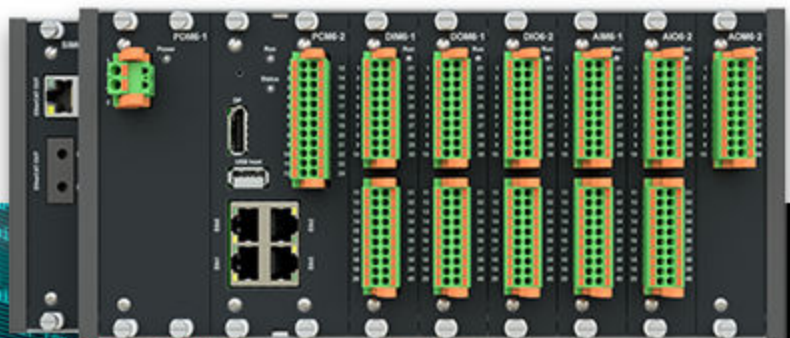


iE 650 PLC

CODESYS guide



Improve
Tomorrow



1. Introduction	4
1.1 CODESYS Overview.....	4
1.2 Firmware and Software version used in this guide	4
1.3 Legal information	4
2. iE 650 PLC Programming Environment	5
2.1 Starting a New CODESYS Project.....	5
2.2 Configure the Device	9
2.3 Login and User Management Setup	12
2.4 CODESYS Library Management	16
2.5 Add EtherCAT bus	17
2.6 Task Configuration	20
2.7 Mission watchdog	21
2.8 Setting the I/O Default State	22
2.9 Controller Load Monitoring	23
2.10 Automatic scanning equipment	24
2.11 Persistent variable	26
2.12 Creating an HMI	28
3. iE 650 PLC program configuration	38
3.1 PCM6.2 program configuration	38
3.1.1 Digital program configuration.....	38
3.1.2 Serial Programming	39
3.1.3 CANopen program configuration	43
3.2 DIO6.1 - Configuration example.....	52
3.3 AIO 6.1 - Configuration example.....	54
3.3.1 AIO 6.1 – Output startup configuration	54
3.3.2 AIO 6.1 – Input startup configuration.....	55
3.3.3 AIO6.1 - Variable mapping.....	56
3.3.4 CODESYS program example.....	57
3.3.5 Scaling example.....	58
3.4 TIM6.1 Program configuration	58
3.4.1 3-Wire PT100.....	59
3.5 IFM 6.1 Program Configuration	60
3.5.1 IFM6.1 - Modbus RTU Client	60
3.6 IFM 6.2 Program Configuration	67
3.6.1 SSI program configuration.....	67
3.6.2 FI program configuration	69
3.7 SIM6.4 program configuration.....	70

3.7.1 EtherCAT redundancy70

3.7.2 Two individual EtherCAT circuits72

3.8 SIM6.5 program configuration.....73

3.8.1 EtherCAT redundancy73

3.8.2 Two individual EtherCAT circuits.....75

1. Introduction

1.1 CODESYS Overview

The iE 650 PLC programming environment is provided by the CODESYS IDE. CODESYS is a powerful PLC programming tool that supports the IEC 61131-3 standard and offers six programming languages:

- IL (Instruction List)
- ST (Structured Text)
- FBD (Function Block Diagram)
- LD (Ladder Diagram)
- CFC (Continuous Function Chart)
- SFC (Sequential Function Chart)

1.2 Firmware and Software version used in this guide

- iE 650 PLC Firmware: 2.0.16.1
- CODESYS TSP Version: 1.3.8.0
- CODESYS IDE: CODESYS 3.5 SP21 Patch 4.

1.3 Legal information

Disclaimer

DEIF A/S reserves the right to change any of the contents of this document without prior notice. The English version of this document always contains the most recent and up-to-date information about the product. DEIF does not take responsibility for the accuracy of translations, and translations might not be updated at the same time as the English document. If there is a discrepancy, the English version prevails.

Trademarks

DEIF is a trademark of DEIF A/S.

CODESYS® is a trademark of CODESYS GmbH.

All trademarks are the properties of their respective owners.

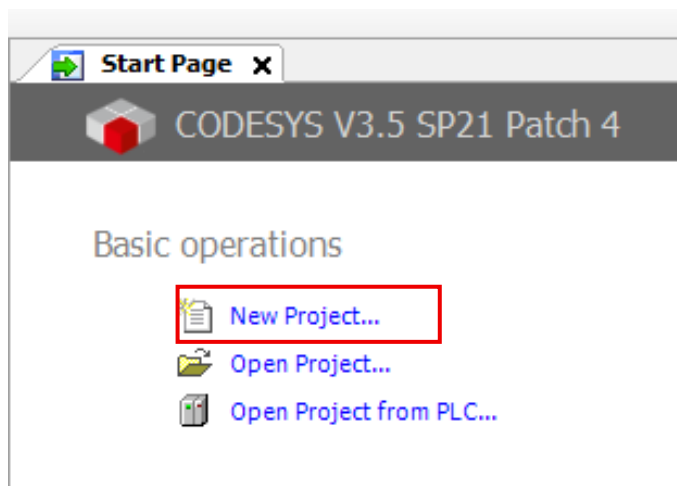
Copyright

©Copyright DEIF A/S. All rights reserved.

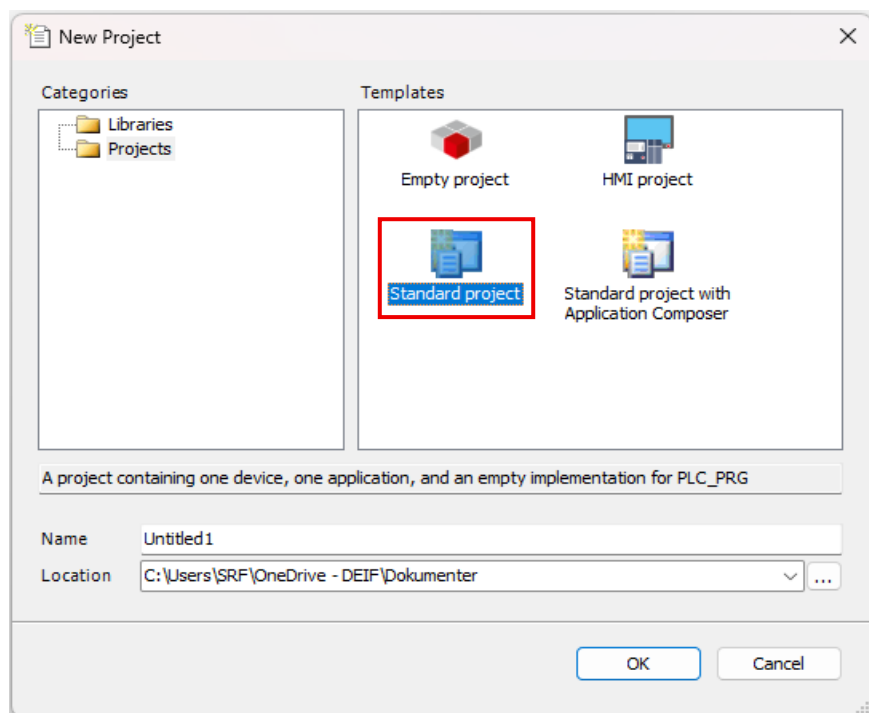
2. iE 650 PLC Programming Environment

2.1 Starting a New CODESYS Project

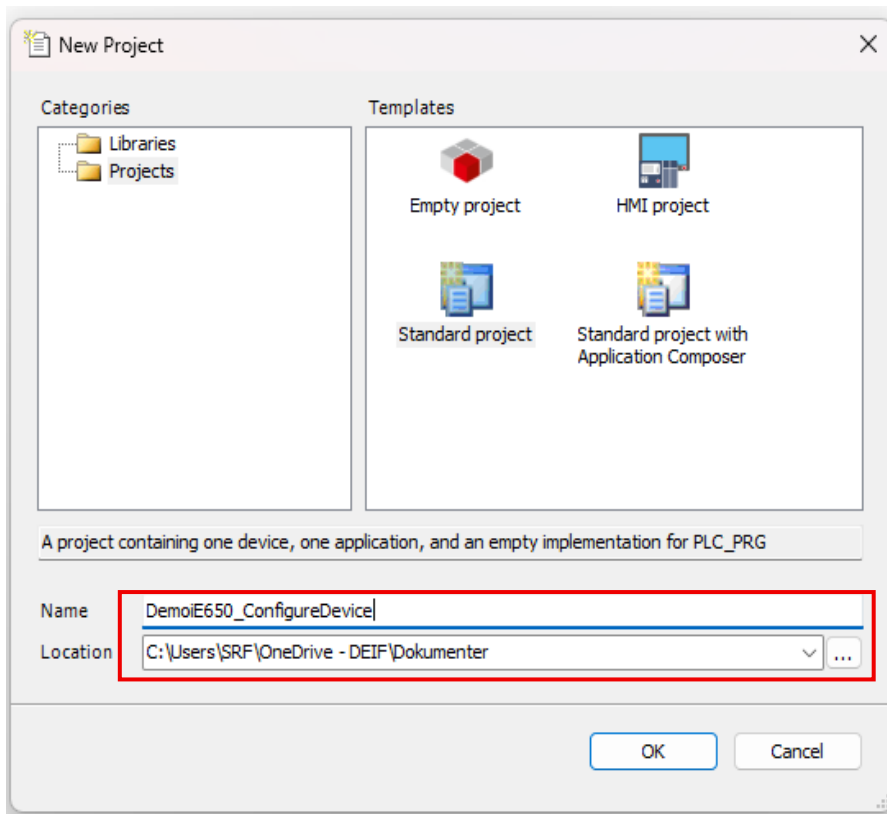
When opening CODESYS, select **New Project**



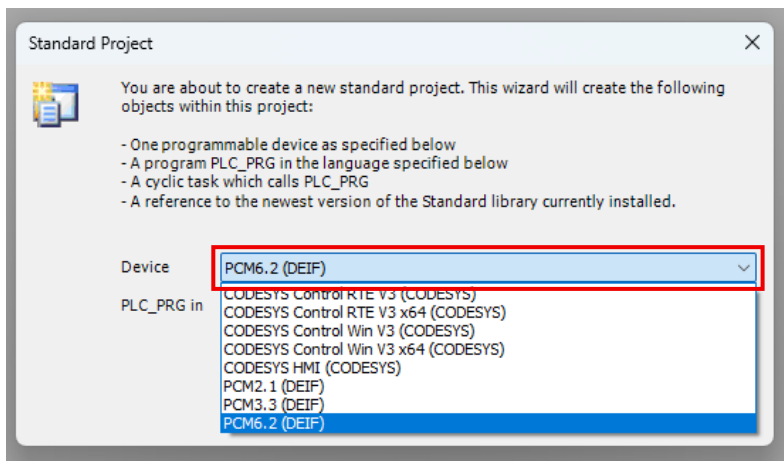
Choose **Standard Project**.



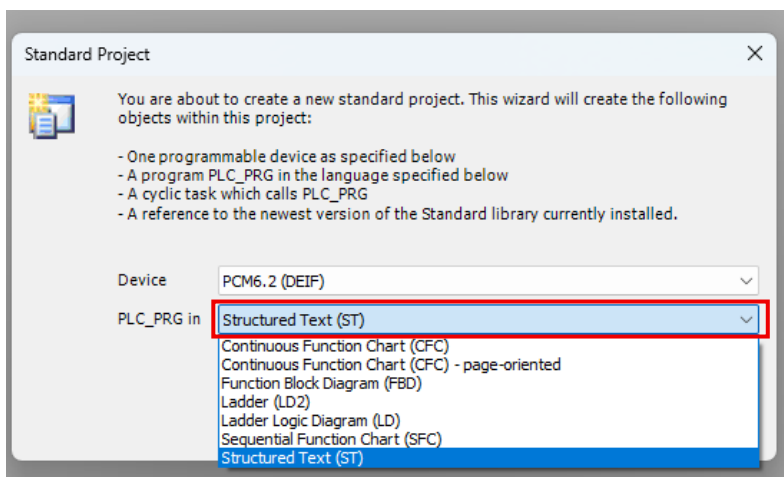
Enter a **descriptive project name** and choose a **preferred save location**.



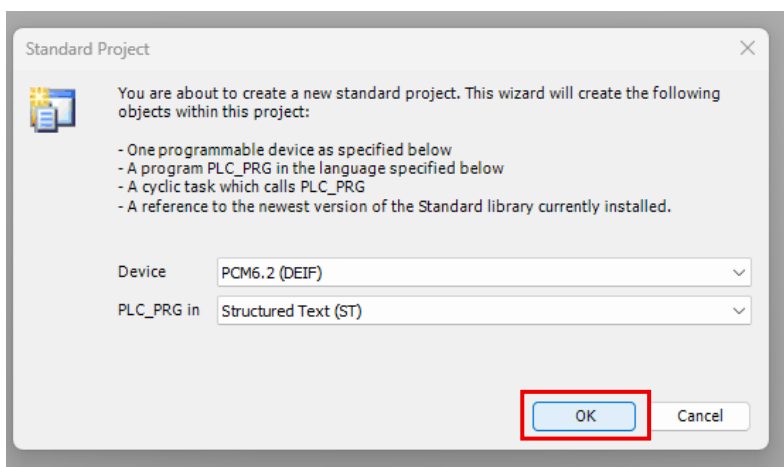
Under **Device**, select **PCM6.2** for the iE 650 PLC.



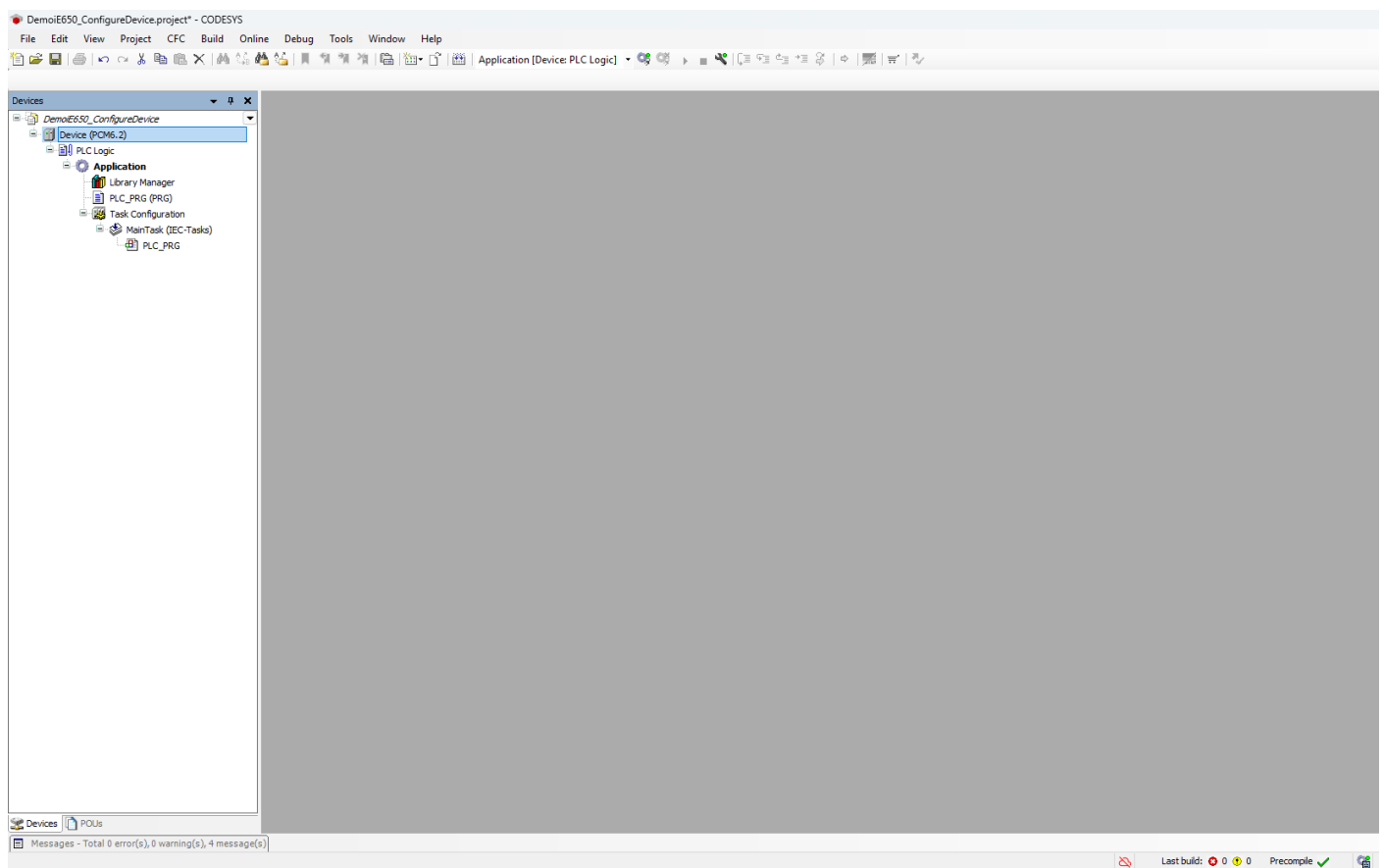
Select your preferred programming language.



Click **OK**.

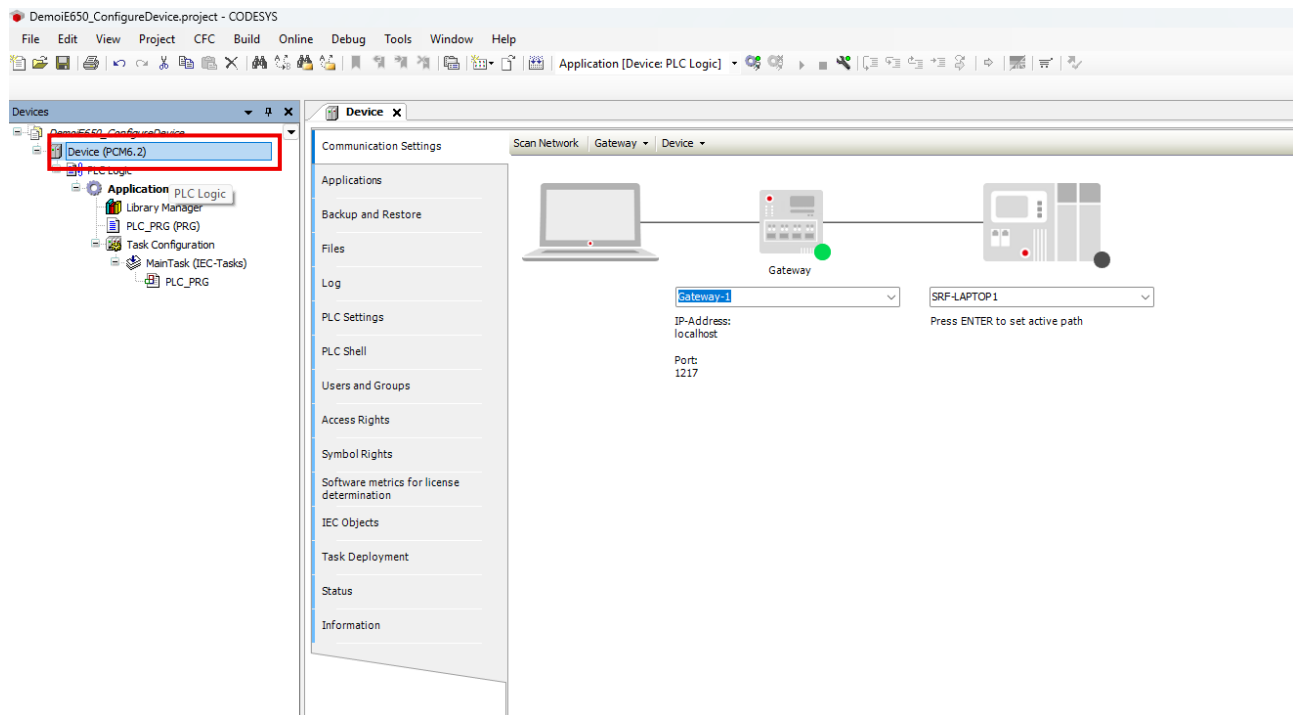


CODESYS will create and open your project.
Your CODESYS Project will appear similar to the example shown below.

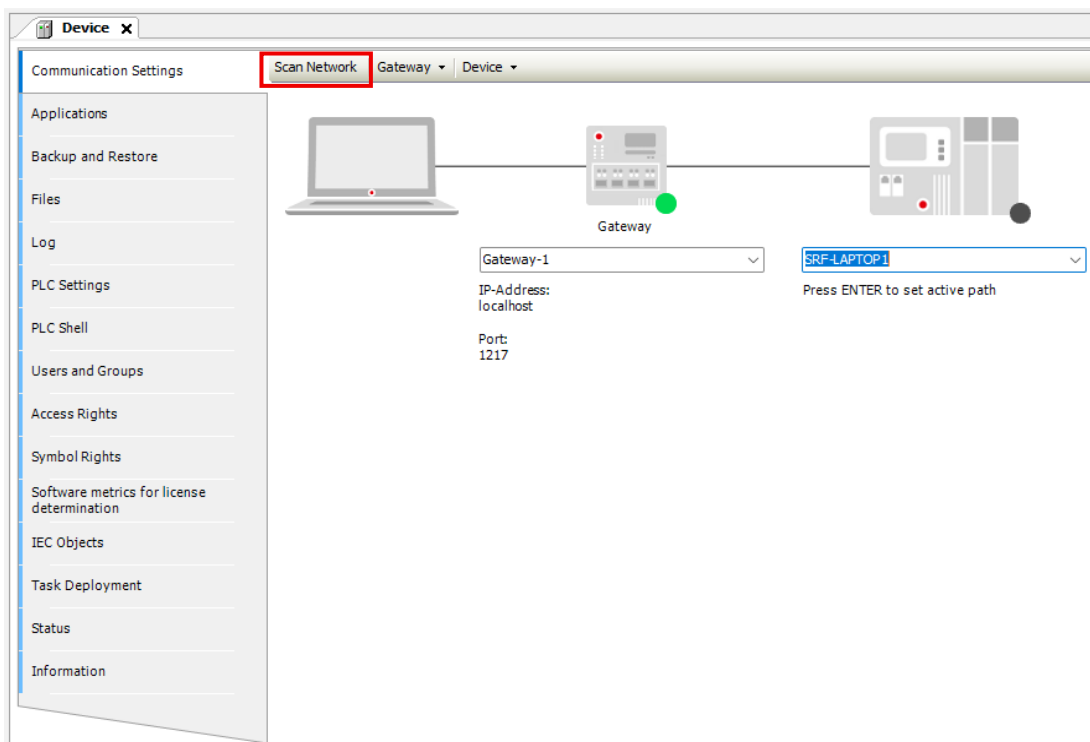


2.2 Configure the Device

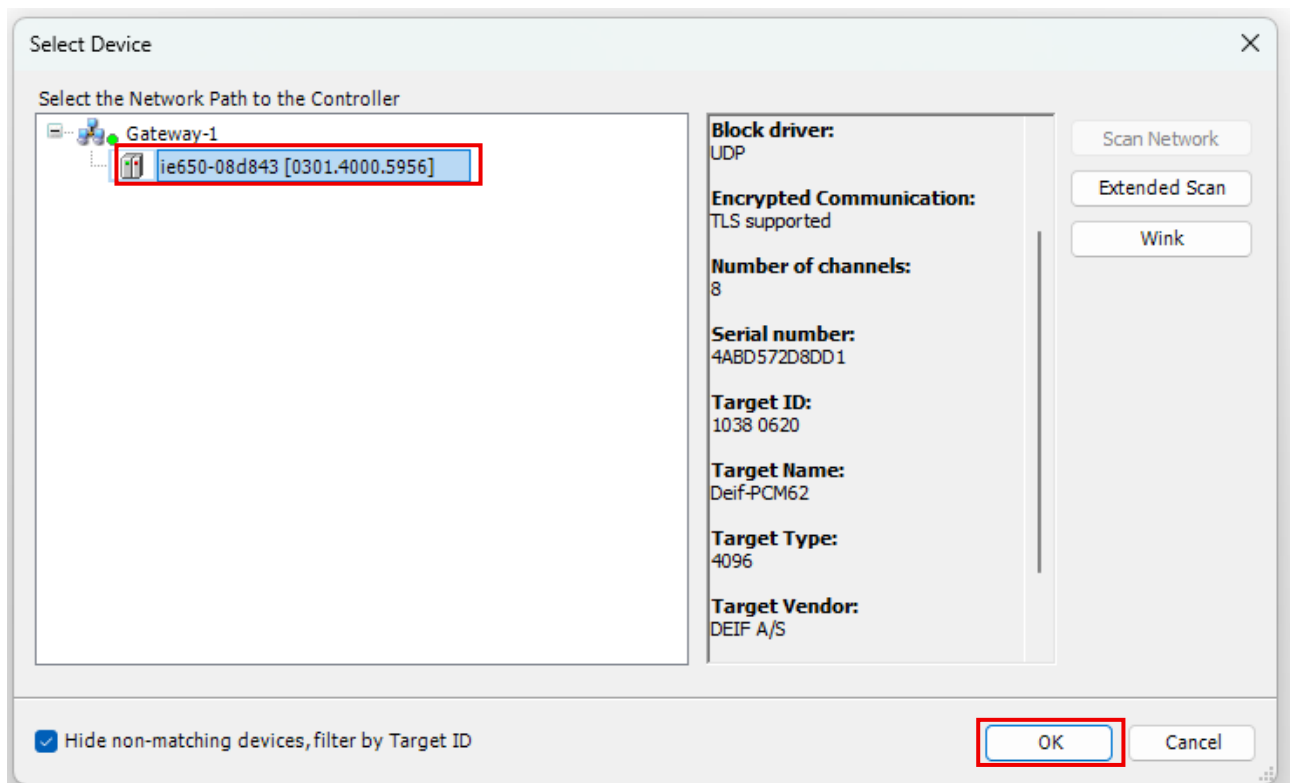
To configure the device, double click the **Device** entry in the device tree.



