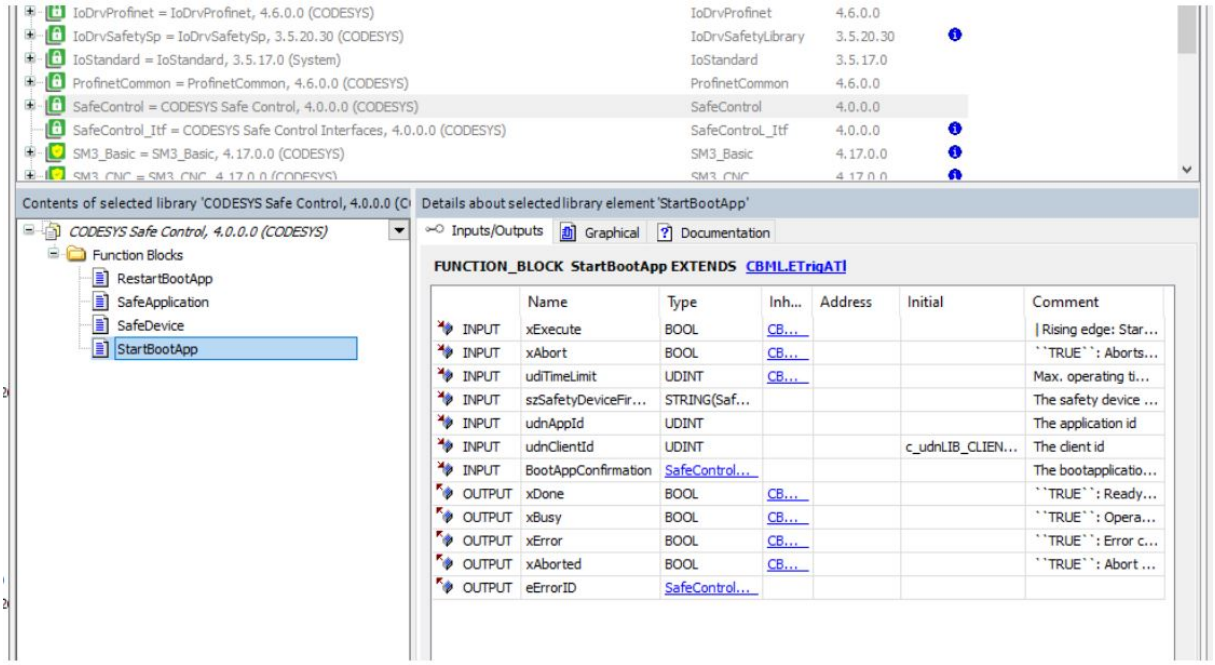
The start of the Boot Application differs between SIL3 OEM controllers and Safe Control Core. For Safe Control Core, the user must confirm the start of the Boot Application. This is supported by the CODESYS Safe Control Lib (see Diagnosis on Standard).

Starting from Safety Extension 4.3.0.0, the status is displayed in the project tree and the active application status (see screenshot). For Safety Extension versions earlier than 4.3.0.0, no display is available in either the tree or the status, and both fields are empty.



15 Diagnosis on Standard

The CODESYS Safe Control Service Package includes the `CODESYS Safe Control` library, providing the following function blocks (FBs):



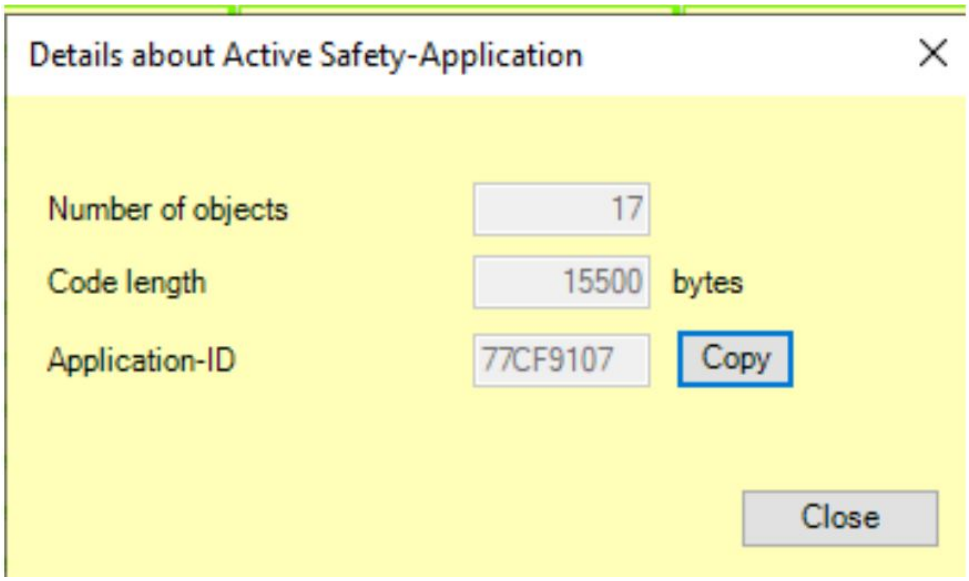
- **FB SafeDevice**: Provides information and diagnostics about the device. **Note**: Information regarding external timers is currently not meaningfully usable (see Jira SIL3SL-725).
- **FB SafeApplication**: Provides information about the loaded application or the start of the Boot Application.