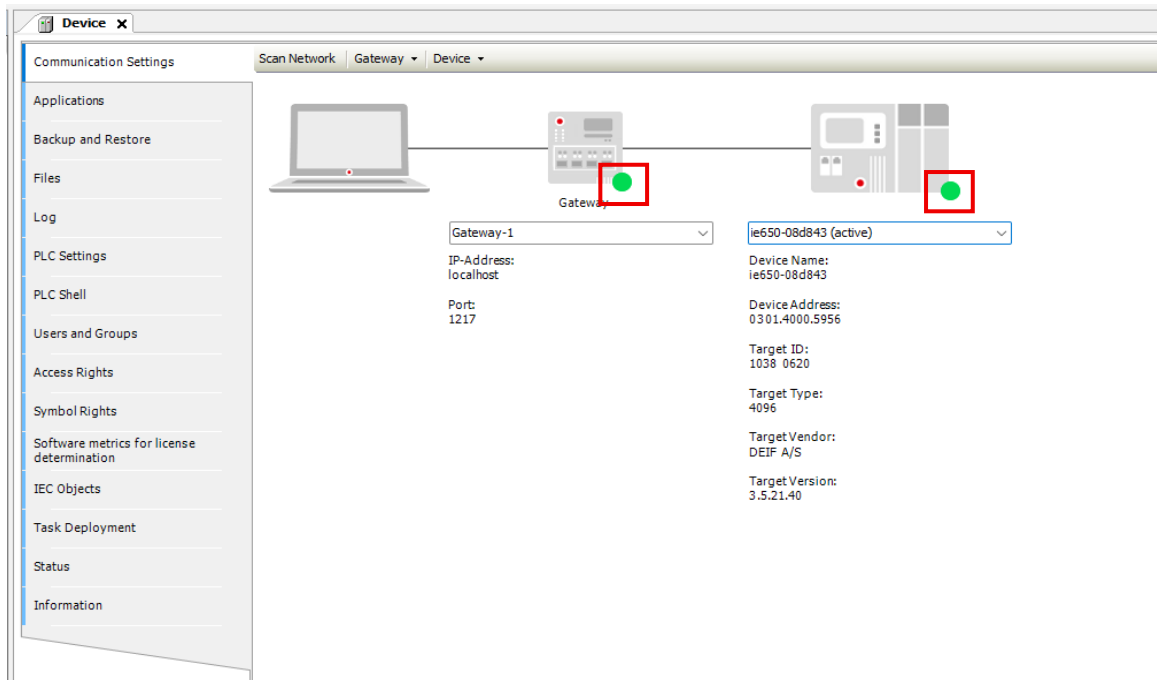
Click **Scan Network** to search for your PLC on the network.



Select your device from the list and click **OK**.

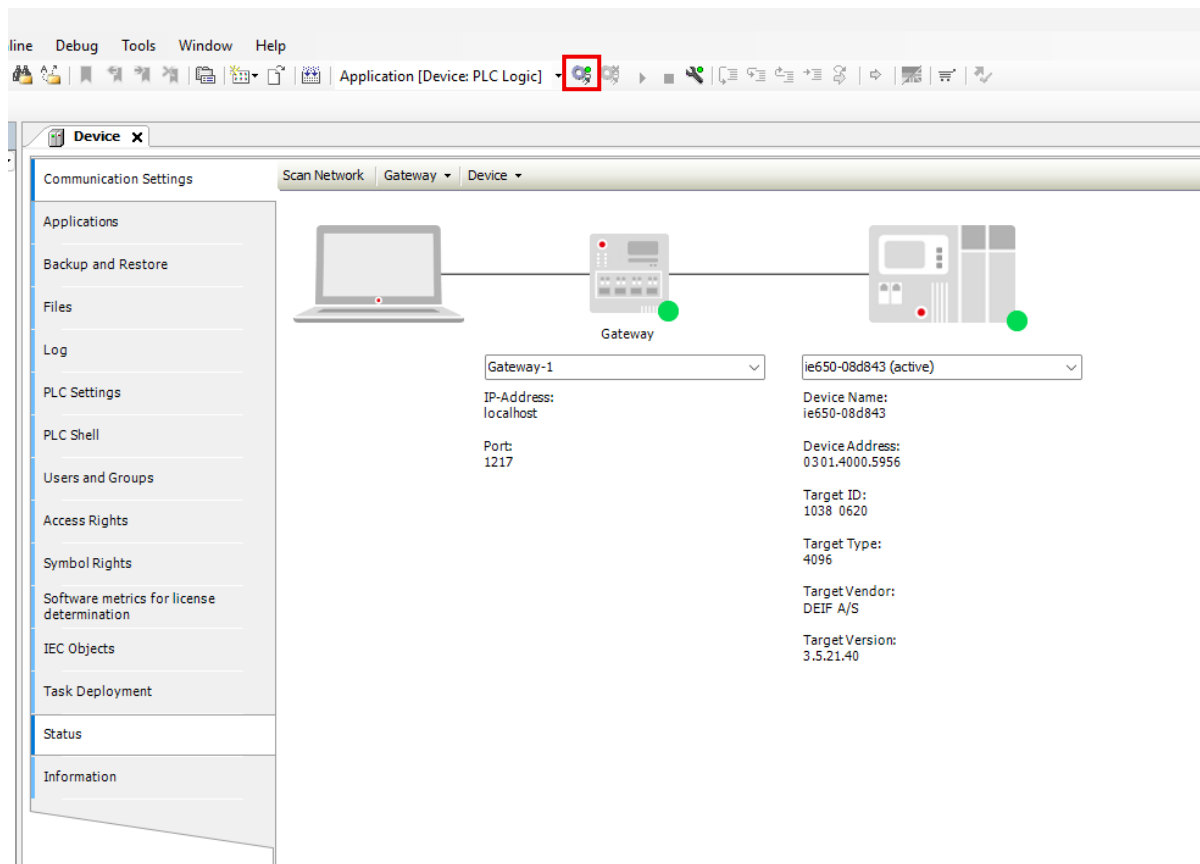


When both the gateway and the device appear **green**, communication to the PLC is established.

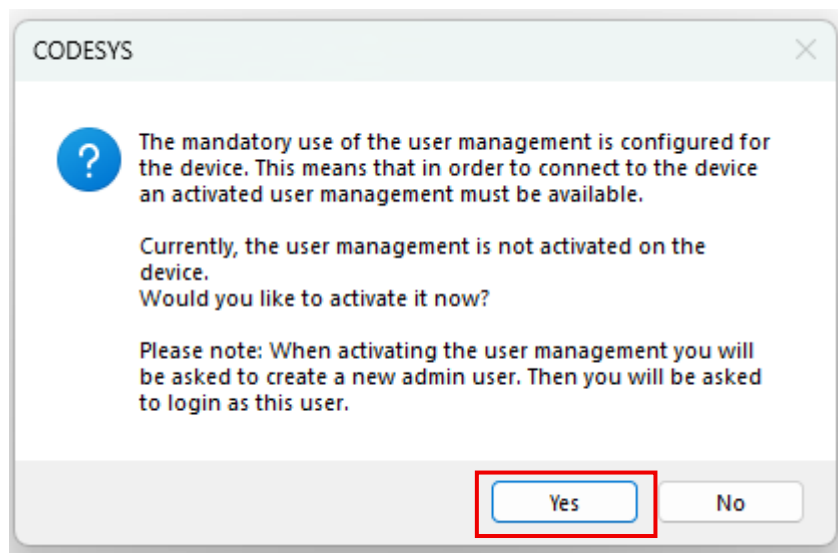


2.3 Login and User Management Setup

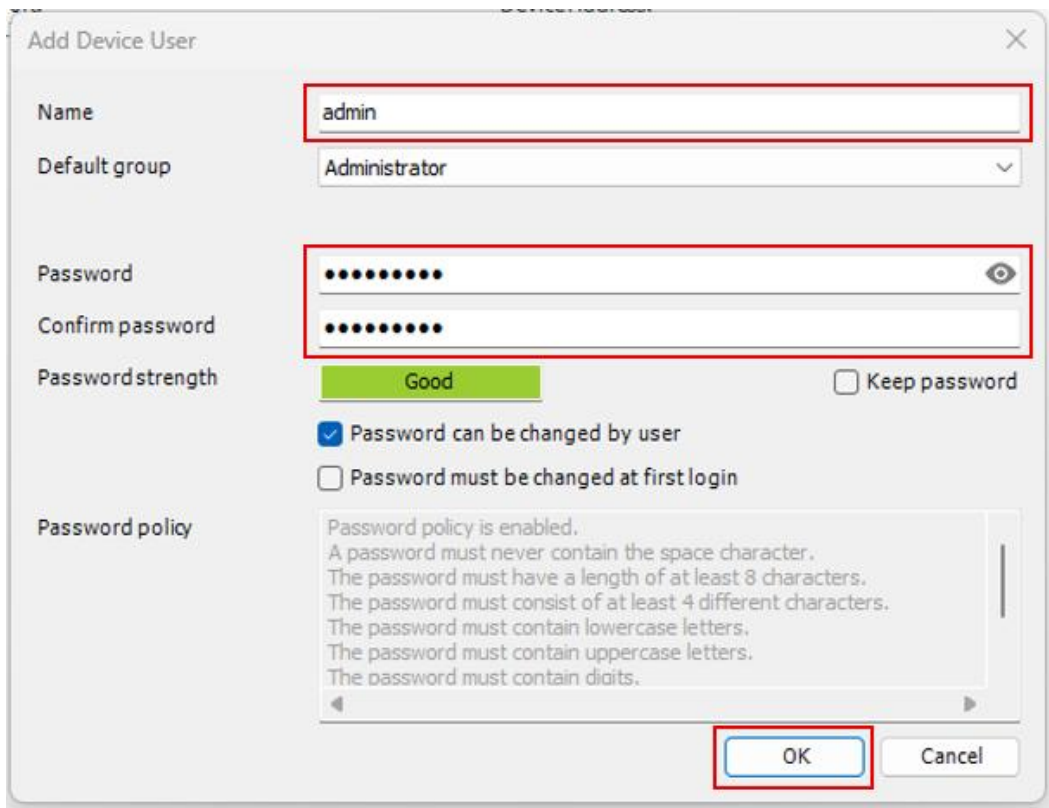
Click the **Login** icon to log in to the PLC.



On the first login attempt, CODESYS will prompt you to activate user management. Click **Yes**

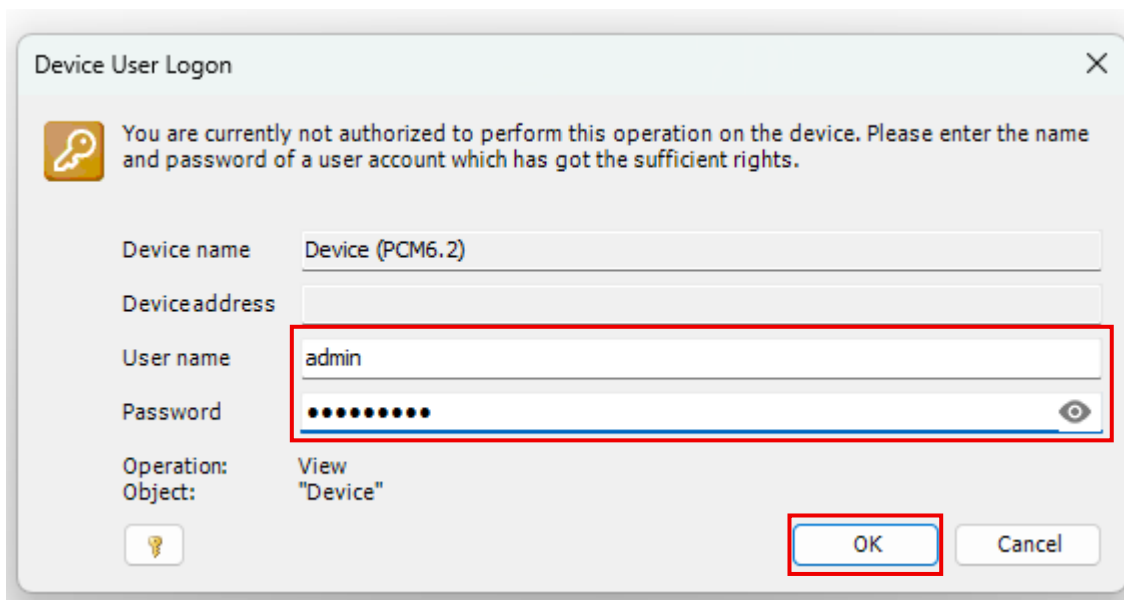


Create a device user by entering a username and password. Click **OK**.



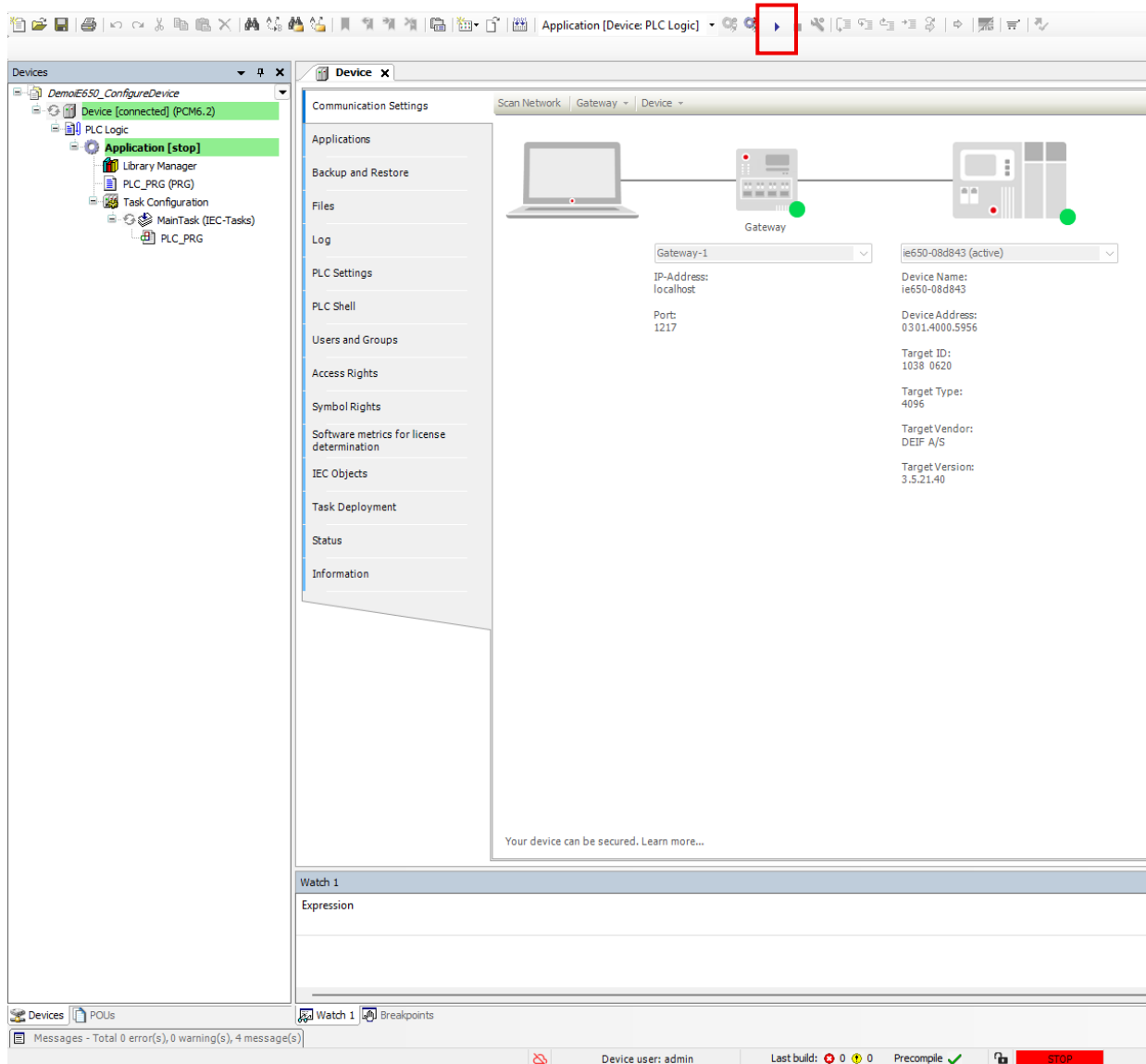
The 'Add Device User' dialog box is shown. It has a title bar with a close button. The fields are: 'Name' with 'admin' entered, 'Default group' with 'Administrator' selected, 'Password' and 'Confirm password' both masked with dots, 'Password strength' showing 'Good', and checkboxes for 'Keep password' (unchecked), 'Password can be changed by user' (checked), and 'Password must be changed at first login' (unchecked). A 'Password policy' section lists rules: no space, at least 8 characters, at least 4 different characters, lowercase, uppercase, and digits. At the bottom are 'OK' and 'Cancel' buttons.

Log in by using the newly created user credentials. Click **OK**.

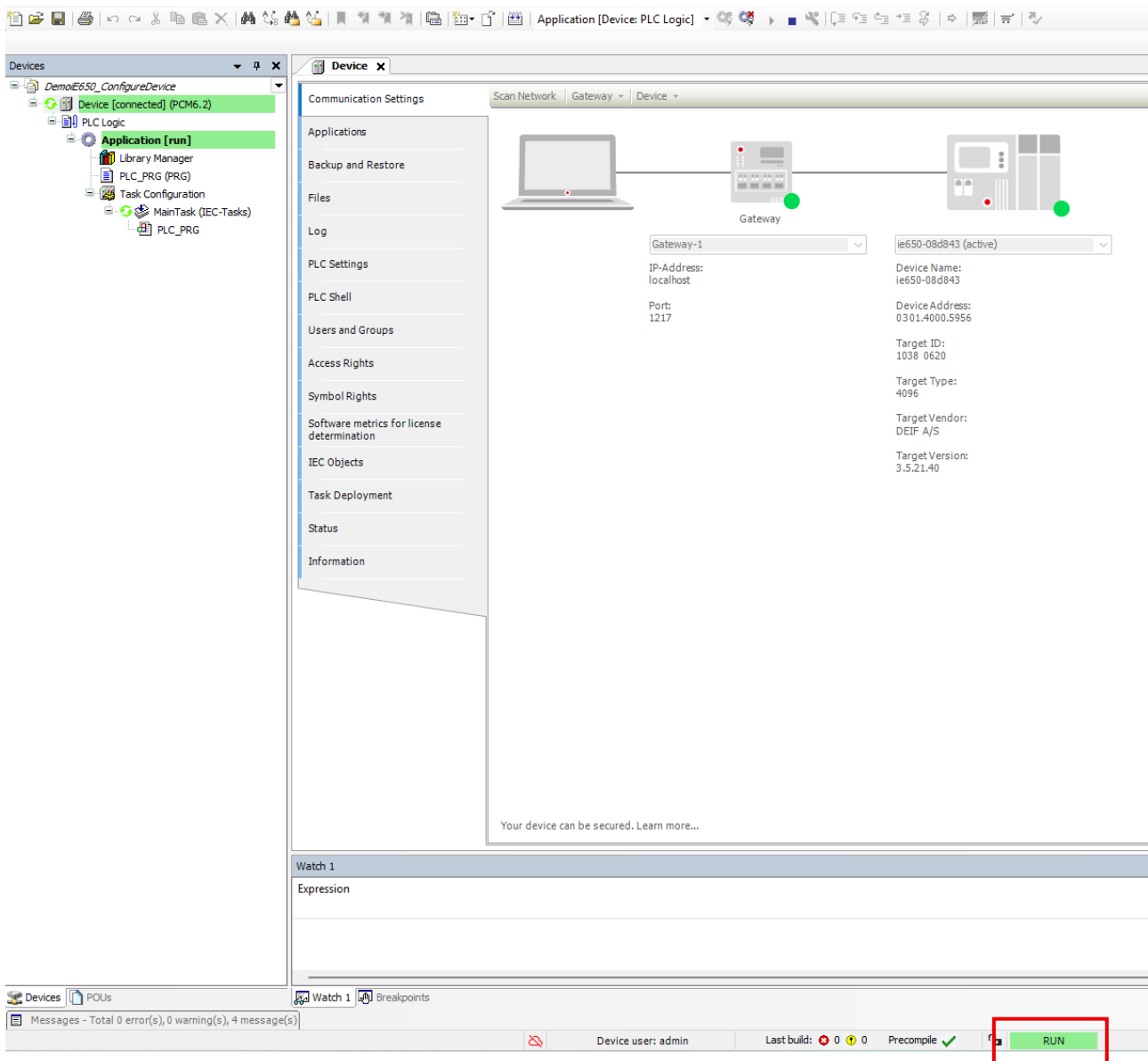


The 'Device User Logon' dialog box is shown. It has a title bar with a close button. A message icon and text state: 'You are currently not authorized to perform this operation on the device. Please enter the name and password of a user account which has got the sufficient rights.' Fields include: 'Device name' with 'Device (PCM6.2)', 'Device address' (empty), 'User name' with 'admin', and 'Password' masked with dots. 'Operation:' is 'View' and 'Object:' is '"Device"'. At the bottom are 'OK' and 'Cancel' buttons.

Once you are logged in, start the application by clicking the **Play** icon.

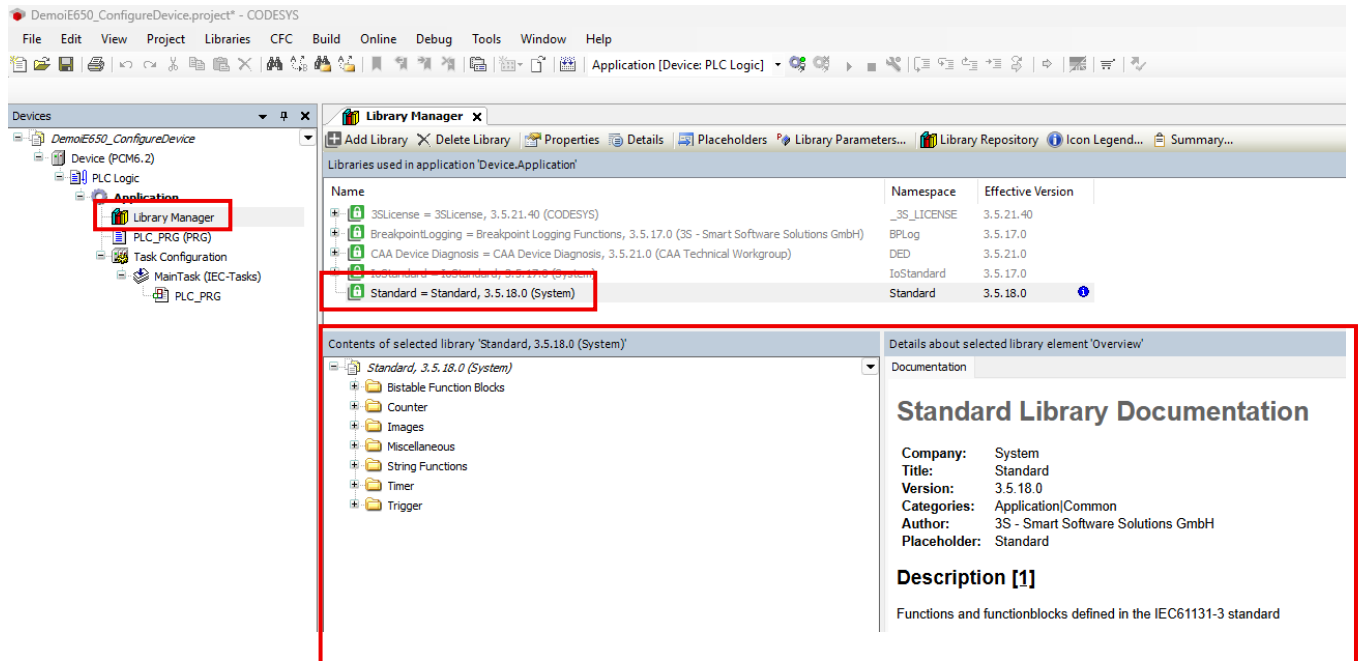


The status bar at the bottom will change to **RUN**, indicating that the PLC application is active.



2.4 CODESYS Library Management

CODESYS standard libraries and DEIF libraries are managed through the Library Manager, where they can be added to the project and called in the program.

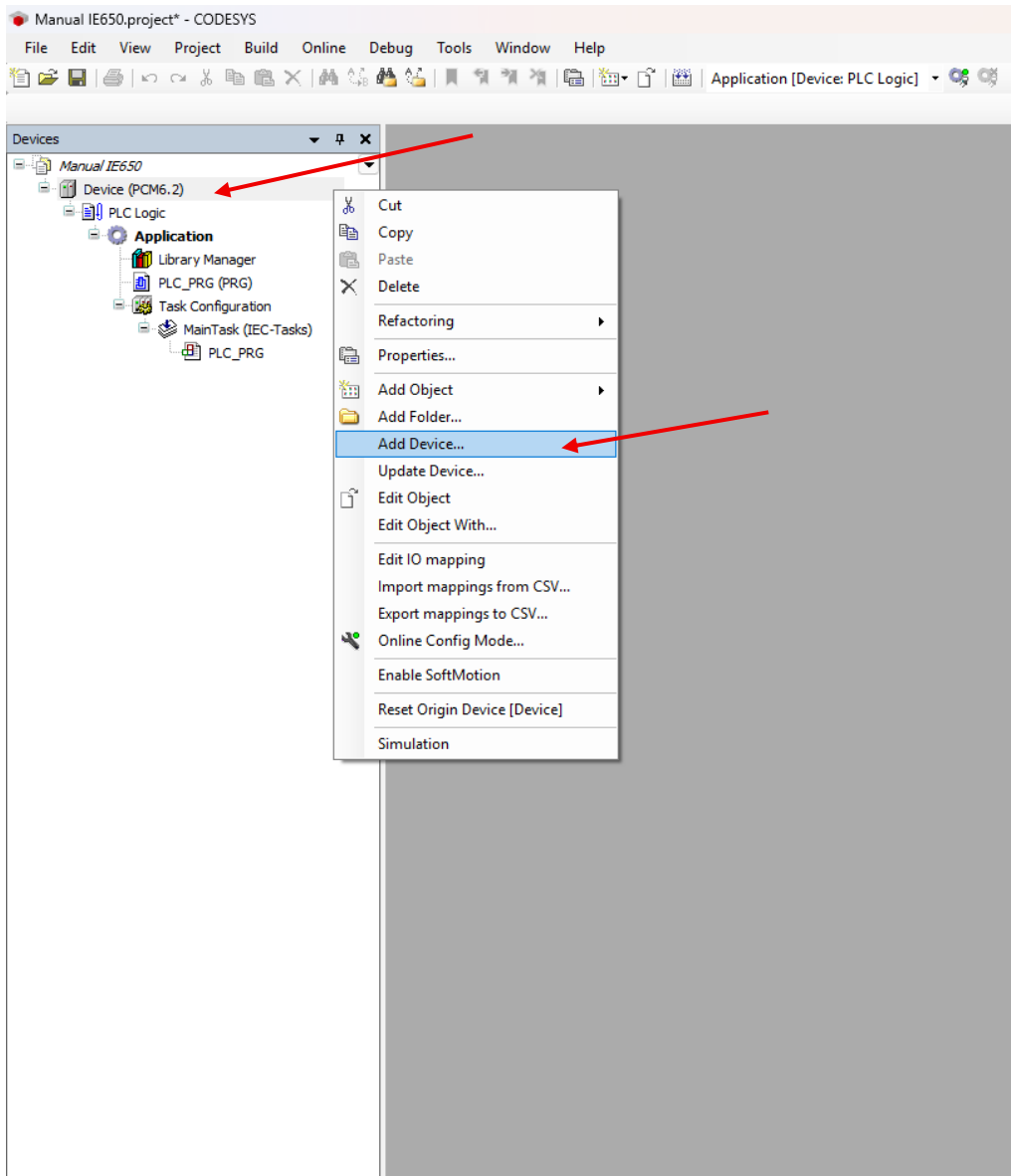


2.5 Add EtherCAT bus

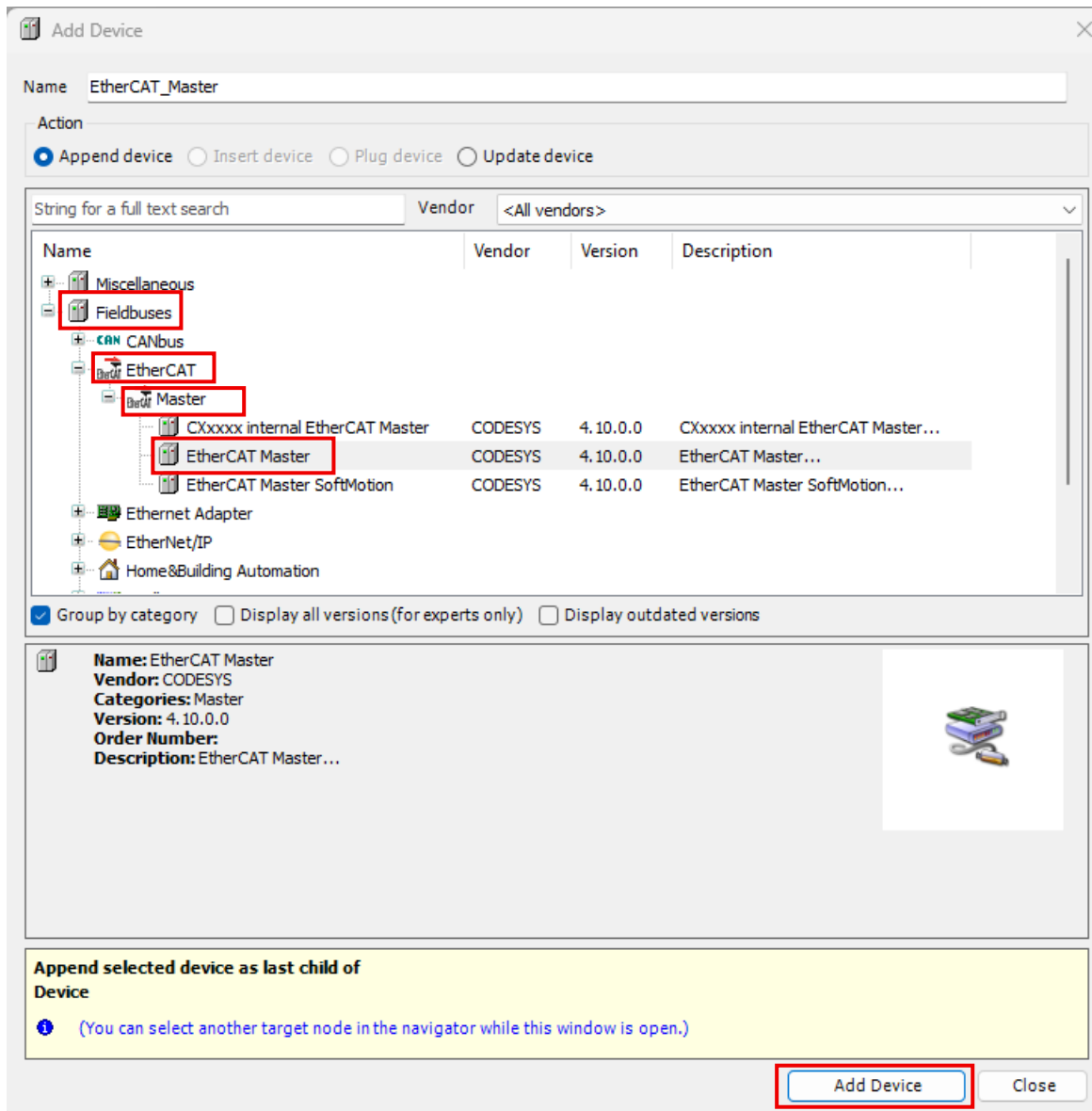
The modules of the iE 650 PLC product communicate with each other using the EtherCAT communication protocol.

Add EtherCAT Master:

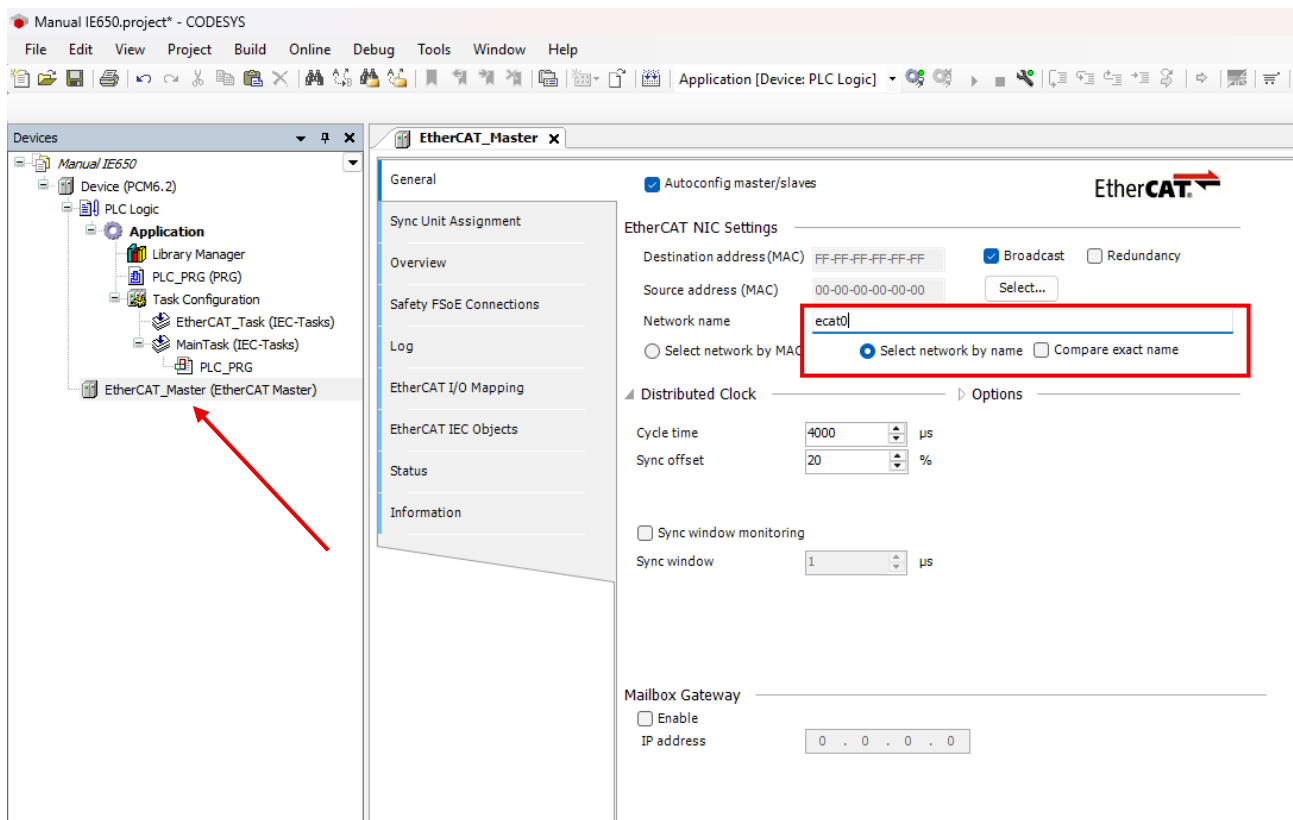
Right-click on **Device** and select **Add Device**.



Select **Fieldbuses / EtherCAT / Master / EtherCAT Master** and click **Add Device**.



Open the **EtherCAT Master** Device and fill in **ecat0** for **Network Name** and choose **Select network by name**

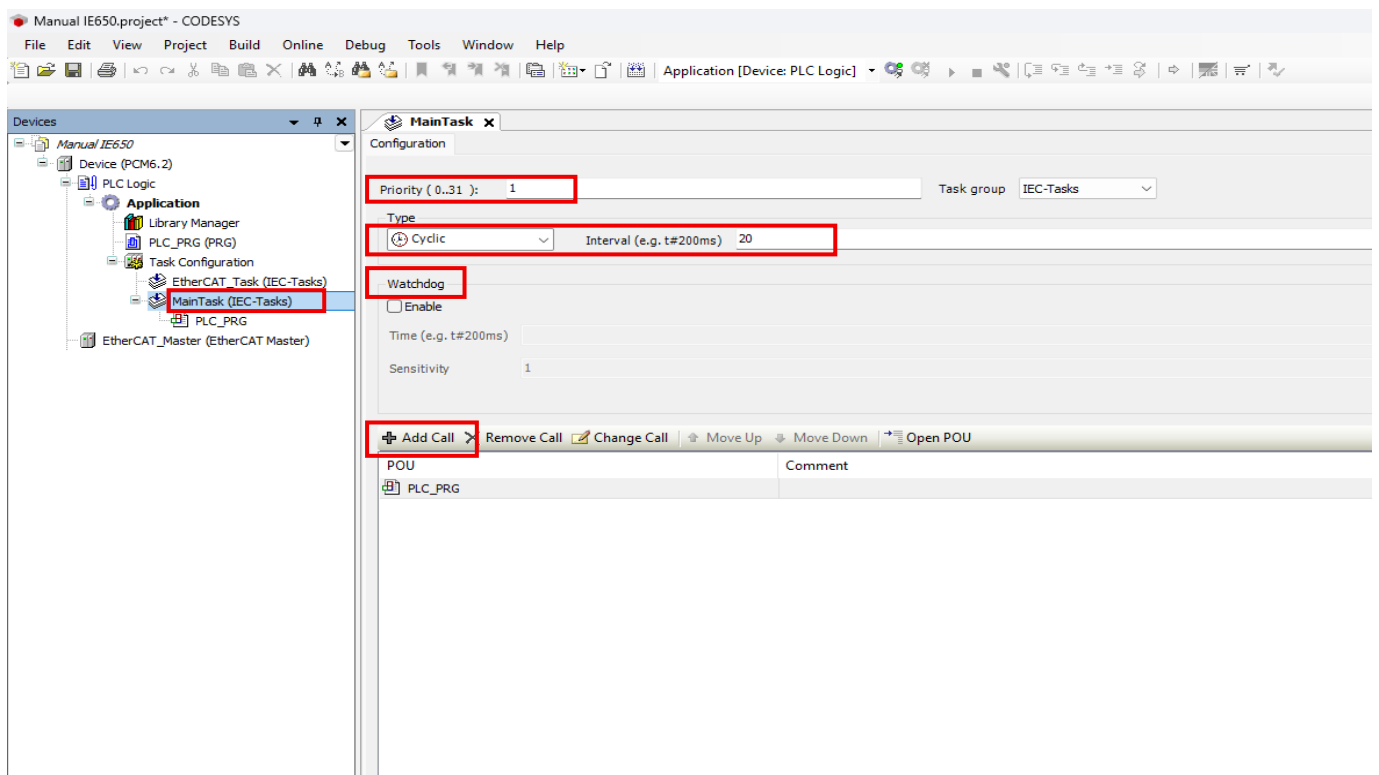


2.6 Task Configuration

Main Task

Configures how and when the task is executed, including timing, priority, execution method, program assignment, and watchdog monitoring.

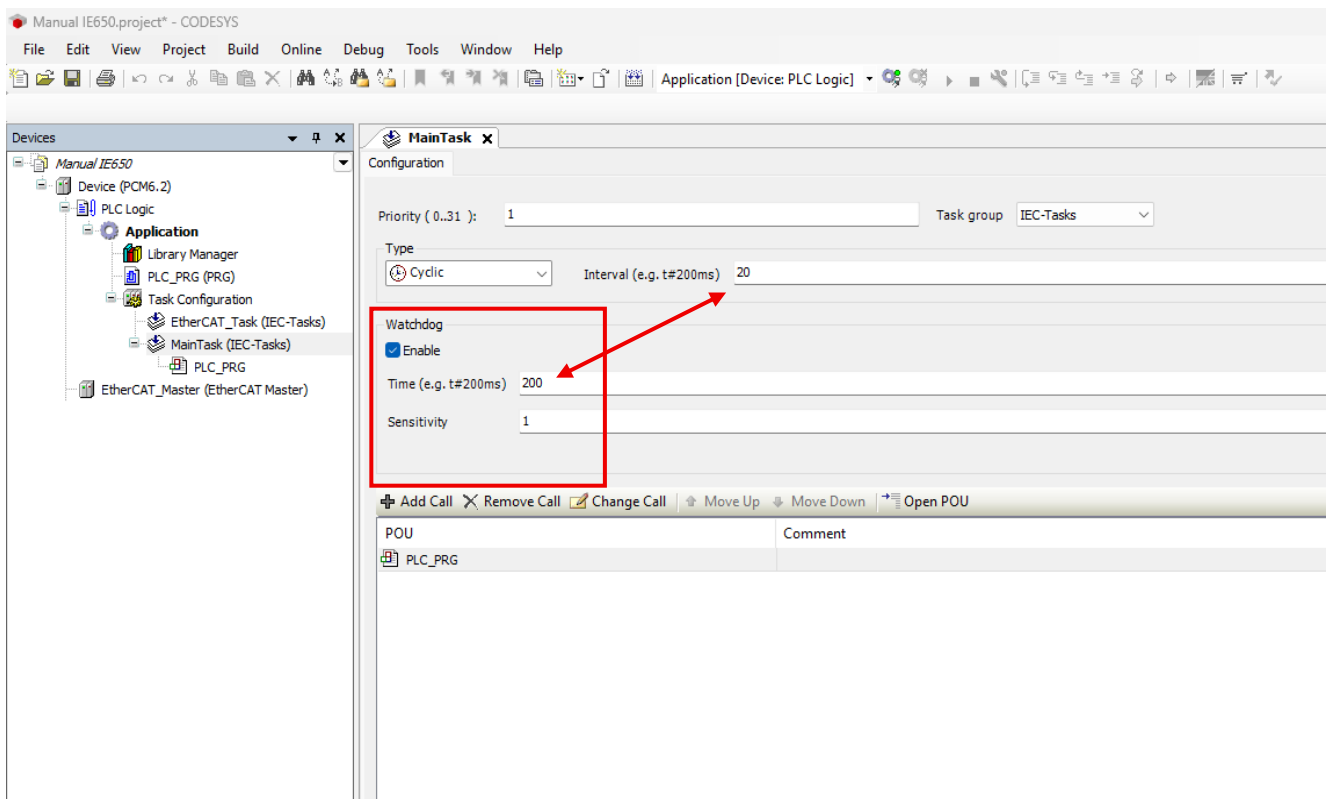
- **Priority:** Sets the task priority from 0 to 31 (0 = highest, 31 = lowest)
- **Interval:** Defines how often the task runs.
- **Type:** Specifies the execution method, typically Cyclic.
- **Watchdog:** Supervises task execution
- **Add Call:** Assigns the program unit that the task will execute.
- **EtherCAT_Task:** Usually left at its default setting.



2.7 Mission watchdog

The Task **Watchdog** function enables detection and control of excessive task runtimes and CPU overload conditions in a CODESYS project.

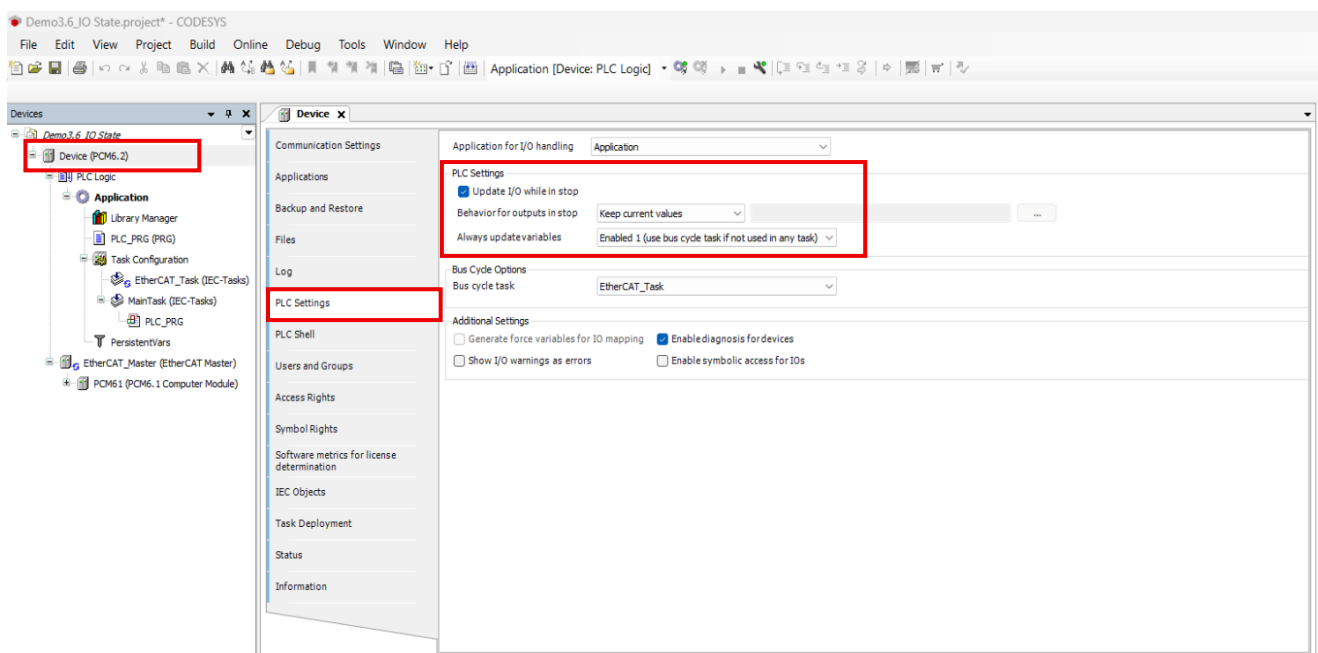
- The watchdog timeout value, **Time**, must be configured higher than the normal task runtime.
- If a task exceeds the configured timeout, the watchdog evaluates whether the runtime exceeds **Time multiplied by Sensitivity**, or whether the task runtime exceeds the timeout repeatedly for more cycles than allowed by the Sensitivity setting. In either case, the watchdog will be triggered
- Watchdog faults are most often caused by programming errors such as infinite loops, null pointer calls, division by zero, or other exceptions. When such a fault occurs, the real-time system halts execution and raises an **Exception Error**, leaving all program variables unchanged from their pre-fault state.
- For the handling of abnormal system faults, you can use the interface callback procedure provided by CODESYS and set in the callback procedure what action to take when an abnormal system fault occurs, such as restarting the controller.
- The watchdog functionality can be enabled, disabled, or re-enabled by using an interface function.



2.8 Setting the I/O Default State

The I/O default status can be modified from the **Device/PLC Settings**

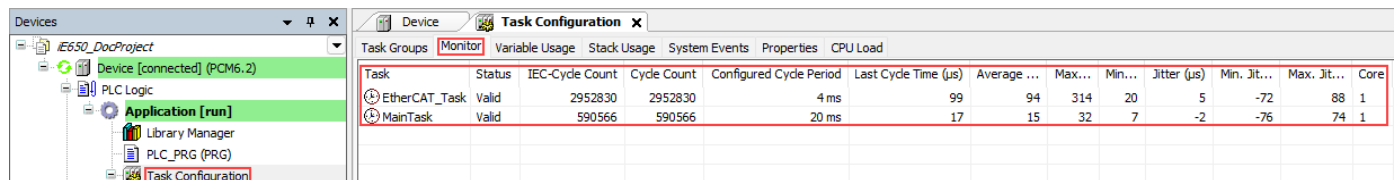
- The **update I/O while in stop** needs to be enabled so that the PLC in Stop mode communicates with field devices, reads inputs, and potentially writes outputs.
- **Behaviour for Outputs in stop** determines the output behaviour when the PLC turns to stop mode. The selections are:
 - Keep current values
 - Set all outputs to default
 - Execute program
- **Always update variables** determines whether variables are refreshed every cycle, even if they are not explicitly used in the program logic. The selections are:
 - **Deactivated - update only if used in a task.**
Variables are only updated if the compiler detects they are used in the application code.
 - **Enabled 1 - use bus cycle task if not used in any task.**
Variables not used in any task are updated automatically using the Bus Cycle Task.



2.9 Controller Load Monitoring

In CODESYS, you can view task run cycle times and statistics as well as the execution time of each subsystem, through **Task Configuration > Monitor**.