- **FB StartBootApp**: FB for confirming the start of the loaded Boot Application. Starting from Safety Extension 4.3.0.0, this additional state is displayed in the project tree and in the status of the active application. For versions earlier than 4.2.0.0, no application status is displayed.

To start the Boot Application, the current value of the FB output `SafeApplication.BootAppConfirmatio` must be specified, along with a Client ID defined in the runtime's CFG file. The default for the Standard Application is `16#ED387206`.

Product Marketing: Set up Debian + vPLC + vSafePLC from scratch.
CODESYS Project: For using CODESYS Safe Control – 38.

ApplicationId: Starting from version 4.3.0.0, available via the command `Build` → `Show Details about Active Safety Application`.



Example Implementation for Boot Application Start Confirmation:

Declaration:

```
PROGRAM StartSafetyBA
VAR
xStartBA : BOOL := FALSE;
xRestart : BOOL := FALSE;

fbSafeApp : SafeControl.SafeApplication;
fbSafeDev : SafeControl.SafeDevice;

fbStartBA : SafeControl.StartBootApp;

fbRestart: SafeControl.RestartBootApp;

END_VAR
VAR CONSTANT
c_udnClientId : UDINT := 16#ED387206;     // The given client id of PLC
END_VAR
```

Implementation:

```
fbSafeApp(xEnable := TRUE);
fbSafeDev(xEnable := TRUE);
fbStartBA(xExecute := (fbSafeApp.xBootAppConfirmationRequested AND xStartBA),
szSafetyDeviceFirmware := fbSafeDev.szSafetyDeviceFirmware,
udnAppId := fbSafeApp.AppInfo.udnAppId,
udnClientId := c_udnClientId,
BootAppConfirmation := fbSafeApp.BootAppConfirmation);
// For automatic test execution FB with the restart of the bootapplication is
    also implemented
fbRestart(xExecute := xRestart);
```

16 Device Editors

- Communication
- Safety Online Information
- Status
- Information

16.1 Further Links

- Online Help for Virtual Safe Control: `https://content.helpme-codesys.com/de/CODESYS%20Control/_rtsl_scenario_safe_house.html`
- Docker Installation: `https://docs.docker.com/engine/install/debian/`

17 Use Case: Virtual Safe Emergency Stop

17.1 Opening the Project

To begin, open the project **vSafeTest_sim**, located in the appendix.