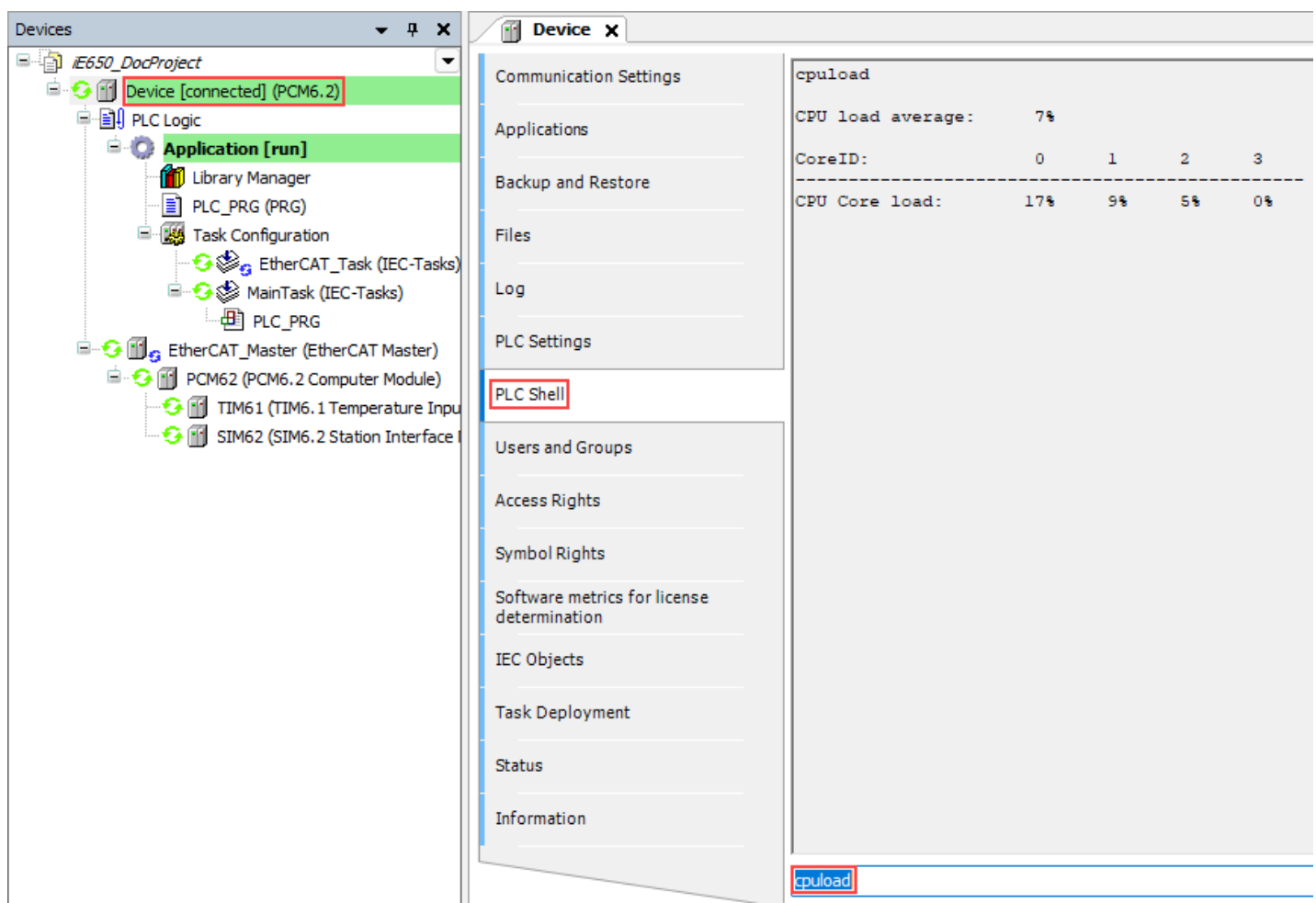
- The task runtime must be less than the task setup period and as short as possible to avoid system failures due to task timeouts.



Task	Status	IEC-Cycle Count	Cycle Count	Configured Cycle Period	Last Cycle Time (µs)	Average ...	Max...	Min...	Jitter (µs)	Min. Jit...	Max. Jit...	Core
EtherCAT_Task	Valid	2952830	2952830	4 ms	99	94	314	20	5	-72	88	1
MainTask	Valid	590566	590566	20 ms	17	15	32	7	-2	-76	74	1

Controller load conditions can also be viewed through the **PLC Shell** web control interface.

When logged in (Online) to a running application, enter the command **cpuload** in the **Device > PLC Shell** page to view the controller load.



The screenshot shows the CODESYS interface with the 'Device' window open. The 'PLC Shell' tab is selected, and the command 'cpuload' has been entered in the input field. The output of the command is displayed in the main area of the window.

```

cpuload

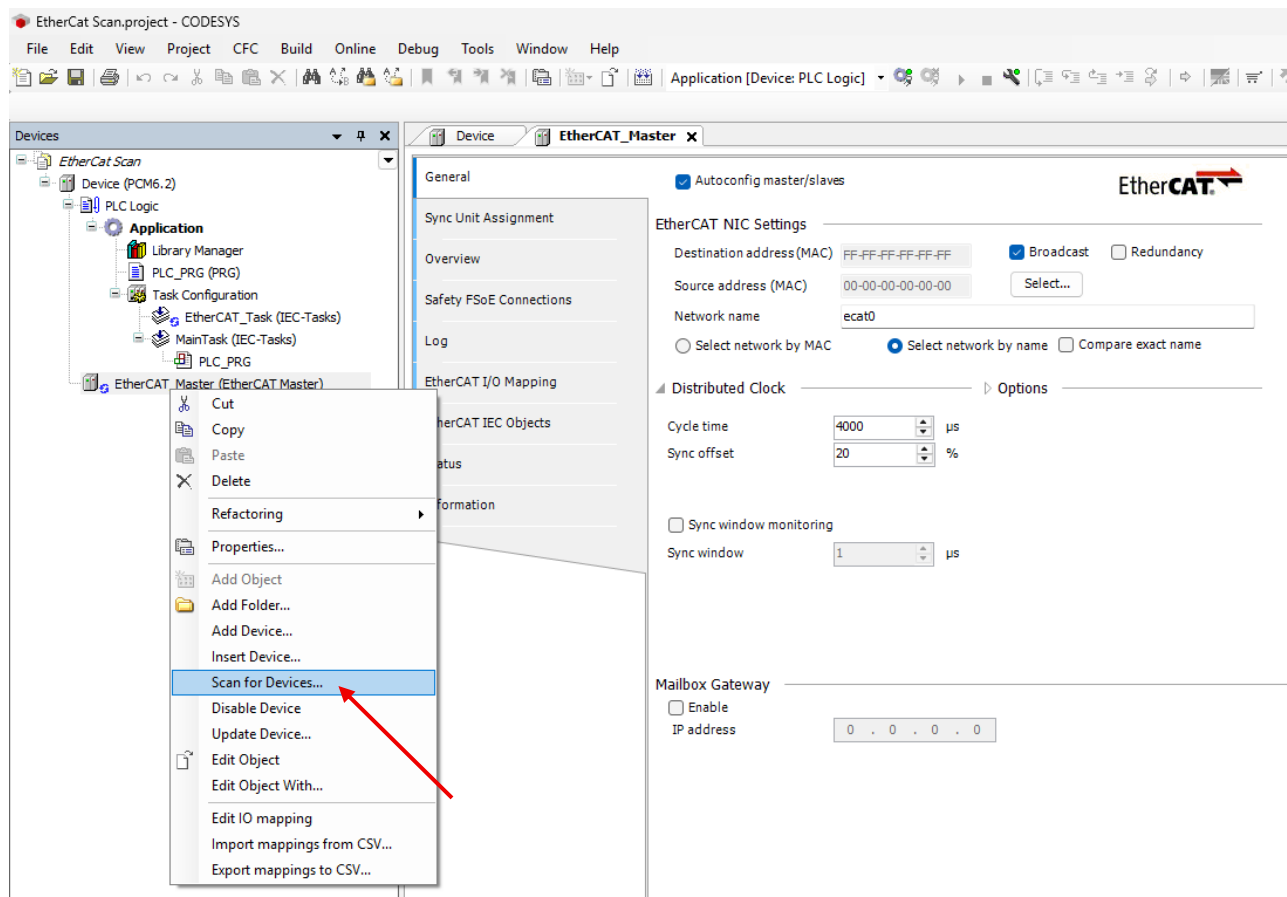
CPU load average:      7%

CoreID:                0      1      2      3
-----
CPU Core load:         17%    9%    5%    0%
  
```

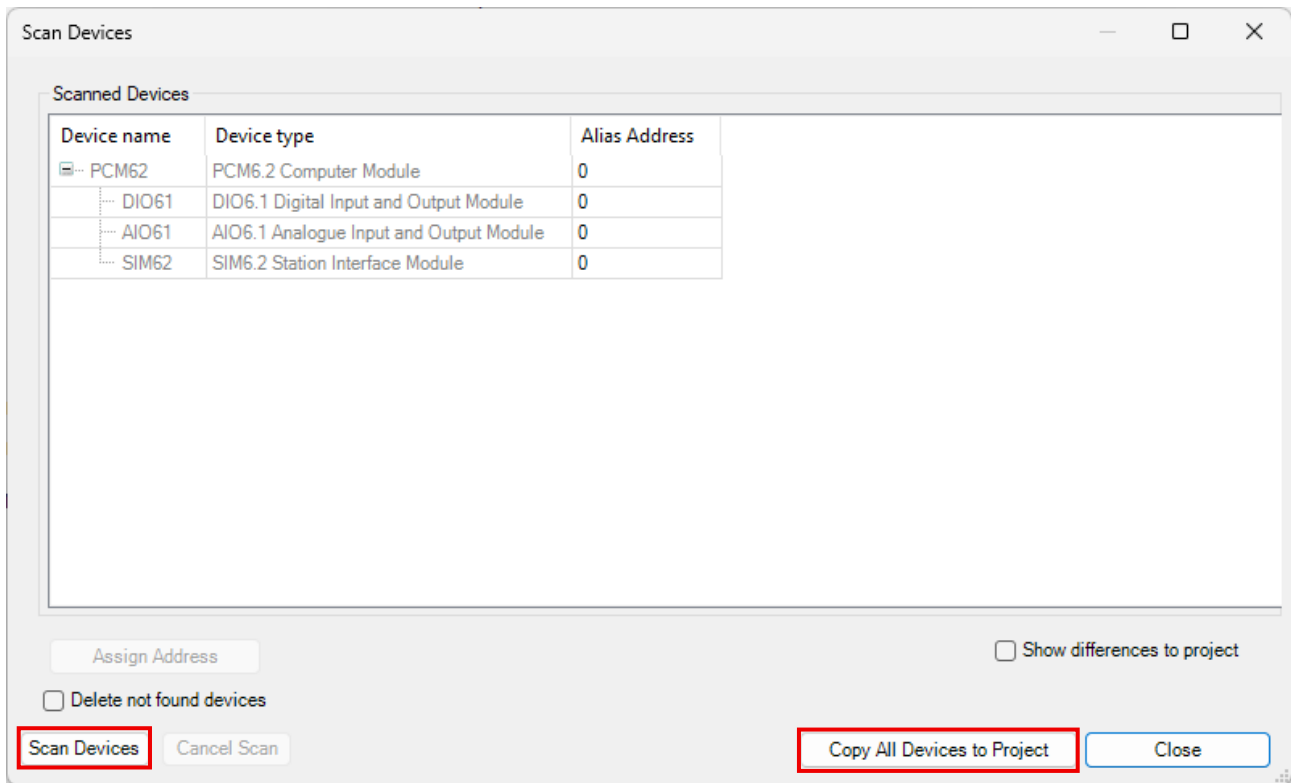
2.10 Automatic scanning equipment

CODESYS supports the iE 650 PLC Auto Scan Add Device feature.

- Once the EtherCAT bus has been added and the application has been logged in and downloaded to the device, right-click **EtherCAT_Master** and select **Scan for Devices**.



- Click on **Copy All Devices to Project**.
- **Logout** of the device.
- **Log back in** the device and **start the program**.



Note: Not all hardware devices can be scanned and added to the project automatically. It is recommended to check that the scanned devices are meeting the configuration requirements. If devices are missing it is possible to add them manually.

To add manually

- Right-click on **EtherCAT_Master** and select **Add Device** and select e.g. **PCM6.2**.
- Right-click the **PCM6.2** and select **Add Device**
- Now add the other cards like **DIO6.1**, **AIO6.1**, and **SIM6.2**.

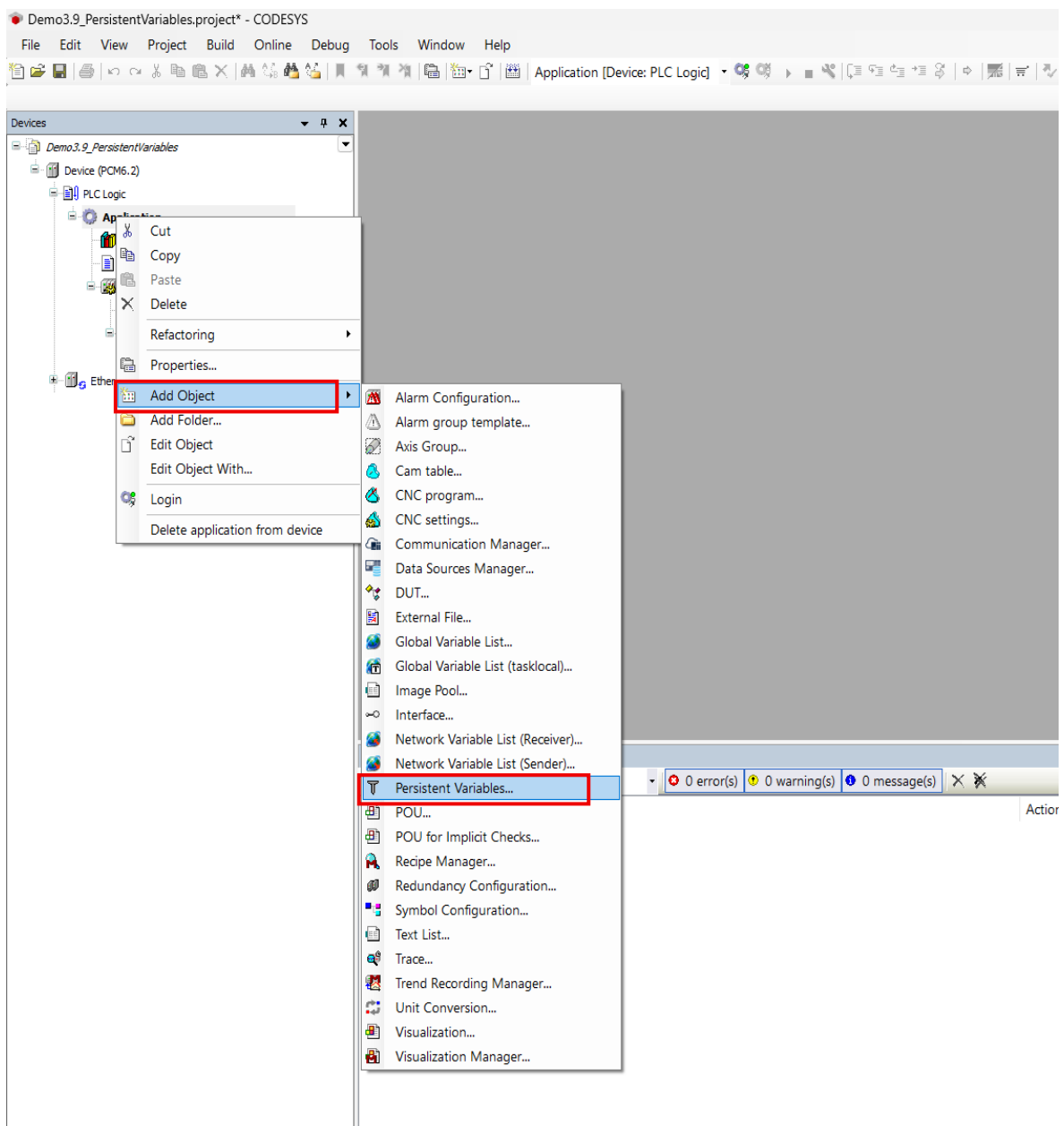
2.11 Persistent variable

Persistent variables in CODESYS are used to retain data even after events like power loss, PLC restart, or download, depending on configuration. They are essential for storing critical values that should not reset during normal operation interruptions.

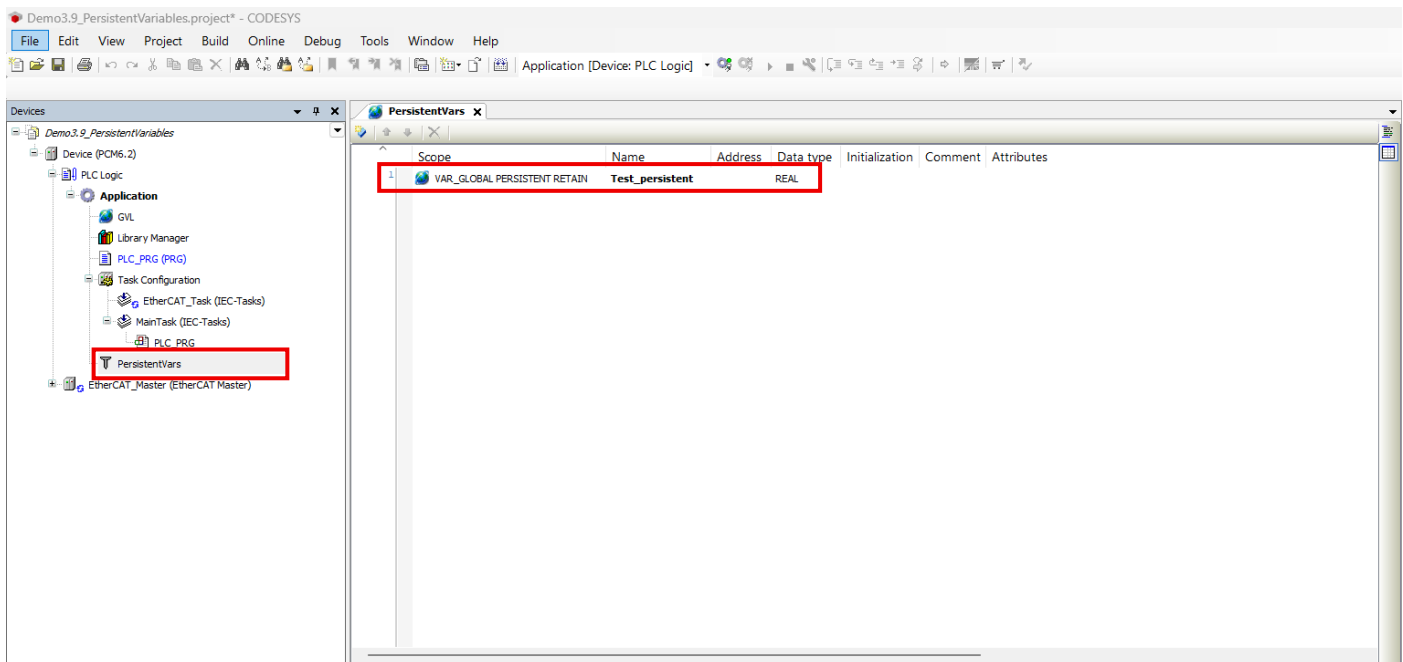
While these variables are designed to retain their values reliably, it is still advisable to maintain backups of critical data to safeguard against potential loss caused by hardware damage, incorrect updates during engineering, or other unforeseen issues.

The persistent variables can be added by the following steps.

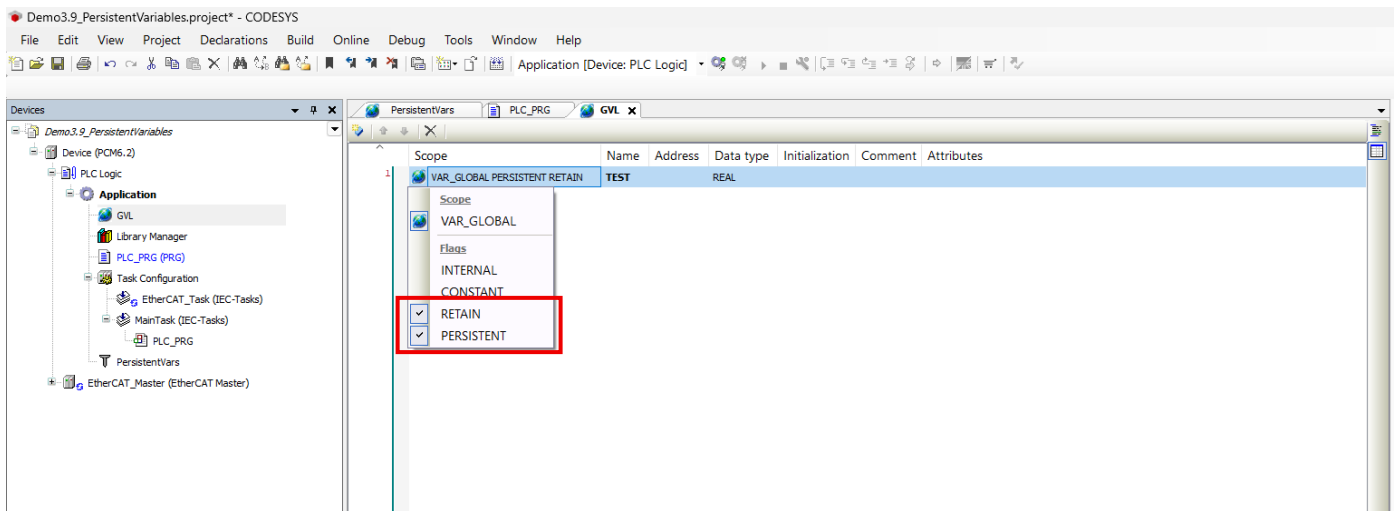
- Right-click on **Application** then select **Add Object** and then **Persistent Variables** in the pop-up dialogue box as shown in the below picture.



- The tags can be added by clicking **Insert**.



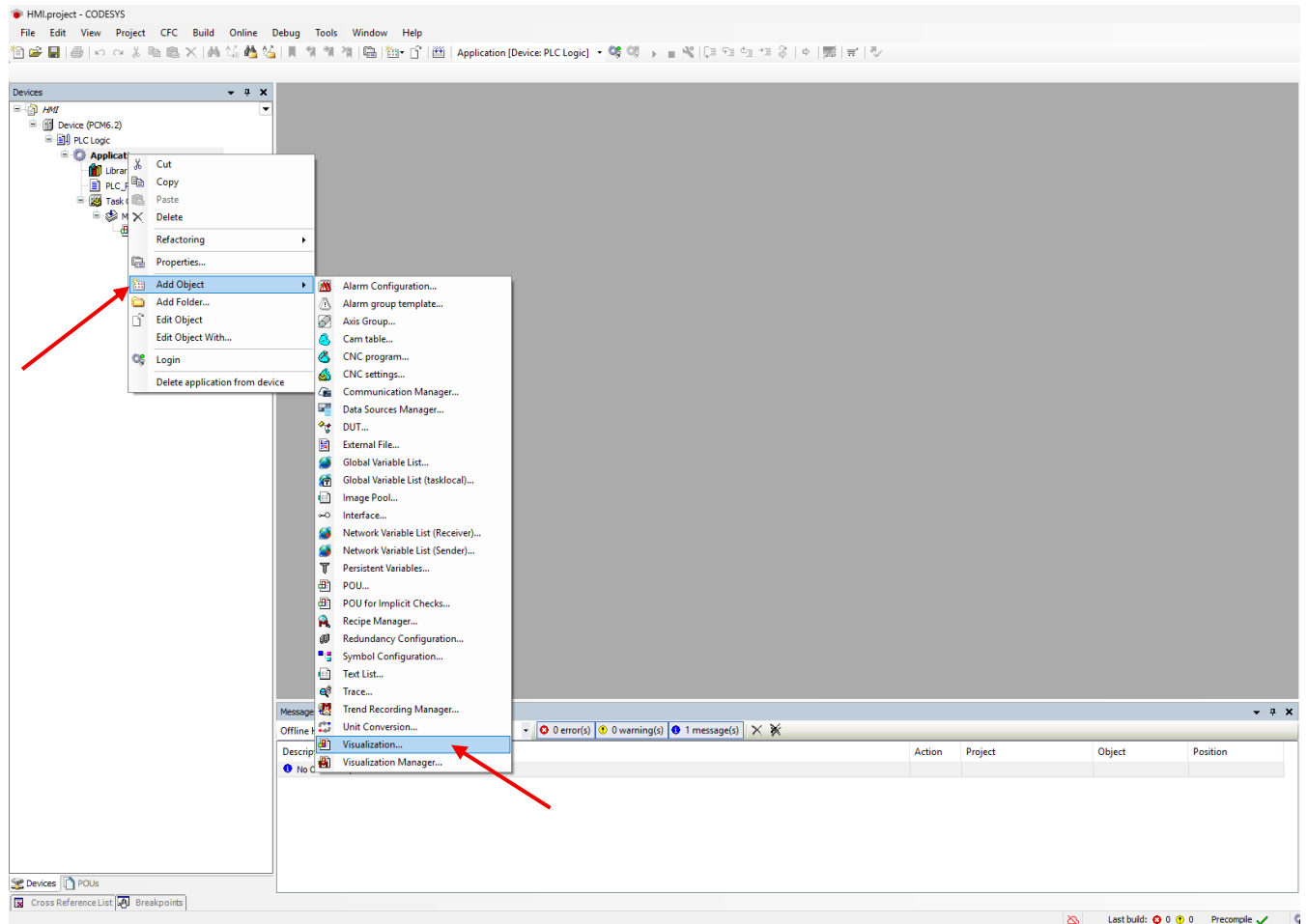
Note: A tag defined in the GVL can be converted to persistent tag by selecting the scope of the tag to **PERSISTENT** and **RETAIN** as illustrated in the image below.



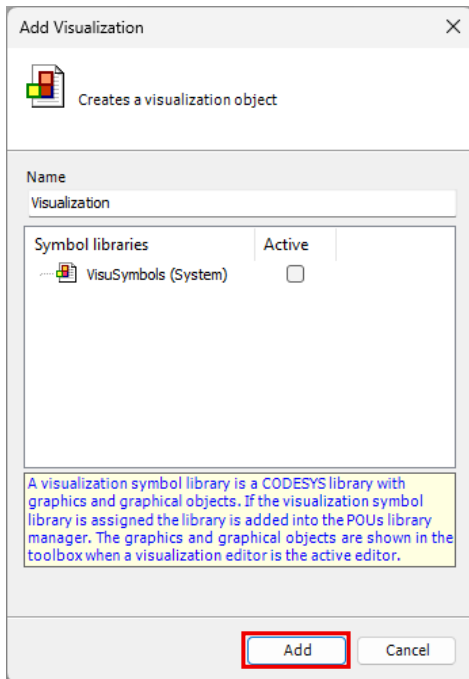
2.12 Creating an HMI

CODESYS supports the creation and editing of HMI interfaces which can be accessed through a browser.

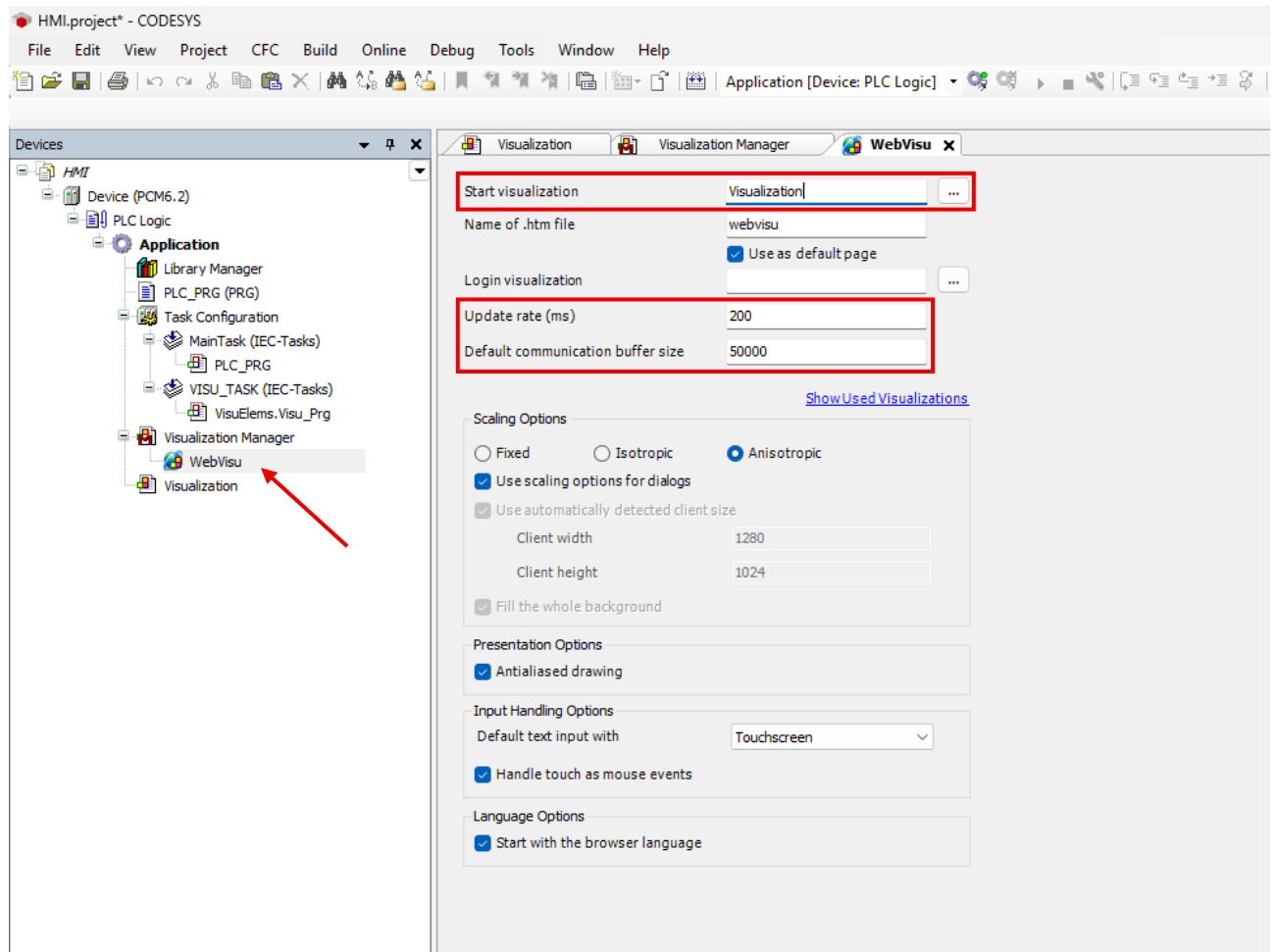
- Create a new project and select **Standard project**.
- Right-click on **Application** and then **Add Object / Visualization**.



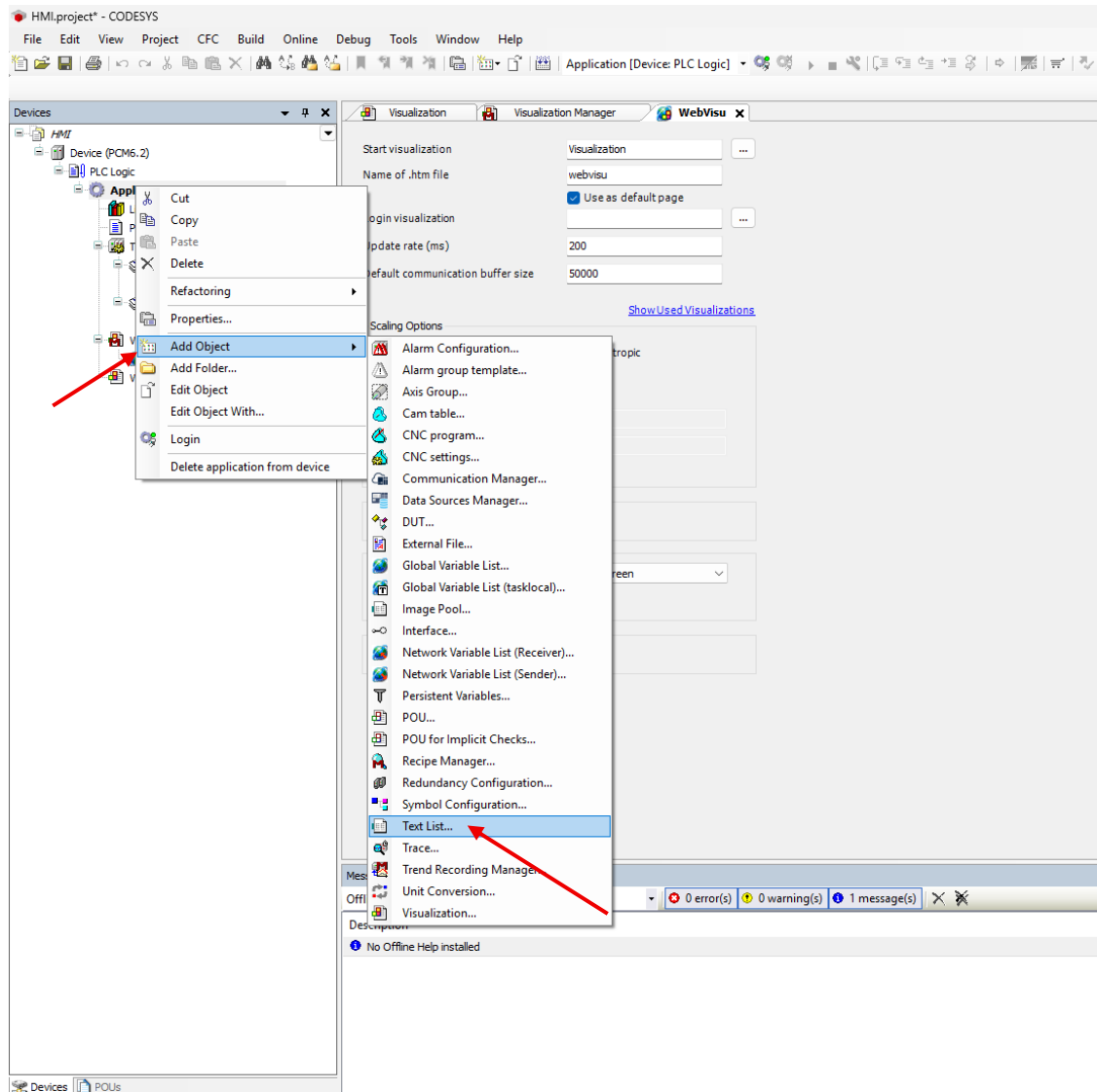
- Click on **Add**.



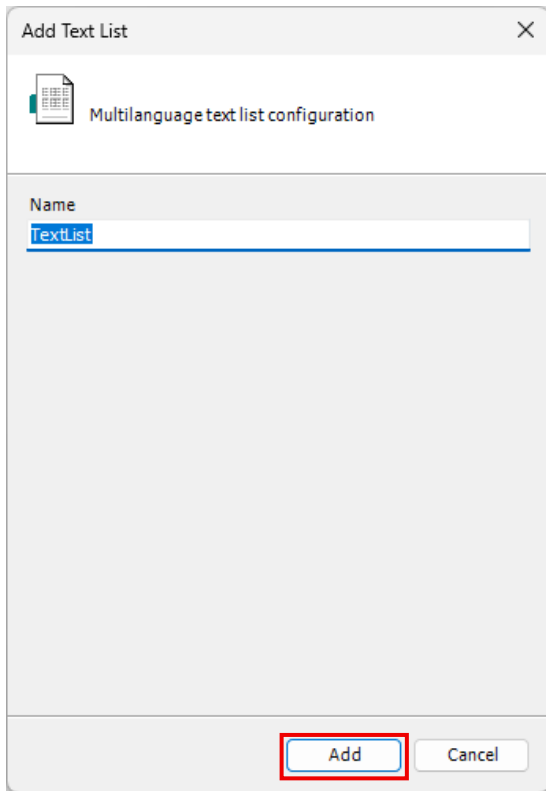
- Double-click **WebVisu** under **Visualization Manager**.
- Here you can set the first interface displayed when a user logs in to the HMI by modifying **Start visualization**.
- It's also possible to change the interface refresh rate and buffer size - the default settings are usually applied.



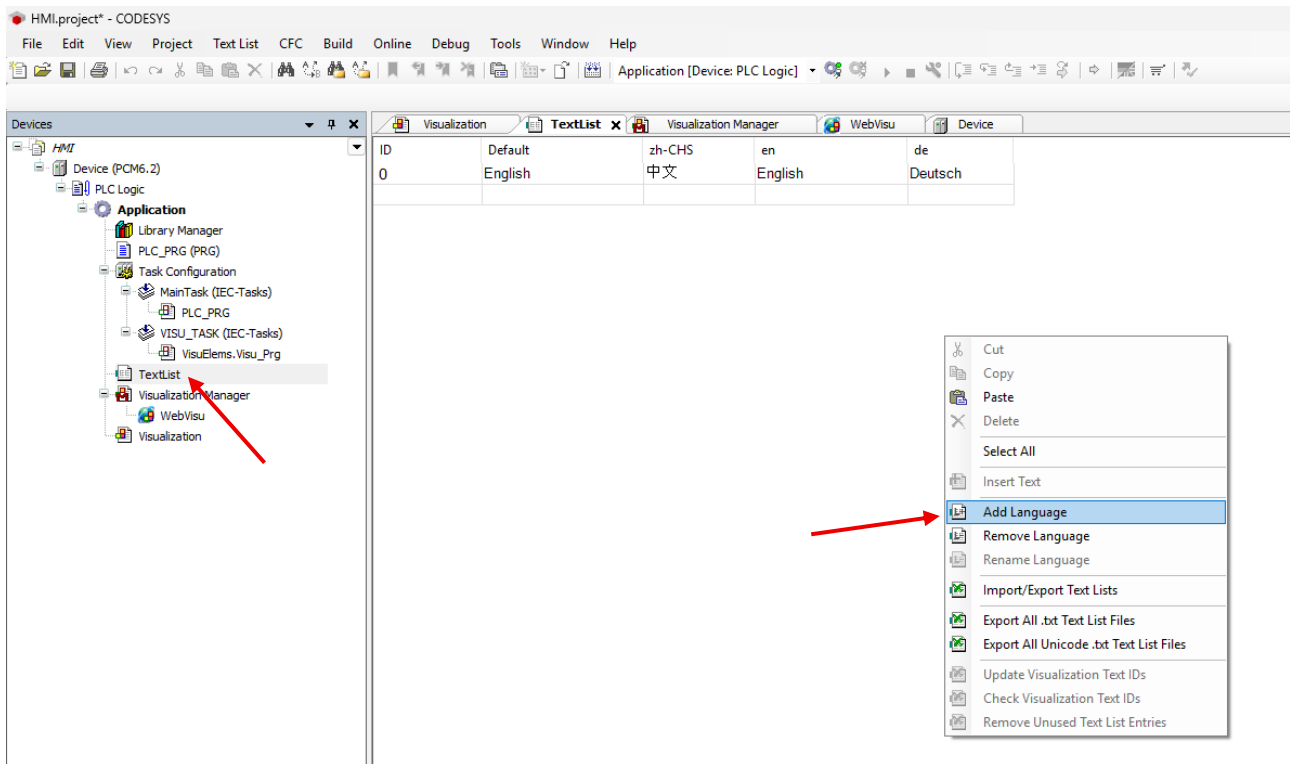
- To be able to switch languages of the HMI a **TextList** is needed.
- Right-click on **Application** and then **Add Object / TextList**.



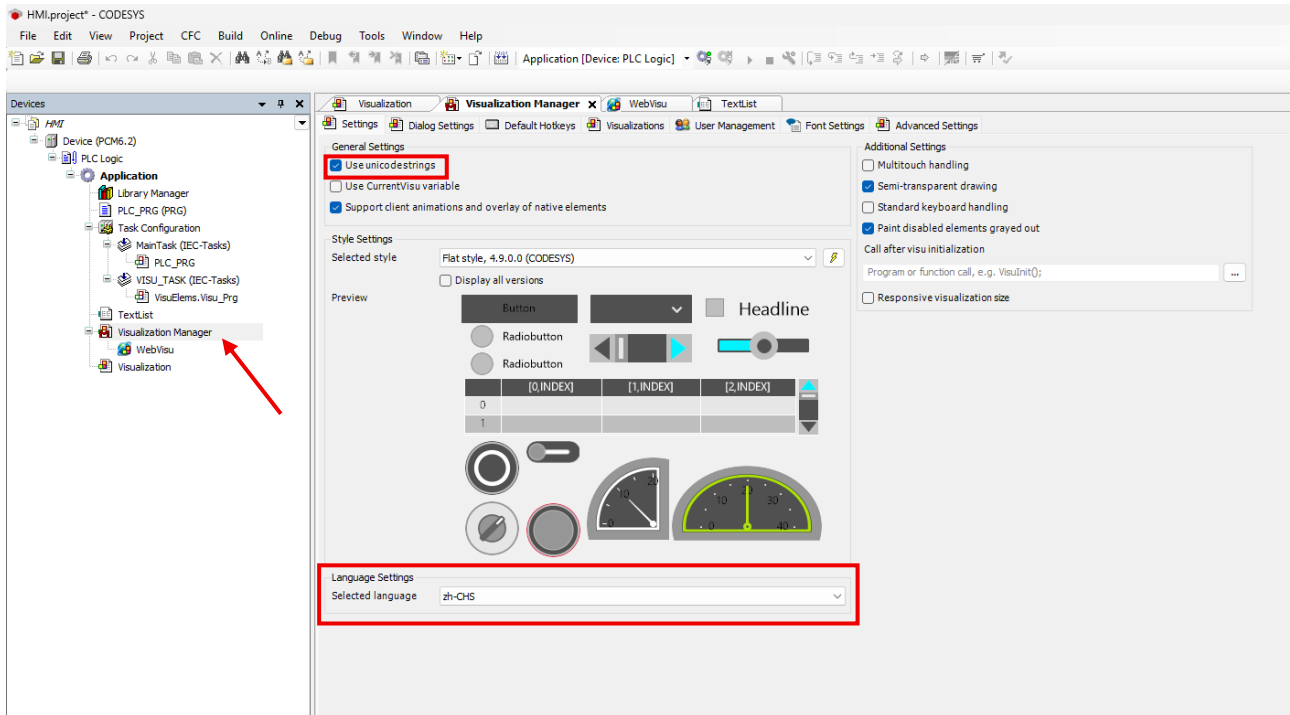
- You can change the name and then click **Add**.



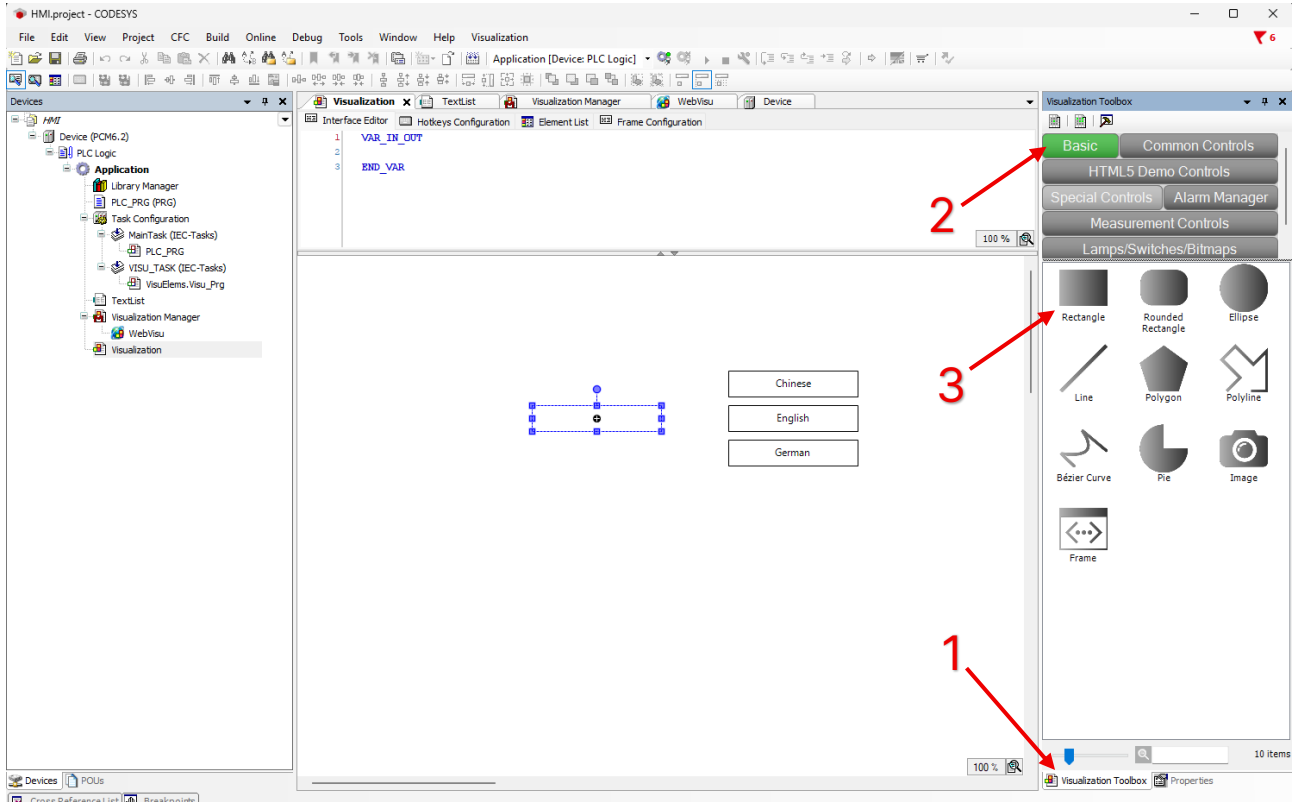
- If the HMI requires multiple languages new columns can be added in the **TextList**.
- Right-click in the white space and select **Add Language**.



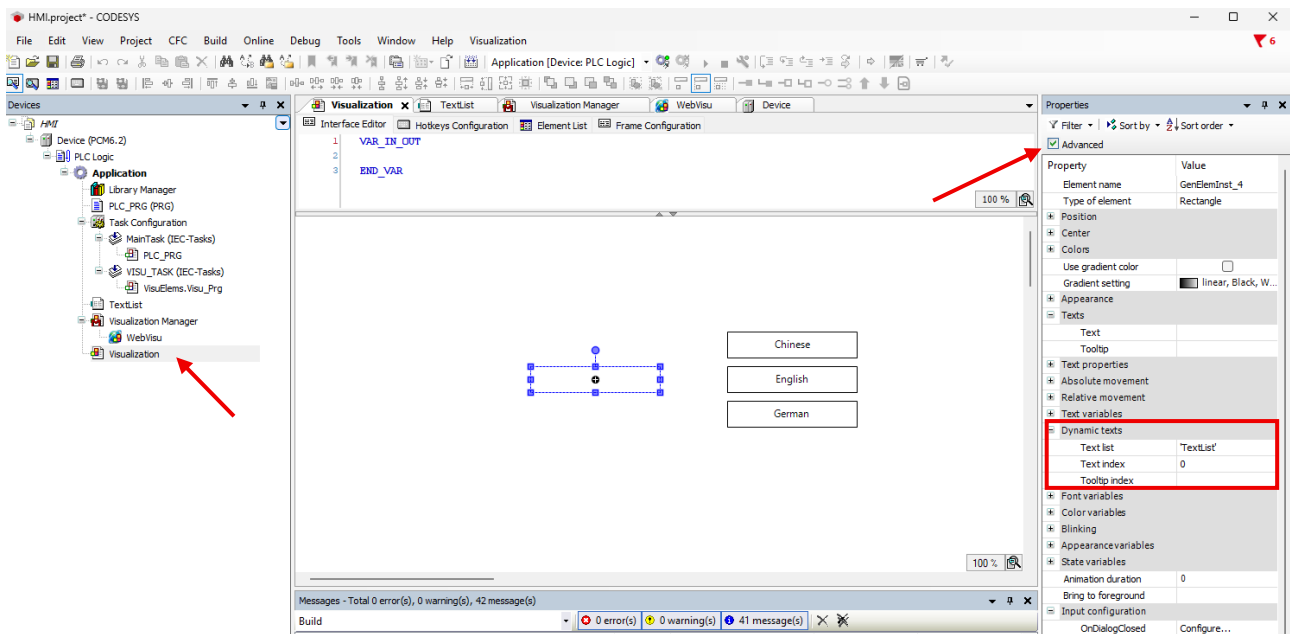
- The Visualization Manager can only display Chinese in the interface by checking **Use unicodestrings** then it's possible to select **zh-CHS** in the **Language Settings**.



- Double-click the **Visualization**. Here you can build the HMI.
- Add elements by selecting the **Visualization Toolbox** select an element there and drag it into the HMI screen. Or select an element and press into the HMI screen.