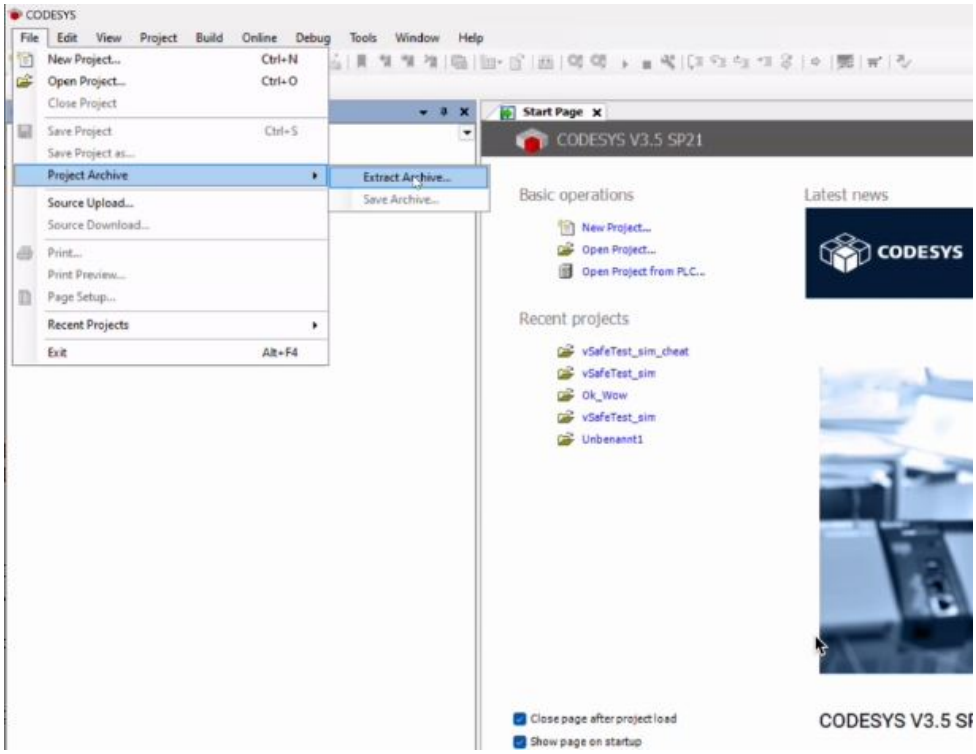


Figure 11: Project loading screen

After a brief loading period, a project environment popup will appear. Click **OK** in the bottom-right corner to continue.

17.2 Establishing Device Connections

1. Connect to the Device:

- Double-click on **Device** in the project tree.
- Ensure the correct **VGateway** is selected. If not, select it from the list.
- Click **Scan Network**, then select the appropriate PLC.
- Verify that both the device and gateway display green status indicators, confirming a connection to the Linux PC.

2. Connect to Safe Control:

- Double-click on **CODESYS Safe Control for Linux SL** in the project tree.
- Ensure the correct **VGateway** is selected. If not, select it from the list.
- Click **Scan Network**, then select the appropriate PLC.

- Verify that both the Safe Control and gateway display green status indicators, confirming a connection to the Linux PC.

3. Log into the Systems:

   - Navigate to the **Application** section under **Device** and click **Login** to connect to the standard system. Use the same credentials as previously used (see Section 11.6 for Safe POU login details).
   - Right-click on **SafetyApp** and select **Login** to connect to the safe system too.

4. Verify Connection Status:

   - Ensure both devices display green status indicators.
   - Confirm the central status bar shows both system **RUN**

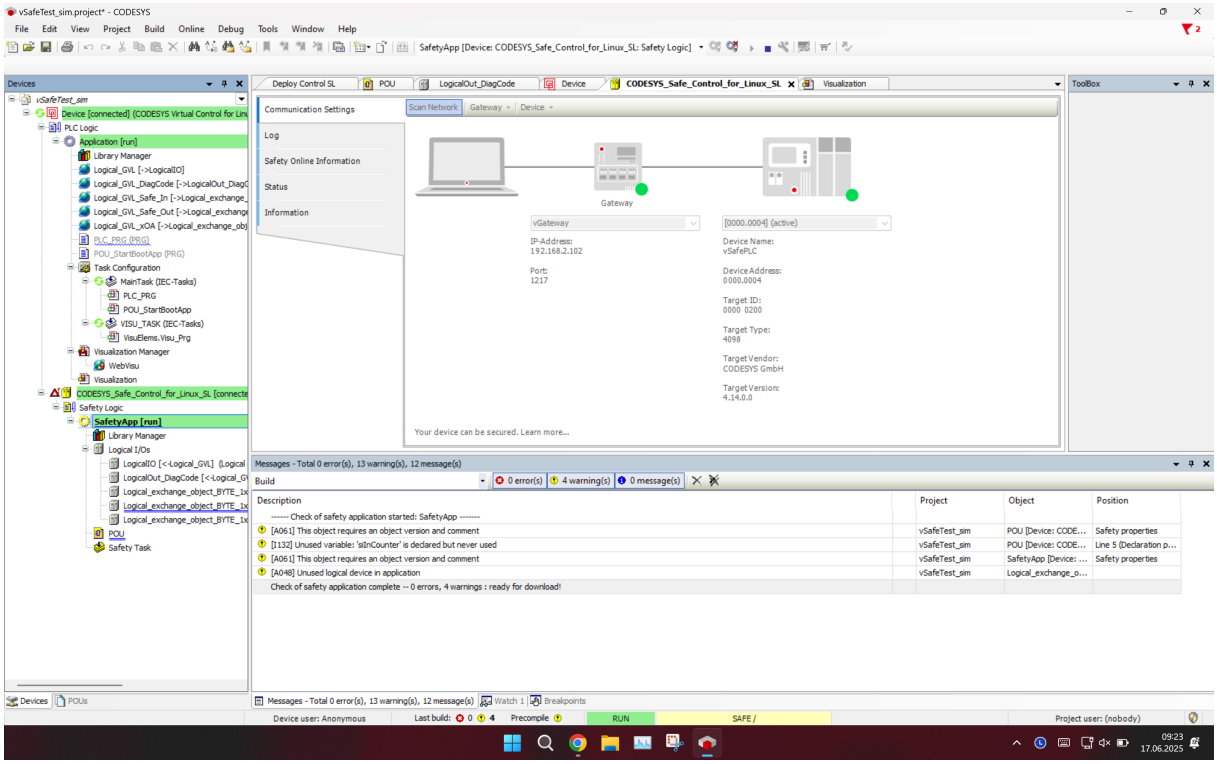


Figure 12: Successful connection with green status indicators

17.3 Handling Version Compatibility Issues

If version compatibility issues arise:

- Right-click on the affected device and select **Update Device**. Choose **Linux SL 4.15.0.0** from the popup window.
- For Safe Control, select **CODESYS Safe Control**, click **Update Device**, and choose version **4.15.0.0**.

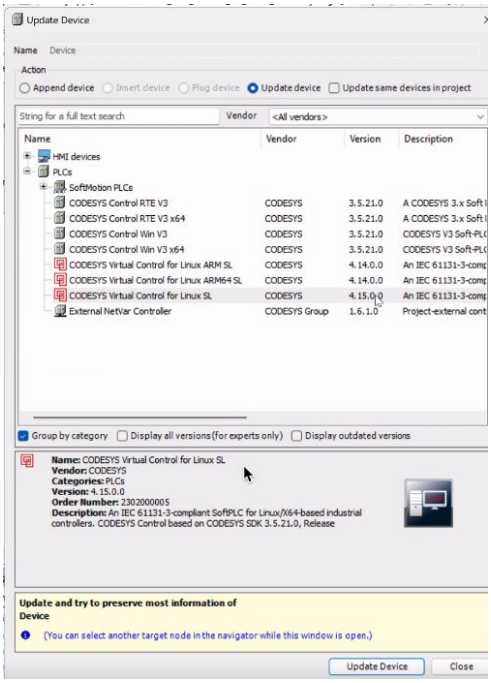


Figure 13: Device version update

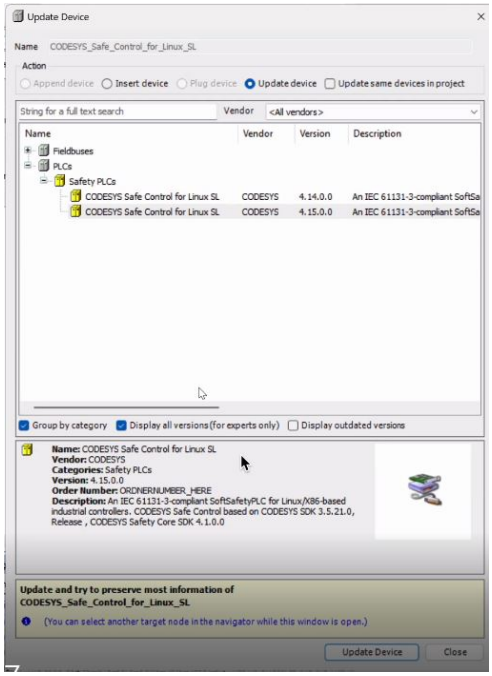


Figure 14: Safe Control version update

17.4 Testing the Emergency Stop

To test the emergency stop (*Not-Aus*):

- Open the **Visualization** view.
- Toggle the switch labeled **Logical GVL Safe Out** to simulate an emergency stop.
- Verify that the **EStop** LED indicator lights up, confirming the emergency stop simulation was triggered correctly.

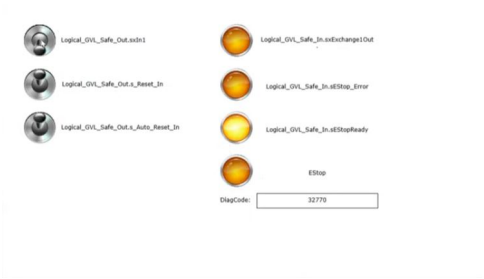


Figure 15: Flip the lever

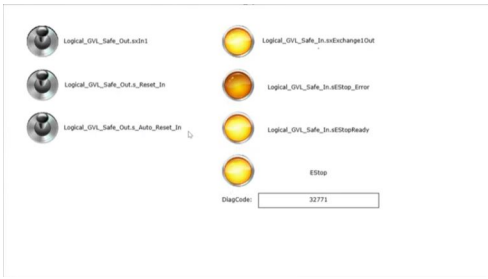


Figure 16: EStop LED lights up

This completes the setup and validation of the virtual safety mechanism, ensuring both visual feedback and system response function as intended.