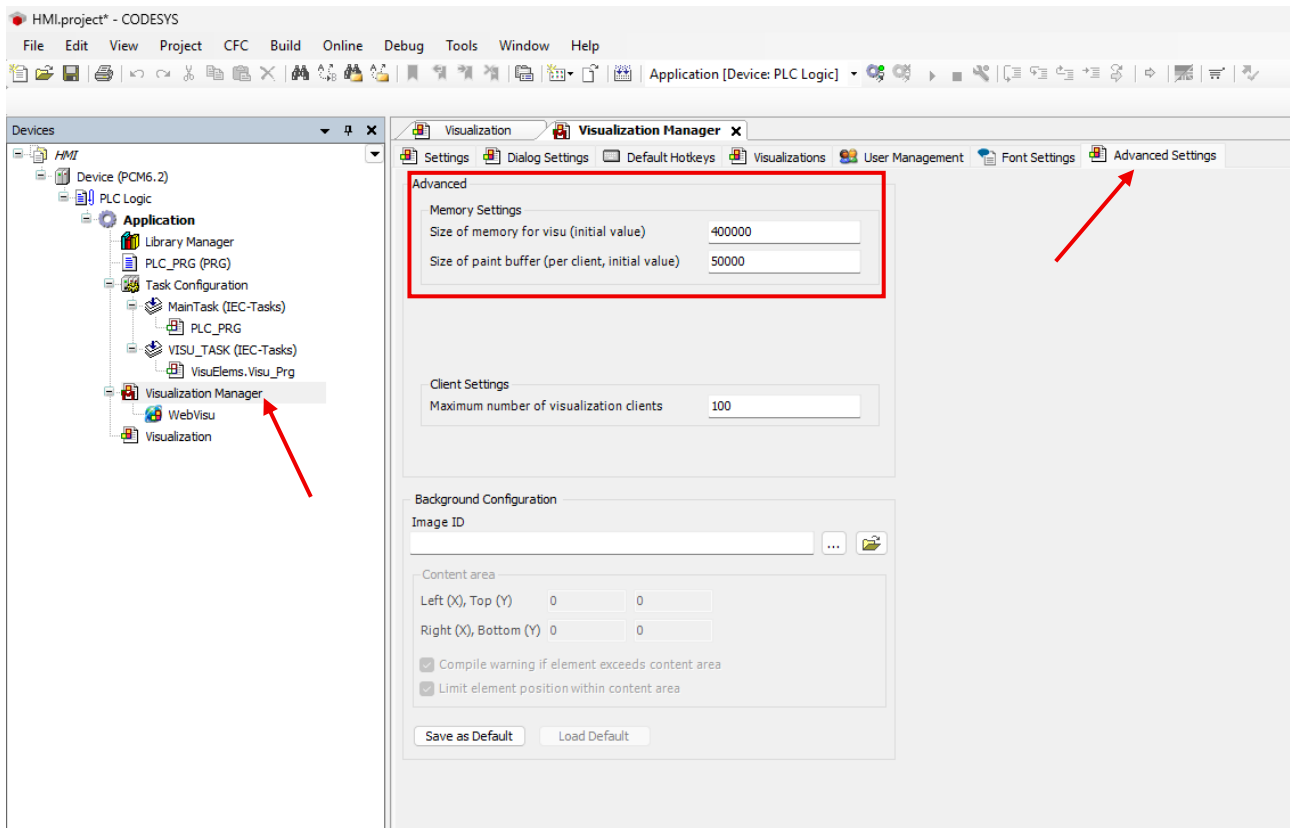
- It's a good idea to enable the **Advanced** settings to show all properties of HMI elements.
- In the picture are four rectangles the left one can display the text of the **TextList**.



- To access the HMI in the browser, enter **https://IP:8443/webvisu.htm** e.g. **https://192.168.20.13:8443/webvisu.htm**

Note: If you add too many elements on a single page, memory problems can appear which show as a white screen or lag. To fix this:

- Double-click the **Visualization Manager**.
- Under **Advanced** it's possible to increase the memory.



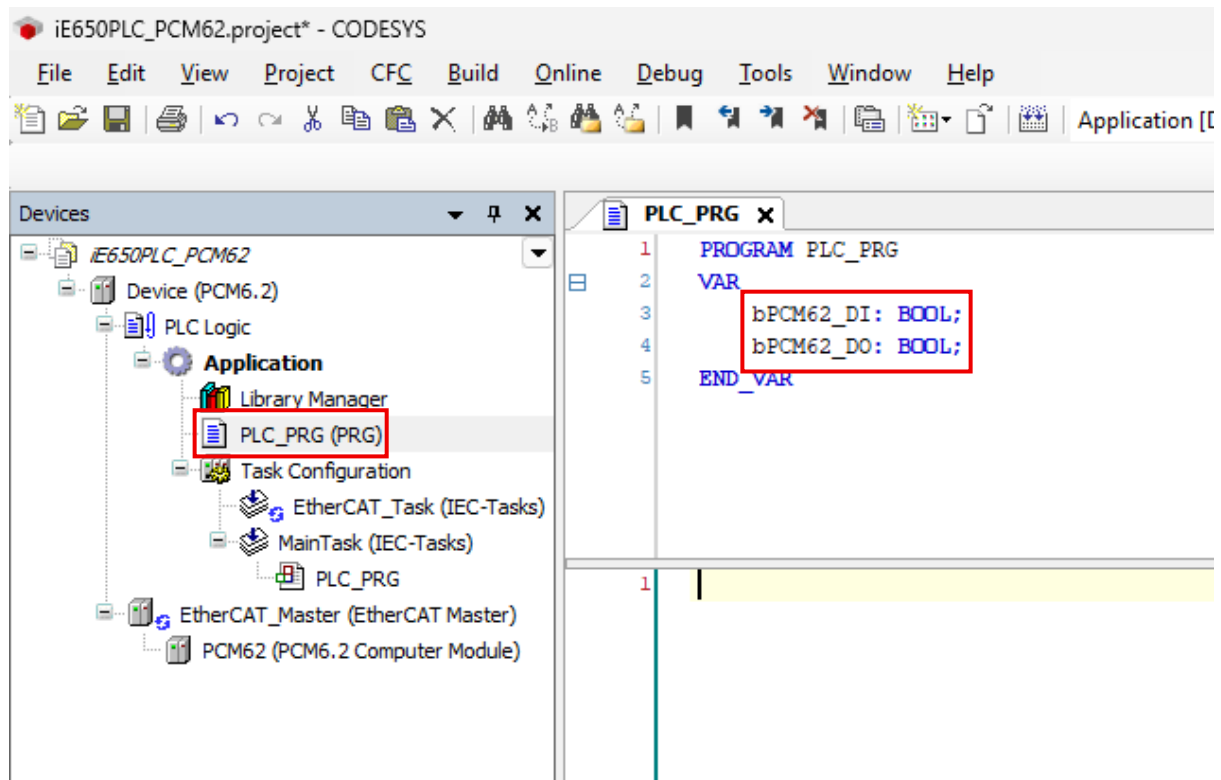
3. iE 650 PLC program configuration

3.1 PCM6.2 program configuration

3.1.1 Digital program configuration

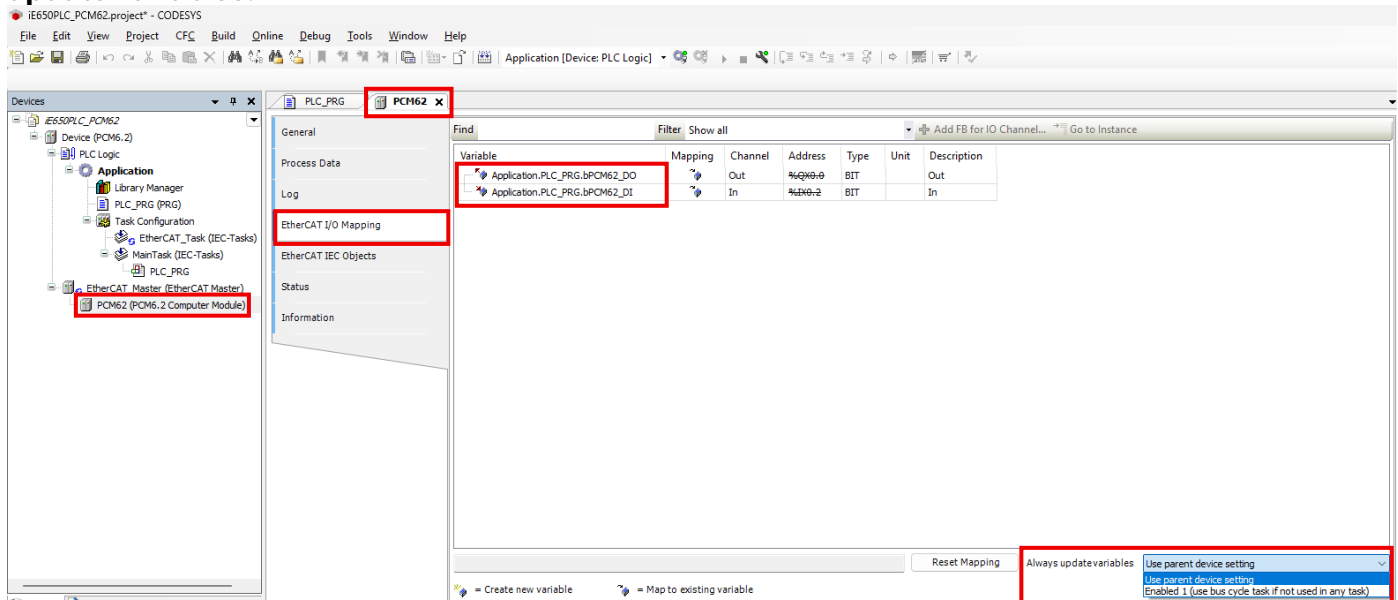
The PCM6.2 contains one DI and one DO, that can be used for any digital signal but are in many applications used for status/watchdog.

Below is a simple configuration of two Boolean variables in a PRG.



Above variables are assigned in the **EtherCAT I/O Mapping**.

In this example the channel of these assignments is not called in a task, therefore it is needed to select **Enabled 1 (use bus cycle task if not used in any task)** under **Always update variables**.



3.1.2 Serial Programming

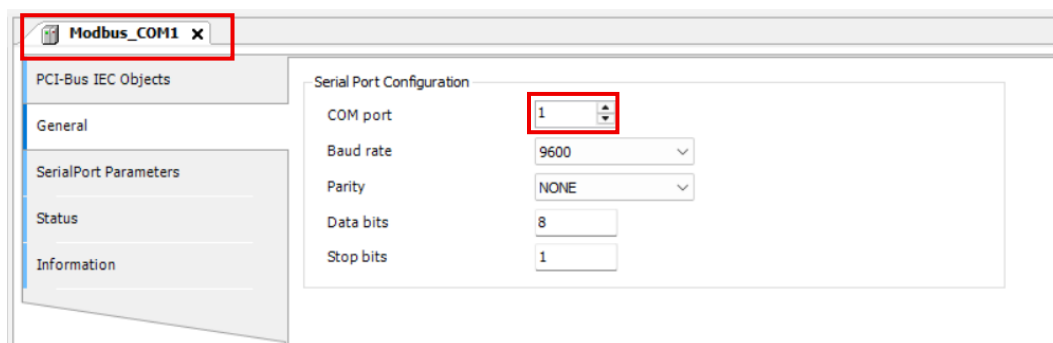
PCM 6.2 module has 2 RS-422/485 channels and supports Modbus-RTU communication.

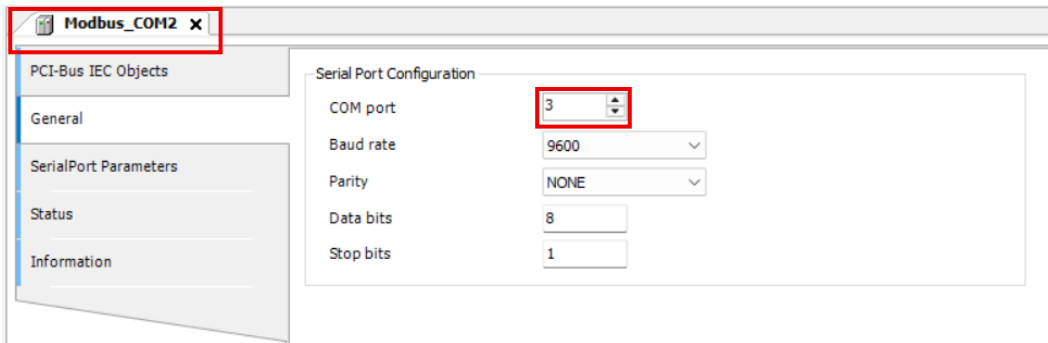
This example uses the Modbus-RTU communication.

Please refer to the Demo Project – **Demo 3.1.2_PCM6.2_ModbusRTU.project**.

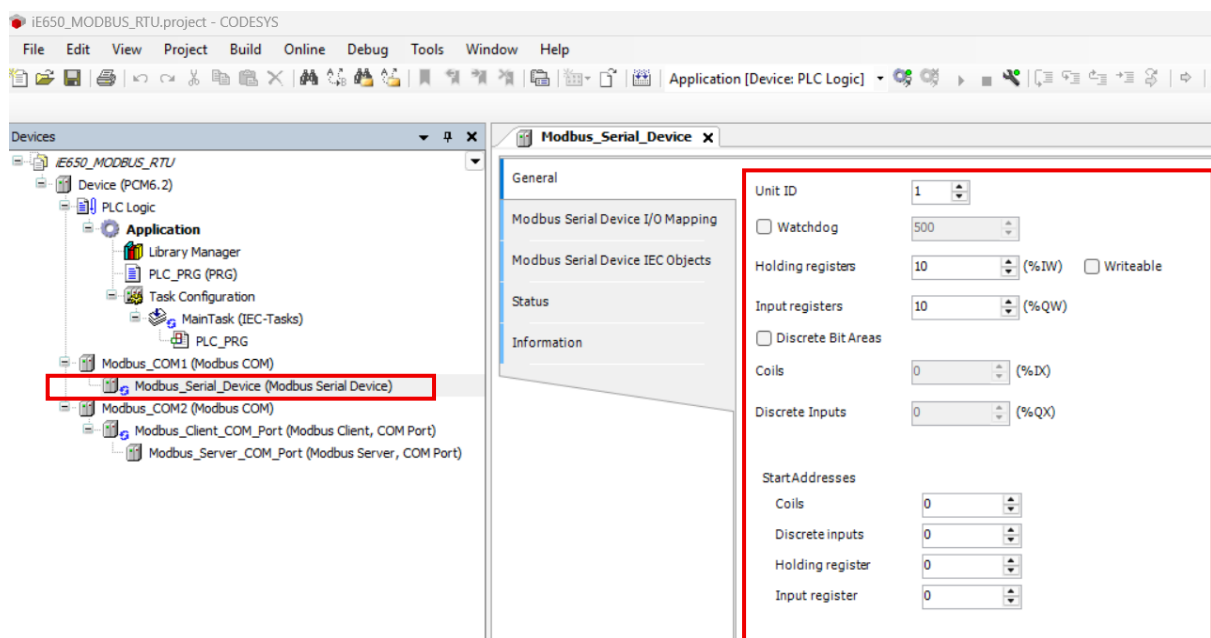
In this example, COM port 1 acts as a Modbus Server device and the COM port 2 as Modbus Client device and both the ports are connected physically.

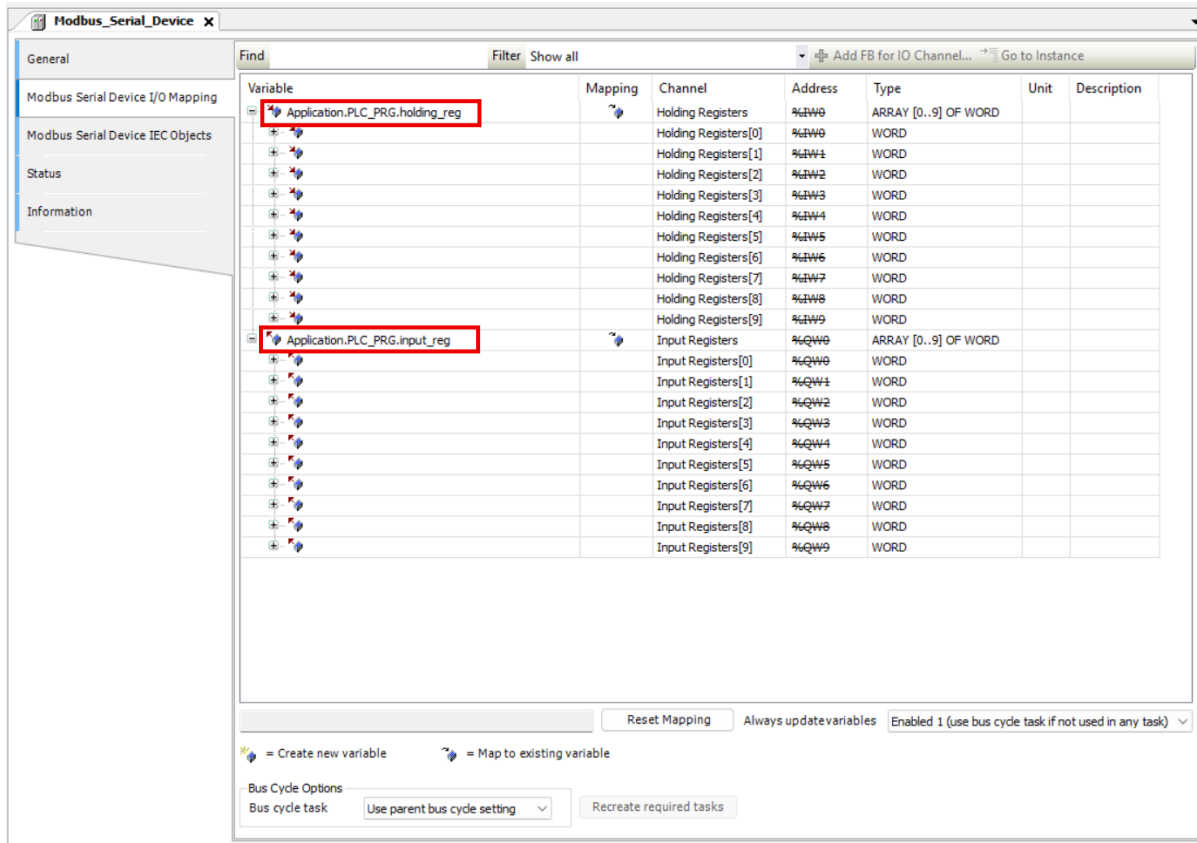
The Serial port configuration parameters are configured as below for the respective ports



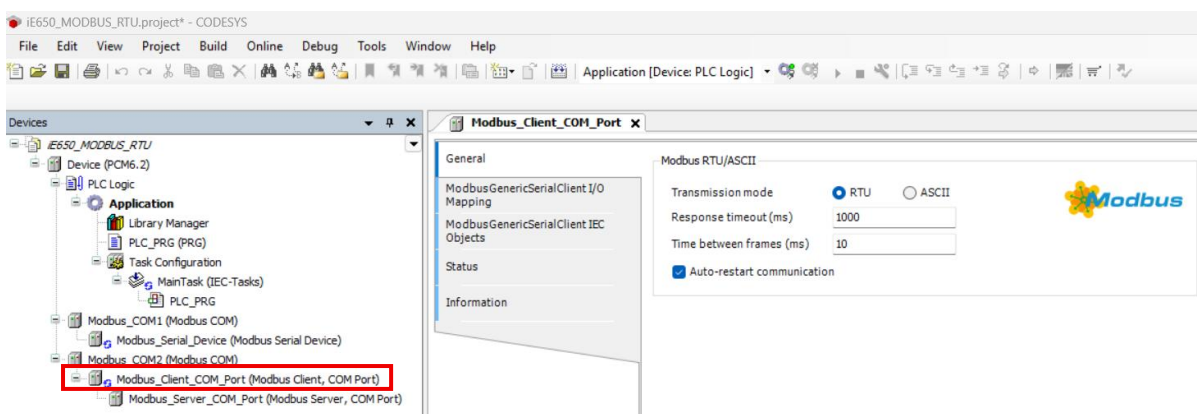


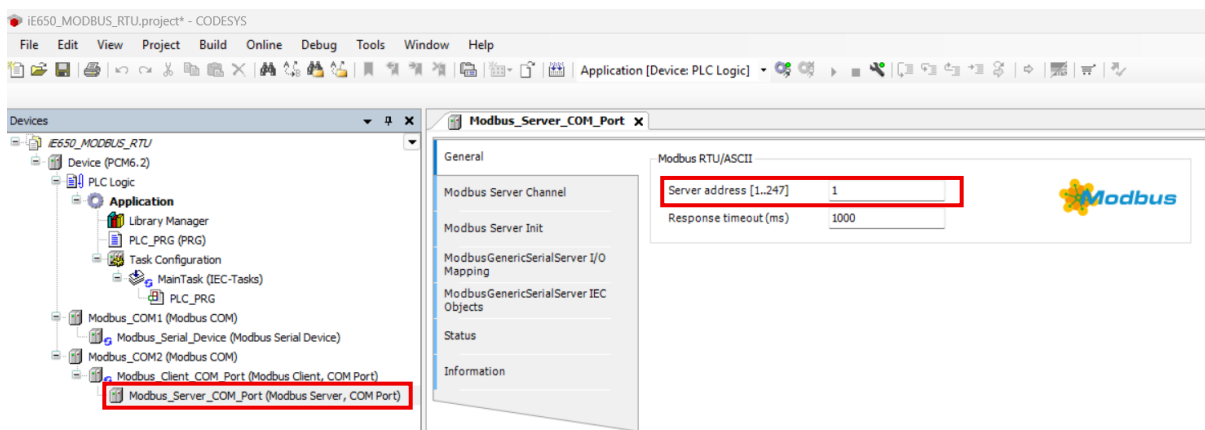
The **Modbus Server device** is added to COM port1 and the configured as shown below. The variables mapped to the IO mapping, holds the data of the respective **Holding registers** and **Input registers**.



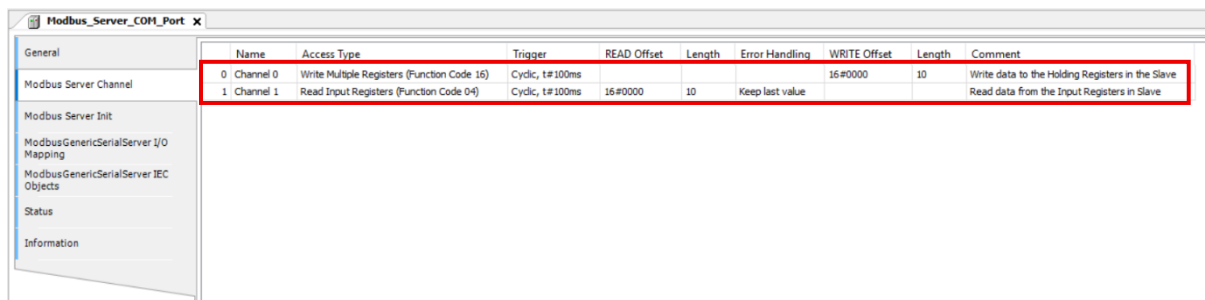


On COM port2, **Modbus_Client_COM_Port** is added, which works as a Modbus Client on COM port 2, the device **Modbus_Server_COM_Port** is not a device itself, but the settings COM2 needs to successfully communicate with the physical slave (in this case COM1). Please find the respective settings of these two devices as shown below.

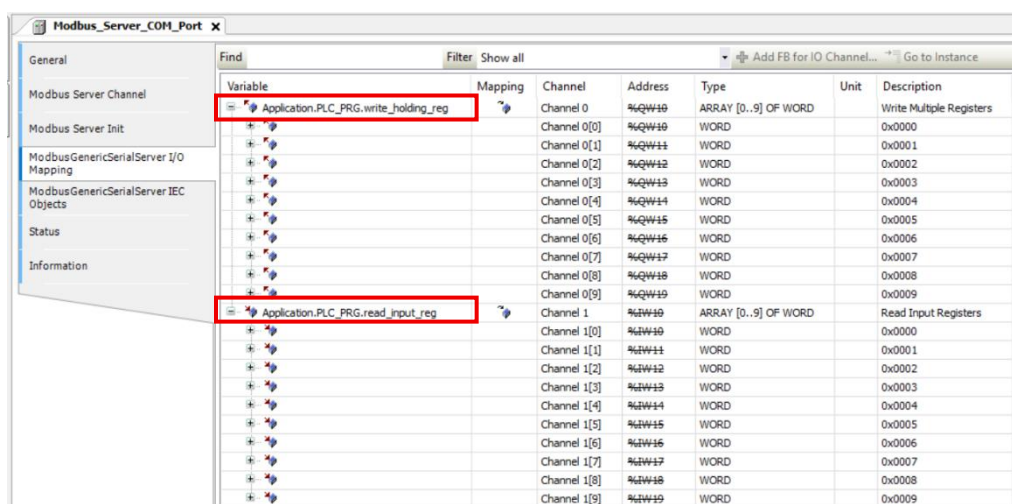




The below configuration allows the Modbus Client in COM 2 to Read/Write data from Holding and Input registers.



These values are mapped to the following variables as shown below.



All the variable mappings are called in **PLC_PRG**, to monitor them in the watch window.

```
PLC_PRG x
1 PROGRAM PLC_PRG
2 VAR
3   holding_reg      : ARRAY [0..9] OF WORD;
4   input_reg        : ARRAY [0..9] OF WORD;
5   write_holding_reg : ARRAY [0..9] OF WORD;
6   read_input_reg   : ARRAY [0..9] OF WORD;
7 END_VAR

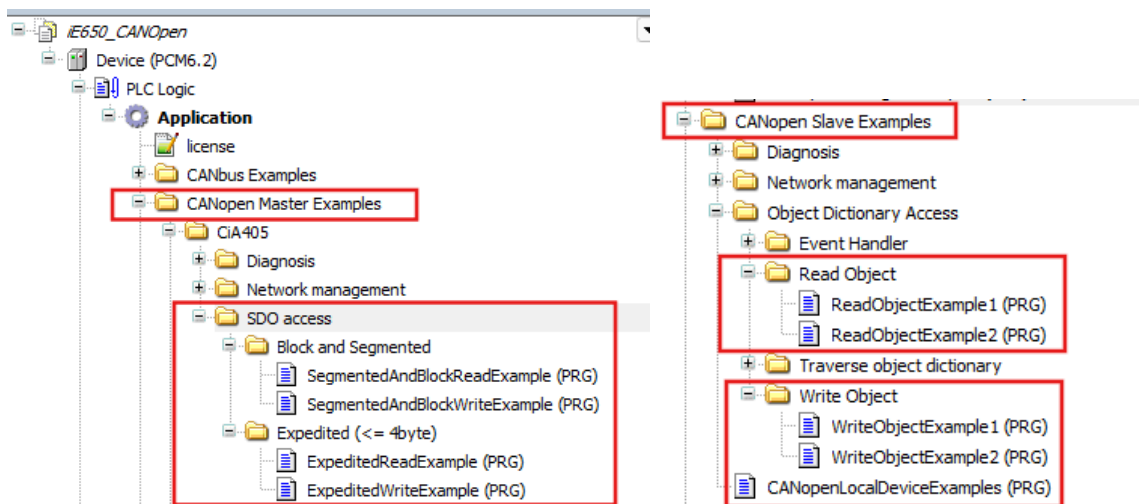
1 // COM 1
2 holding_reg[0] := holding_reg[0];
3 input_reg[0]   := input_reg[0];
4 // COM 2
5 write_holding_reg[0] := write_holding_reg[0];
6 read_input_reg[0]   := read_input_reg[0];
```

After successful download of the project, when values are passed into the **write_holding_reg** variable the same is reflected in the **holding_reg** variable. The values passed into **input_reg** variable is reflected into the **read_input_reg** variable. This concludes that COM1 & COM2 are communicating.

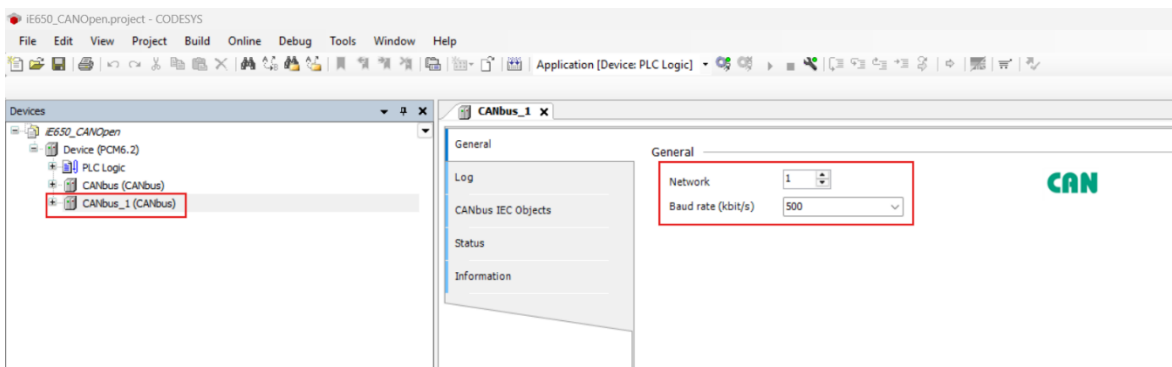
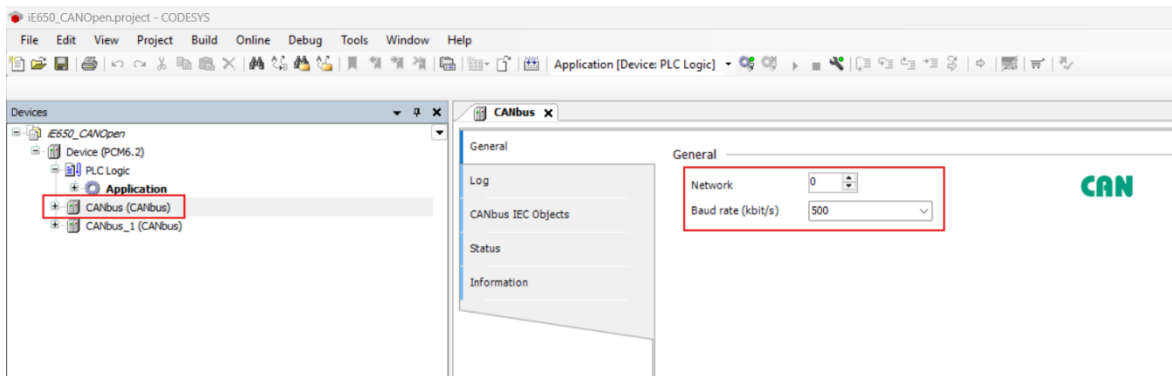
3.1.3 CANopen program configuration

The PCM6.2 module has 2 CAN interfaces, each interface can be used as master or slave.

Please refer to the Demo Project – **Demo 3.1.3_PCM6.2_CANOpen.project**. In this document, we will focus on the programs related to SDO access from the CANopen Master examples and programs related to reading and writing of the CANopen objects from the CANopen Slave examples.



Ensure the CANbus and CANbus_1 network are configured with 0 & 1 respectively and the Baud rate is set to 500 kbits/s. Refer the below image.



Please note that the internal CAN terminating resistor is not working and an external resistor needs to be added to either CAN port 1 or CAN port 2.

There are two methods to read or write the data of the slave device using SDO on the CANopen master. They are

- **Expedited Read/Write** Example when the data is less than or equal to 4 bytes.
- **Segmented & Block Read/Write** Example when the data is more than 4 bytes.

For the **Expedited Read example**, Login to the controller and write TRUE to the variable xEnable. After writing, observe the device type of CANopenRemoteDevice is being read to the variable dwValue. Please refer the image below

Expression	Type	Value	Prepared value	Address	Comment
xEnable	BOOL	FALSE			Set this variable to true to start the example code. Variable will be set to FALSE
dwValue	DWORD	401			read value
dwAbortCode	DWORD	0			contains abort code in case of error
SDO_Read4	CIA405.SDO_READ4				

```

1 IF SDO_Read4.CONFIRM[0] THEN
2 //value was read without error
3 // ==> Copy data from SDO read function block
4 MEMH11s.MemCpy(pbyDest := ADR(dwValue[0]), pbySrc := ADR(SDO_Read4.DATA[1][0]), dwSize := SIZEOF(dwValue[0]));
5 // ==> On big endian systems: dwValue has to be swapped to CANopen byte order (= Little Endian).
6 MEMH11s.SwapLocalToIntel(pAddress := ADR(dwValue[0]), lSize := TO_INT(SIZEOF(dwValue[0]))); //function only swaps on big endian systems
7 xEnable := TRUE;
8 ELSEIF SDO_Read4.ERROR[CANOPEN_KR] < CIA405.CANOPEN_KERNEL_ERROR.CANOPEN_KERNEL_NO_ERROR THEN
9 //Error occurred
10 IF SDO_Read4.ERROR[CANOPEN_KR] = CIA405.CANOPEN_KERNEL_ERROR.CANOPEN_KERNEL_OTHER_ERROR THEN
11 //output error info contains abort code in little endian format
12 MEMH11s.MemCpy(pbyDest := ADR(dwAbortCode[0]), pbySrc := ADR(SDO_Read4.ERRORINFO[0]), dwSize := SIZEOF(dwAbortCode[0]));
13 MEMH11s.SwapLocalToIntel(pAddress := ADR(dwAbortCode[0]), lSize := TO_INT(SIZEOF(dwAbortCode[0]))); //function only swaps on big endian systems
14 END_IF
15 xEnable := FALSE;
16 END_IF
17
18 SDO_Read4(
19 NETWORK_ID := CANbus.NetId + 1, //CODESYS NetId starts by 0 (the number entered into the CANbus configurator); CIA405 NETWORK by 1; That is the reason why we have to increment it here.
20 xEnable := xEnable,
21 TIMEOUT := 0, //Timeout for SDO read in ms. Here: 0 ==> no timeout
22 DEVICE_ID := CANopenRemoteDevice.NodeID, // NodeID of destination device
23 CHANNEL := 0, //SDO channel which should be used. 0 means auto channeling ==> the next free channel will be used. It depends on the slave which settings are working here but 0 and channel 1 is always working.
24 INDEX := 16#1000, //object index which should be read
25 SUBINDEX := 0 //object sub index which should be read
26 );
27 RETURN

```

This can be verified by checking the value of **Device Type Object** in the Object Dictionary of the CANopenRemote device (this is the device that is configured in the CAN2 interface - CANopenLocalDevice)

Index	Name	Data Type	Default Value	Access Type
16#1000	Device Type	UNSIGNED32	16#191	ro
16#1001	Error Register	UNSIGNED8	16#0	ro
16#1003	Predefined Error Field	UNSIGNED8	16#0	rw
16#1005	Sync COB-ID	UNSIGNED32	16#80	rw
16#1006	Communication Cycle Period	UNSIGNED32	16#0	rw
16#1008	Manufacturer device name	VISIBLE_STRING	'CANopenDeviceExample'	const
16#100C	Guard Time	UNSIGNED16	16#0	rw
16#100D	Life Time Factor	UNSIGNED8	0	rw
16#1014	COB-ID EMCY	UNSIGNED32	16#80	rw
16#1016	Consumer Heartbeat Time	UNSIGNED16	16#0	rw
16#1017	Producer Heartbeat Time	UNSIGNED16	16#0	rw
16#1018	Identity Object	UNSIGNED16	16#0	rw
16#1200	ServerSdoParameter			
16#1400	RPDO communication parameter			
16#1600	RPDO mapping parameter			
16#1800	TPDO communication parameter			
16#1A00	TPDO mapping parameter			
16#3000	TestString	VISIBLE_STRING	*	rw
16#6000	Read Input 8 Bit			
16#6200	Write Output 8 Bit			
16#6412	Write Analogue Output 32 Bit			

For the **Expedited Write example**, Login to the controller and write TRUE to the variable xEnable. After writing, the value of the variable dwValue being written to the Object ID Index-16#6412 & Sub index – 1.

This can be verified by monitoring the variable mapped to the Object ID on the Slave device. Please refer the images shown below.

The screenshot shows the 'ExpeditedWriteExample' program. The variable declaration table at the top lists:

Expression	Type	Value	Prepared value	Address	Comment
xEnable	BOOL	FALSE			Set this variable to true to start the example code. Variable will be reset to FALSE
dValue	DINT	10			value which should be written
dwAbortCode	DWORD	0			contains abort code in case of error
SDOWrite4	CIA405_SDO_WRITE4				
RTrig	Standard_R_TRIG				

The code below shows the logic for writing data to the SDO. A red box highlights the following code segment:

```

NETWORK 1 := CANbus.NetId + 1, //CODESYS NetId starts by 0 (the number entered into the CANbus configurator); CIA405 NETWORK by 1; That is the reason why we have to increment it here.
ENABLE := xEnable;
TIMEOUT := 0, //Timeout for SDO write in ms. Here: 0 ==> no timeout
DEVICE := CANopenRemoteDevice.NodeId, // NodeID of destination device
CHANNEL := 0, //SDO channel which should be used. 0 means auto channeling ==> the next free channel will be used. It depends on the slave which settings are working here but 0 and channel 1 is always working.
INDEX := 16#6412, //object index which should be written
SUBINDEX := 1, //object sub index which should be written
DATALENGTH := TO_USINT(SIZEOF(dValue)) //length of data to be written
  
```

The screenshot shows the 'CANopenLocalDevice' configuration window. The 'Object Dictionary' tab is active, displaying a list of objects. A red box highlights the following object:

Index	Name	Data Type	Default Value	Access Type	Variable Mapping
16#1000	Device Type	UNSIGNED32	16#191	ro	
16#1001	Error Register	UNSIGNED8	16#0	ro	
16#1003	Predefined Error Field				
16#1005	Sync COB-ID	UNSIGNED32	16#80	rw	
16#1006	Communication Cycle Period	UNSIGNED32	16#0	rw	
16#1008	Manufacturer device name	VISIBLE_STRING	'CANopenDeviceExample'	const	
16#100C	Guard Time	UNSIGNED16	16#0	rw	
16#100D	Life Time Factor	UNSIGNED8	0	rw	
16#1014	COB-ID EMCY	UNSIGNED32	\$NODEID+16#80	rw	
16#1016	Consumer Heartbeat Time				
16#1017	Producer Heartbeat Time	UNSIGNED16	16#0	rw	
16#1018	Identity Object				
16#1200	ServerSdoParameter				
16#1400	RPDO communication parameter				
16#1600	RPDO mapping parameter				
16#1800	TPDO communication parameter				
16#1A00	TPDO mapping parameter				
16#3000	TestString	VISIBLE_STRING	"	rw	CANopenLocalDeviceExamples.stTestString
16#6000	Read Input 8 Bit				
16#6200	Write Output 8 Bit				
16#6412	Write Analogue Output 32 Bit				
16#6412:16#00	Number of Elements	UNSIGNED8	16#1	ro	
16#6412:16#01	AnalogueOutput32	INTEGER32	0	rww	CANopenLocalDeviceExamples.dTestValue

The variable mapped to the Object ID Index 16#6412, Sub index 1 is called in the program CANopenLocalDeviceExamples, the value of this variable can be monitored in the **Watch window** of the same program as shown below.

The screenshot shows the 'CANopenLocalDeviceExamples' program. The 'Watch' window displays the following variables:

Expression	Type	Value
strTestString	STRING	"
dTestValue	DINT	10
xEventHandlerRegistered	BOOL	TRUE
eventHandler	ObjectDictionaryEventHandler	

For the **Segmented & Blocked Read Example**, Login to the controller and write TRUE to the variable xEnable. After writing, observe the device name of CANopenRemoteDevice is being read to the variable psData as shown below.

```

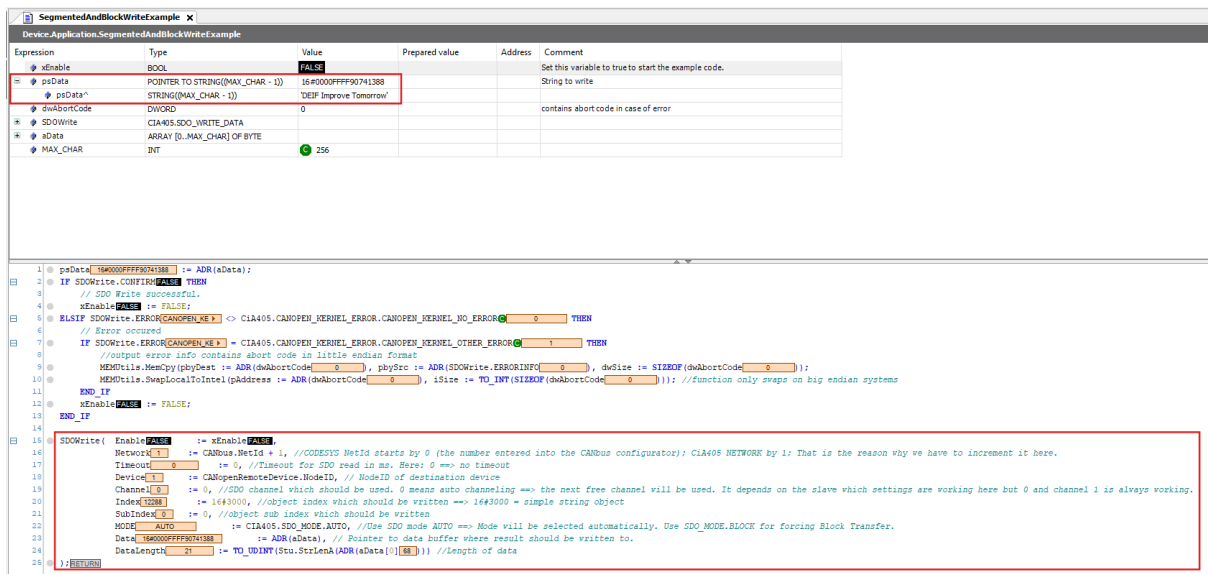
1  psData[16#0000FFFF74110] := ADR(aData);
2  RTrig(CIA405_SDO_READ_DATA, FALSE, xEnable);
3
4  IF RTrig.Q THEN
5      // Rising edge for xEnable detected ==> Prepare SDO write function block
6
7      // ==> Initialize read buffer with 0 (clear old read results)
8      MEMTLibs.MemSet(pbyBuffer := ADR(aData[0]), byValue := 0, dwSize := SIZEOF(aData));
9
10     // ==> Set udiDataLength to read buffer length.
11     // Note: Do not forget setting udiDataLength each time when you start a new SDO read operation.
12     // This is necessary because udiDataLength is a VAR_IN_OUT variable and will be modified by SDORead function block!
13     // It contains the number of valid bytes read by the function block.
14     udiDataLength := SIZEOF(aData) - 1; // -1 only needed because CANopen strings are not null terminated! So we need one additional byte for the null termination. ==> Not needed for other datatypes!
15 END_IF
16
17 IF SDORead.CONFIRM THEN
18     // SDO Read successful. aData contains data (= Device Name); udiDataLength contains number of bytes read (string length without null termination).
19     xEnable := FALSE;
20     ELIF SDORead.ERROR(CANOPEN_REMOTE_DEVICE) <> CIA405_CANOPEN_REMOTE_DEVICE_ERROR_CANOPEN_REMOTE_DEVICE_ERROR_NO_ERROR THEN
21         // Error occurred
22         IF SDORead.ERROR(CANOPEN_REMOTE_DEVICE) = CIA405_CANOPEN_REMOTE_DEVICE_ERROR_CANOPEN_REMOTE_DEVICE_ERROR_OTHER_ERROR THEN
23             //output error info contains short code in little endian format
24             MEMTLibs.MemCpy(pbyDest := ADR(dwAbortCode), pbySrc := ADR(SDORead.ERRORINFO), dwSize := SIZEOF(dwAbortCode));
25             MEMTLibs.SwapLocalToIntel(pbyDest := ADR(dwAbortCode), iSize := TO_INT(SIZEOF(dwAbortCode))); //function only swaps on big endian systems
26         END_IF
27         xEnable := FALSE;
28     END_IF
29 END_IF
30
31 SDORead(
32     Enable := xEnable,
33     Network := CANbusNetId + 1, //CODESYS NetId starts by 0 (the number entered into the CANbus configurator); CIA405 NETWORK by 1; That is the reason why we have to increment it here.
34     Timeout := 0, //Timeout for SDO read in ms. Here: 0 ==> no timeout
35     DeviceID := CANopenRemoteDevice.NodeID, // NodeID of destination device
36     Channel := 0, //SDO channel which should be used. 0 means auto channeling ==> the next free channel will be used. It depends on the slave which settings are working here but 0 and channel 1 is always working.
37     Index := 16#1008, //Object index which should be read ==> 16#1008Sub0 = Device Name
38     SubIndex := 0, //Object sub index which should be read ==> 16#1008Sub0 = Device Name
39     Mode := AUTO, //CIA405_SDO_MODE.AUTO, //Use SDO mode AUTO ==> Mode will be selected automatically. Use SDO_MODE.BLOCK for forcing Block Transfer.
40     Data := ADR(aData), // Pointer to data buffer where result should be written to.
41     DataLength := udiDataLength, //VAR_IN: Length of data buffer; VAR_OUT: Length of data written to data buffer.
42 );
43 RETURN

```

This can be verified by checking the value of Object ID 16#1008 in the Object Dictionary of the CANopenRemote device (this is the device that is configured in the CAN2 interface - CANopenLocalDevice)

Index	Name	Data Type	Default Value	Access Type	Variable Mapping
16#1000	Device Type	UNSIGNED32	16#191	ro	
16#1001	Error Register	UNSIGNED8	16#0	ro	
16#1003	Predefined Error Field				
16#1005	Sync COB-ID	UNSIGNED32	16#80	rw	
16#1006	Communication Cycle Period	UNSIGNED32	16#0	rw	
16#1008	Manufacturer device name	VISIBLE_STRING	'CANopenDeviceExample'	const	
16#100C	Guard Time	UNSIGNED16	16#0	rw	
16#100D	Life Time Factor	UNSIGNED8	0	rw	
16#1014	COB-ID EMCY	UNSIGNED32	\$NODEID+16#80	rw	
16#1016	Consumer Heartbeat Time				
16#1017	Producer Heartbeat Time	UNSIGNED16	16#0	rw	
16#1018	Identity Object				
16#1200	ServerSdoParameter				
16#1400	RPDO communication parameter				
16#1600	RPDO mapping parameter				
16#1800	TPDO communication parameter				
16#1A00	TPDO mapping parameter				
16#3000	TestString	VISIBLE_STRING	"	rw	CANopenLocalDeviceExamples.strTestString
16#6000	Read Input 8 Bit				
16#6200	Write Output 8 Bit				
16#6412	Write Analogue Output 32 Bit				

For the **Segmented & Blocked Write Example**, Login to the controller and write TRUE to the variable xEnable. After writing, the value in the String pointer variable psData being written to the Object ID Index - 16#3000 & Sub Index – 0. This can be verified by monitoring the variable mapped to the Object ID on the Slave device. Please refer the images shown below.



CANopenLocalDevice						
General						
+ Add Object + Add Subobject X Delete Clone Import from EDS...						
Object Dictionary	Index	Name	Data Type	Default Value	Access Type	Variable Mapping
	16#1000	Device Type	UNSIGNED32	16#191	ro	
	16#1001	Error Register	UNSIGNED8	16#0	ro	
	16#1003	Predefined Error Field				
	16#1005	Sync COB-ID	UNSIGNED32	16#80	rw	
	16#1006	Communication Cycle Period	UNSIGNED32	16#0	rw	
	16#1008	Manufacturer device name	VISIBLE_STRING	'CANopenDeviceExample'	const	
	16#100C	Guard Time	UNSIGNED16	16#0	rw	
	16#100D	Life Time Factor	UNSIGNED8	0	rw	
	16#1014	COB-ID EMCY	UNSIGNED32	\$NODEID+16#80	rw	
	16#1016	Consumer Heartbeat Time				
	16#1017	Producer Heartbeat Time	UNSIGNED16	16#0	rw	
	16#1018	Identity Object				
	16#1200	ServerSdoParameter				
	16#1400	RPDO communication parameter				
	16#1600	RPDO mapping parameter				
	16#1800	TPDO communication parameter				
	16#1A00	TPDO mapping parameter				
	16#3000	TestString	VISIBLE_STRING	*	rw	CANopenLocalDeviceExamples.strTestString
	16#6000	Read Input 8 Bit				
	16#6200	Write Output 8 Bit				
	16#6412	Write Analogue Output 32 Bit				

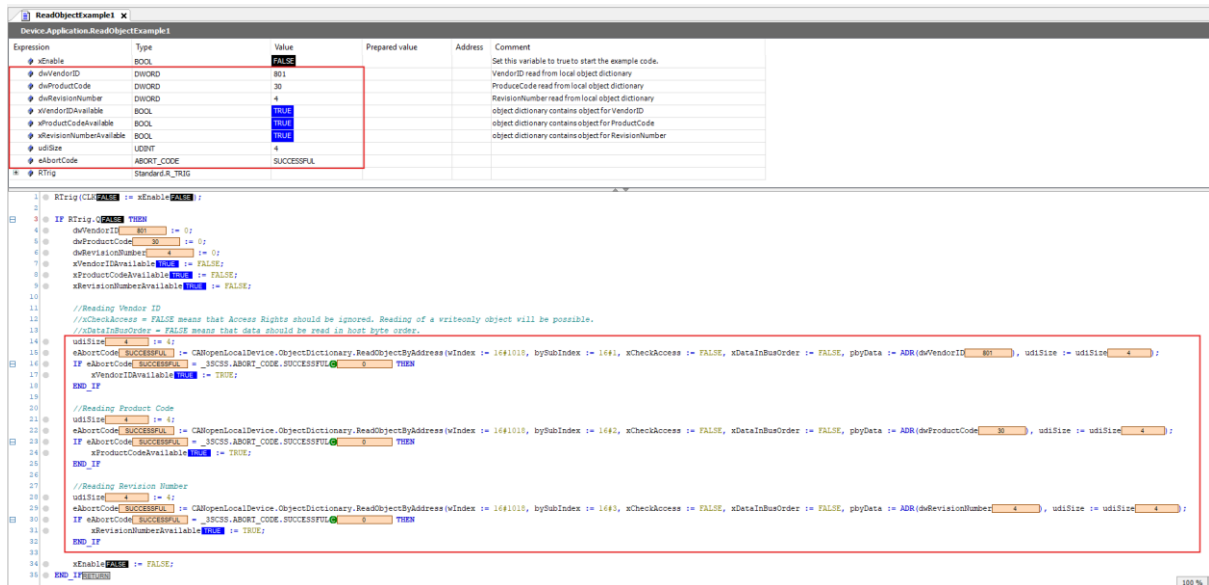
The variable mapped to the Object ID Index 16#3000 is called in the program CANopenLocalDeviceExamples, the value of this variable can be monitored in the **Watch window** of the same program as shown below.

CANopenLocalDeviceExamples					
Device.Application.CANopenLocalDeviceExamples					
Expression	Type	Value	Prepared value	Address	Comment
strTestString	STRING	'DEIF Improve Tomorrow'			Variable strTestString is mapped to object 16#3000; It can be read and written b
diTestValue	DINT	10			
xEventHandlerRegistered	BOOL	TRUE			
eventHandler	ObjectDictionaryEventHandler				

There are two methods to read/write the value of the Object IDs of the CANopen Slave device. They are

- Read/Write an object value by **Object Address** (Example1)
- Read/Write an object value by **Object method** (Example2)

For the Read Object Example1, Login to the controller and write TRUE to the variable xEnable. After writing, observe the values of the Object ID index 16#1018 being read, the status of the data availability is also captured and, a variable showing the Read object status. Please refer the image below.



The values can be verified by checking the CANopenLocalDevice Object Dictionary as shown below

CANopenLocalDevice						
Add Object Add Subobject Delete Clone Import from EDS...						
General						
Object Dictionary	Index	Name	Data Type	Default Value	Access Type	Variable Mapping
PDOs	16#1000	Device Type	UNSIGNED32	16#191	ro	
	16#1001	Error Register	UNSIGNED8	16#0	ro	
	16#1003	Predefined Error Field	UNSIGNED8	16#0	ro	
	16#1005	Sync COB-ID	UNSIGNED32	16#80	rw	
Log	16#1006	Communication Cycle Period	UNSIGNED32	16#0	rw	
	16#1008	Manufacturer device name	VISIBLE_STRING	'CANopenDeviceExample'	const	
	16#100C	Guard Time	UNSIGNED16	16#0	rw	
	16#100D	Life Time Factor	UNSIGNED8	0	rw	
CANopen I/O Mapping	16#1014	COB-ID EMCY	UNSIGNED32	\$NODEID+16#80	rw	
	16#1016	Consumer Heartbeat Time	UNSIGNED32	16#0	rw	
	16#1017	Producer Heartbeat Time	UNSIGNED16	16#0	rw	
	16#1018	Identity Object	UNSIGNED32	16#0	rw	
CANopen IEC Objects	16#1018:16#00	Number of Entries	UNSIGNED8	3	ro	
	16#1018:16#01	Vendor Id	UNSIGNED32	801	ro	
	16#1018:16#02	Product Code	UNSIGNED32	30	ro	
	16#1018:16#03	Revision Number	UNSIGNED32	4	ro	
Status	16#1200	ServerSdoParameter	UNSIGNED32	16#0	rw	
	16#1400	RPDO communication parameter	UNSIGNED32	16#0	rw	
	16#1600	RPDO mapping parameter	UNSIGNED32	16#0	rw	
	16#1800	TPDO communication parameter	UNSIGNED32	16#0	rw	
Information	16#1A00	TPDO mapping parameter	UNSIGNED32	16#0	rw	
	16#3000	TestString	VISIBLE_STRING	"	rw	CANopenLocalDeviceExamples.strTestString
	16#6000	Read Input 8 Bit	UNSIGNED8	16#0	rw	
	16#6200	Write Output 8 Bit	UNSIGNED8	16#0	rw	
	16#6412	Write Analogue Output 32 Bit	UNSIGNED32	16#0	rw	

For the Write Object Example1, Login to the controller and write TRUE to the variable xEnable. After writing, the value of the variable diValue is written to the Object ID index 16#1412 and subindex 1, this can be verified by monitoring the value mapped to the Object ID, which is called in the program CANopenLocalDeviceExamples.

WriteObjectExample1

Expression	Type	Value	Prepared value	Address	Comment
xEnable	BOOL	FALSE			Set this variable to true to start the example code.
dwSyncCOBID	DWORD	128			SyncCOBID which should be written to local object dictionary
diValue	DINT	1234			
xWriteSuccessful	BOOL	TRUE			
uWriteSize	UDINT	4			
AbortCode	ABORT_CODE	SUCCESSFUL			
RTsig	Standard_R_TRIG				

```

1 RTrig(C1) := xEnable;
2 IF RTrig THEN
3   udWriteSize := SINT(1234);
4   AbortCode := CANOpenLocalDevice.ObjectDictionary.WriteObjectByAddress(
5     dwIndex := 16#6412, bySubIndex := 16#1, xCheckAccess := TRUE, xDataInBusOrder := FALSE, pbyData := ADDR(diValue), udWriteSize := udWriteSize);
6   xWriteSuccessful := AbortCode = _SUCS.ABORT_CODE.SUCCESSFUL;
7   xEnable := FALSE;
8 END_IF

```

Address	Write Analogue Output 32 Bit	Value	Comment
16#6412:16#00	Number of Elements	UNSIGNED8	16#1
16#6412:16#01	AnalogueOutput32	INTEGER32	0

ro rww CANopenLocalDeviceExamples.diTestValue

CANopenLocalDeviceExamples

Expression	Type	Value
strTestString	STRING	'DEIF Improve Tomorrow'
diTestValue	DINT	1234
xEventHandlerRegistered	BOOL	TRUE
eventHandler	ObjectDictionaryEventHandler	

For the Read Object Example2, Login to the controller and write TRUE to the variable xEnable. After writing, observe the values of the Object ID index 16#1018 being read, the status of the data availability is also captured and, a variable showing the Read object status. Please refer the image below.

ReadObjectExample2

Expression	Type	Value	Prepared value	Address	Comment
xEnable	BOOL	FALSE			Set this variable to true to start the example code.
dwVendorID	DWORD	801			VendorID read from local object dictionary
dwProductCode	DWORD	30			ProductCode read from local object dictionary
dwRevisionNumber	DWORD	4			RevisionNumber read from local object dictionary
xVendorIDAvailable	BOOL	TRUE			object dictionary contains object for VendorID
xProductCodeAvailable	BOOL	TRUE			object dictionary contains object for ProductCode
xRevisionNumberAvailable	BOOL	TRUE			object dictionary contains object for RevisionNumber
uIdentifyObject	_SUCS.SUCSObject	16#0000FFFF740C28			
uWriteSize	UDINT	4			
AbortCode	ABORT_CODE	SUCCESSFUL			
RTsig	Standard_R_TRIG				

```

4 dwRevisionNumber := 0;
5 xVendorIDAvailable := FALSE;
6 xProductCodeAvailable := FALSE;
7 xRevisionNumberAvailable := FALSE;
8 IF xIdentifyObject = 0 THEN
9   uIdentifyObject := CANOpenLocalDevice.ObjectDictionary.GetObject(16#1018);
10 END_IF
11 IF uIdentifyObject < 0 THEN
12   //Reading Vendor ID
13   //CheckAccess = FALSE means that Access Rights should be ignored. Reading of a writable object will be possible.
14   //DataInBusOrder = FALSE means that data should be read in host byte-order.
15   udWriteSize := 4;
16   AbortCode := CANOpenLocalDevice.ObjectDictionary.ReadObject(uIdentifyObject, bySubIndex := 16#1, xCheckAccess := FALSE, xDataInBusOrder := FALSE, pbyData := ADDR(dwVendorID), udWriteSize := udWriteSize);
17   IF AbortCode = _SUCS.ABORT_CODE.SUCCESSFUL THEN
18     xVendorIDAvailable := TRUE;
19   END_IF
20   //Reading Product Code
21   udWriteSize := 4;
22   AbortCode := CANOpenLocalDevice.ObjectDictionary.ReadObject(uIdentifyObject, bySubIndex := 16#2, xCheckAccess := FALSE, xDataInBusOrder := FALSE, pbyData := ADDR(dwProductCode), udWriteSize := udWriteSize);
23   IF AbortCode = _SUCS.ABORT_CODE.SUCCESSFUL THEN
24     xProductCodeAvailable := TRUE;
25   END_IF
26   //Reading Revision Number
27   udWriteSize := 4;
28   AbortCode := CANOpenLocalDevice.ObjectDictionary.ReadObject(uIdentifyObject, bySubIndex := 16#3, xCheckAccess := FALSE, xDataInBusOrder := FALSE, pbyData := ADDR(dwRevisionNumber), udWriteSize := udWriteSize);
29   IF AbortCode = _SUCS.ABORT_CODE.SUCCESSFUL THEN
30     xRevisionNumberAvailable := TRUE;
31   END_IF
32 END_IF

```

For the Write Object Example2, Login to the controller and write TRUE to the variable xEnable. After writing, the value of the variable diValue is written to the Object ID index 16#1412 and subindex 1, this can be verified by monitoring the value mapped to the Object ID, which is called in the program CANopenLocalDeviceExamples.

WriteObjectExample2

Expression	Type	Value	Prepared value	Address	Comment
xEnable	BOOL	FALSE			Set this variable to true to start the example code.
diSyncCOBID	DINT	128			SyncCOBID which should be written to local object dictionary
diValue	DINT	4321			
writeSuccessful	BOOL	TRUE			
idSyncCOBID	_JES33300Object	16#0000FFFF9074D000			
idSize	UDINT	4			
abortCode	ABORT_CODE	SUCCESSFUL			
RTing	Standard_8_TRIG				

```

1 @ RTing(C16#128 := xEnableFALSE);
2
3 IF RTing.C16#128 THEN
4   xWriteSuccessful := FALSE;
5   IF idSyncCOBID = 0 THEN
6     idSyncCOBID := CANopenLocalDevice.ObjectDictionary.GetObjectIndex(16#1412);
7   END_IF
8
9   IF idSyncCOBID <> 0 THEN
10    udIndex(4) := XINDEX(diSyncCOBID);
11    //xCheckAccess = TRUE means that AccessType should be checked. Writing on a readonly object will fail in this case. By setting xCheckAccess to false writing on a readonly object is possible.
12    //xDataIsInBusOrder = FALSE means that data is in host byte order.
13    abortCode(SUCCESSFUL) := CANopenLocalDevice.ObjectDictionary.WriteObject(idSyncCOBID, bySubIndex := 1, xCheckAccess := TRUE, xDataIsInBusOrder := FALSE, pData := ADDR(diValue), udIndex := udIndex(4));
14    xWriteSuccessful := abortCode(SUCCESSFUL) = _JES33300_ABORT_CODE_SUCCESSFUL;
15  END_IF
16  xEnable := TRUE;
17 END_IF

```

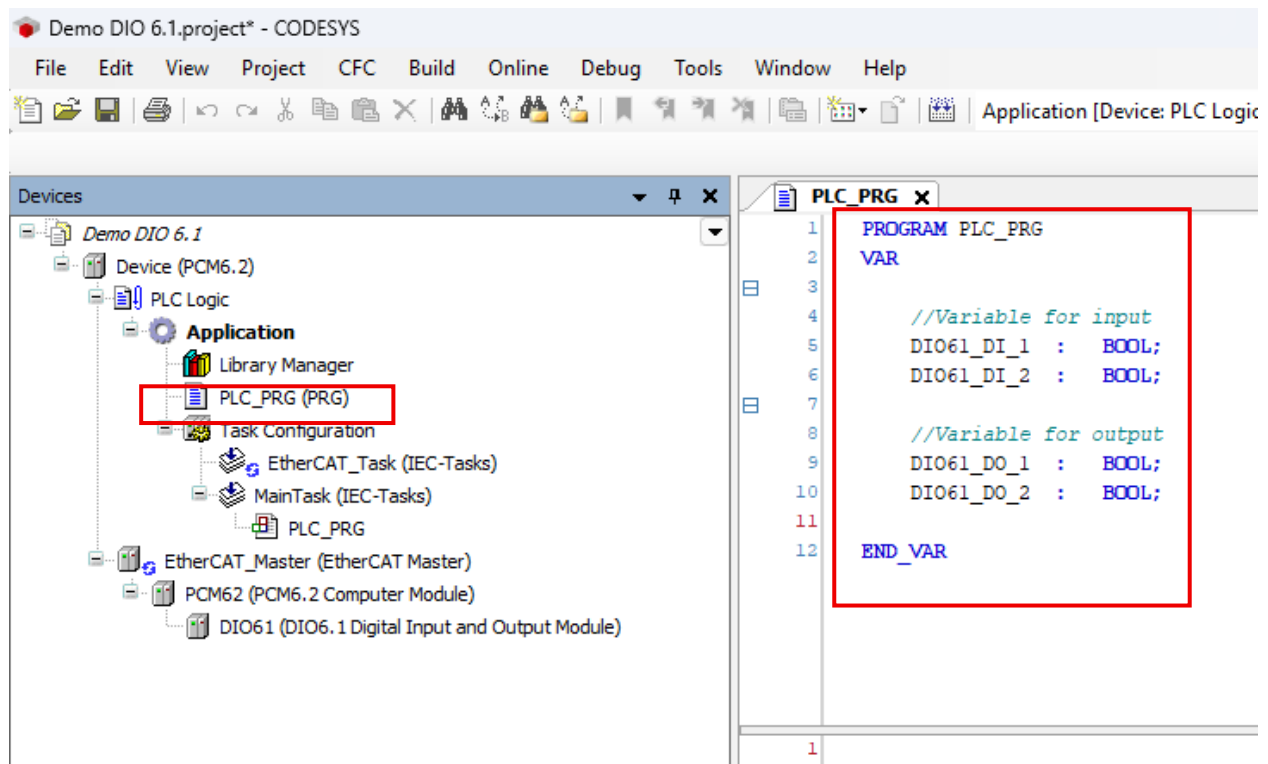
CANopenLocalDeviceExamples

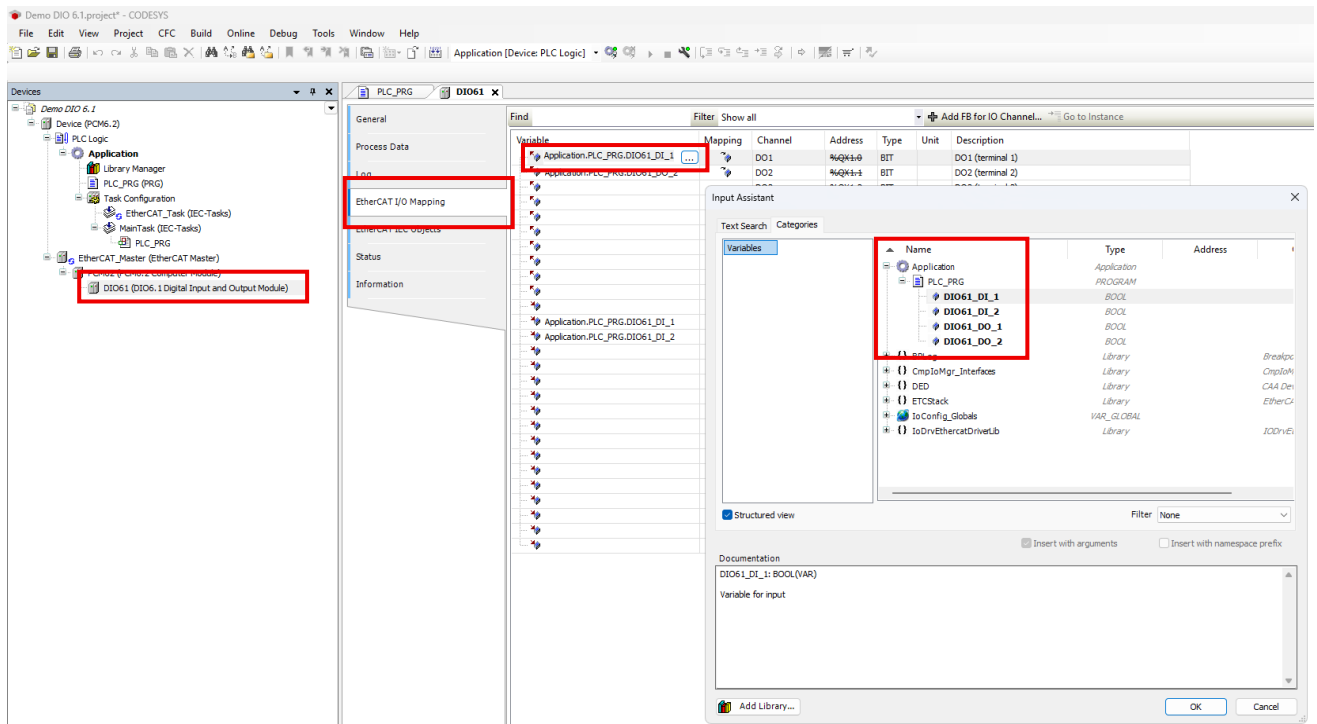
Expression	Type	Value
strTestString	STRING	'DEIF Improve Tomorrow'
diTestValue	DINT	4321
xEventHandlerRegistered	BOOL	TRUE
eventHandler	ObjectDictionaryEventHandler	

3.2 DIO6.1 - Configuration example

The DIO6.1 is a digital input/output module with 10 digital outputs and 16 digital inputs.

In this example, variables are declared in **PLC_PRG** and mapped to the relevant inputs and outputs via the module's **EtherCAT I/O Mapping**.





3.3 AIO 6.1 - Configuration example

The AIO6.1 is an analog input/output module featuring 2 analog outputs and 16 analog inputs. Each channel must be individually configured using startup parameters, based on the signal type of the connected sensor. Startup parameters cannot be applied in bulk, configuration must be performed separately for each channel.

3.3.1 AIO 6.1 – Output startup configuration

- Select the **AIO6.1** module
- Go to **Startup Parameters**
- Click **Add**
- Select the **AO 1 Advanced settings**
- Click **AO 1 Output type**
- Select **Value** - data range -10~10 / 0~10 V / 0~20 mA / 4~20 mA.

The screenshot shows the CODESYS software interface for configuring the AIO6.1 module. The 'Startup Parameters' tab is selected, and the 'Add' button is highlighted. A table lists the parameters for the AIO6.1 module, including AO1 and AO2 output types. The 'AO1 Output type' parameter is highlighted, and a dialog box is open showing the selection of 'AO1 Advanced settings' and 'AO1 Output type'. The 'Value' dropdown is set to '0 - 10 V'.

Line	Index/Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#810D:16#11	AI1 Input type	0 - 10 V	8	☐	☐	0	
2	16#811D:16#11	AI2 Input type	4 - 20 mA	8	☐	☐	0	
3	16#800D:16#11	AO1 Output type	0 - 10 V	8	☐	☐	0	
4	16#801D:16#11	AO2 Output type	4 - 20 mA	8	☐	☐	0	

Select Item from Object Directory

Index/Subindex	Name	Flags	Type	Default
16#10F3:16#00	Diagnosis History			
16#1C12:16#00	RxPDO-Assign			
16#1C13:16#00	TxPDO-Assign			
16#800D:16#00	AO1 Advanced settings			
16#800D:16#11	AO1 Output type	RW	USINT	16#0e
16#800F:16#00	AO1 Vendor data			
16#801D:16#00	AO2 Advanced settings			
16#801F:16#00	AO2 Vendor data			
16#810D:16#00	AI1 Advanced settings			
16#810F:16#00	AI1 Vendor data			
16#811D:16#00	AI2 Advanced settings			
16#811F:16#00	AI2 Vendor data			
16#812D:16#00	AI3 Advanced settings			
16#812F:16#00	AI3 Vendor data			
16#813D:16#00	AI4 Advanced settings			
16#813F:16#00	AI4 Vendor data			

Name: AO1 Output type

Index: 16# 800D Bit length: 8

SubIndex: 16# 11 Value: 0 - 10 V

☐ Byte array

USINT

OK Cancel

3.3.2 AIO 6.1 – Input startup configuration

- Select the **AIO6.1** module
- Go to **Startup Parameters**
- Click **Add**
- Select the **AO 1 Advanced settings**
- Click **AO 1 Output type**
- Select **Value** data range -10~10 / 0~10 V / 0~20 mA / 4~20 mA.

The screenshot shows the CODESYS software interface for configuring the AIO6.1 module. The 'Startup Parameters' tab is selected, and the 'Add' button is highlighted. A table lists the parameters, including AI1 Input type, AI2 Input type, AO1 Output type, and AO2 Output type. A dialog box is open for selecting an object from the directory, showing 'AI1 Input type' selected. The dialog also shows the 'Value' field set to '0 - 10 V'.

Line	Index/Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#810D:16#11	AI1 Input type	0 - 10 V	8	☐	☐	0	
2	16#811D:16#11	AI2 Input type	4 - 20 mA	8	☐	☐	0	
3	16#800D:16#11	AO1 Output type	0 - 10 V	8	☐	☐	0	
4	16#801D:16#11	AO2 Output type	4 - 20 mA	8	☐	☐	0	

Select Item from Object Directory

Index/Subindex	Name	Flags	Type	Default
* 16#10F3:16#00	Diagnosis History			
* 16#1C12:16#00	RxPDO-Assign			
* 16#1C13:16#00	TxPDO-Assign			
* 16#800D:16#00	AO1 Advanced settings			
* 16#800F:16#00	AO1 Vendor data			
* 16#801D:16#00	AO2 Advanced settings			
* 16#801F:16#00	AO2 Vendor data			
* 16#810D:16#00	AI1 Advanced settings			
* 16#810F:16#00	AI1 Input type	RW	USINT	16#0e
* 16#811F:16#00	AI1 Vendor data			
* 16#811D:16#00	AI2 Advanced settings			
* 16#811F:16#00	AI2 Vendor data			
* 16#812D:16#00	AI3 Advanced settings			
* 16#812F:16#00	AI3 Vendor data			
* 16#813D:16#00	AI4 Advanced settings			
* 16#813F:16#00	AI4 Vendor data			

Name: AI1 Input type

Index: 16# 810D Bit length: 8

SubIndex: 16# 11 Value: 0 - 10 V

☐ Byte array USINT

OK Cancel

After configuring the analog startup parameters, they can be edited, removed, or reordered via the toolbar. The **Add** function allows configuration of only one channel at a time, so it must be used multiple times to configure additional parameters.

General

Process Data

Startup Parameters

Log

EtherCAT I/O Mapping

EtherCAT IEC Objects

Add

Edit

Delete

Move Up

Move Down

Line	Index:Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#810D:16#11	AI1 Input type	-10 - 10 V	8	☐	☐	0	
2	16#811D:16#11	AI2 Input type	4 - 20 mA	8	☐	☐	0	
3	16#800D:16#11	AO1 Output type	-10 - 10 V	8	☐	☐	0	
4	16#801D:16#11	AO2 Output type	4 - 20 mA	8	☐	☐	0	

value, appropriate scaling must be applied.

3.3.3 AIO6.1 - Variable mapping

In this example, variables are declared in **PLC_PRG** and mapped to the relevant inputs and outputs via the module's **EtherCAT I/O Mapping**.

```

1  PROGRAM PLC_PRG
2  VAR
3
4      AIO61_AI1  : INT;      //Analog input 1
5      AIO61_AI2  : INT;      //Analog input 2
6
7      rAI1_IN_V   : REAL;    //Scaled input value
8      rAI2_IN_mA  : REAL;    //Scaled input value
9
10     AIO61_AO1   : INT;      //Analog output 1
11     AIO61_AO2   : INT;      //Analog output 2
12
13     rAO1_Out_V  : REAL;    //Scaled output value
14     rAO2_Out_mA : REAL;    //Scaled output value
15
16
17 END_VAR

```

The screenshot shows the Siemens STEP 7 software interface. The 'AIO61' module is selected in the 'Devices' tree. The 'EtherCAT I/O Mapping' tab is active, displaying a table of variables and their mappings. The 'Add FB for IO Channel...' dialog is open, showing the 'Variables' tab with a list of variables including AIO61_AI1, AIO61_AI2, rAI1_IN_V, rAI2_IN_mA, AIO61_AO1, and rAO1_Out_V. The 'AIO61 (AIO6.1 Analog I/O Module)' is selected in the 'Devices' tree.

3.3.4 CODESYS program example

```

1  PROGRAM PLC_PRG
2  VAR
3
4      AIO61_AI1  : INT;      //Analog input 1
5      AIO61_AI2  : INT;      //Analog input 2
6
7      rAI1_IN_V  : REAL;     //Scaled input value
8      rAI2_IN_mA : REAL;     //Scaled input value
9
10     AIO61_AO1   : INT;      //Analog output 1
11     AIO61_AO2   : INT;      //Analog output 2
12
13     rAO1_Out_V  : REAL;     //Scaled output value
14     rAO2_Out_mA : REAL;     //Scaled output value
15
16
17 END_VAR

1
2  rAI1_IN_V := FC_AnalogScaling( // Reading the input value and scaling it to the type selected
3      InputValue:= INT_TO_REAL(AIO61_AI1) ,
4      InputMin:= 0,              // Minimum RAW value
5      InputMax:= 32767,          // Maximum RAW value
6      OutputMin:= -10,           // Minimum value for the configured type
7      OutputMax:= 10,            // Minimum value for the configured type
8      Error=> );
9
10
11  rAI2_IN_mA := FC_AnalogScaling( // Reading the input value and scaling it to the type selected
12      InputValue:= INT_TO_REAL(AIO61_AI2) ,
13      InputMin:= 0,              // Minimum RAW value
14      InputMax:= 32767,          // Maximum RAW value
15      OutputMin:= 4,             // Minimum value for the configured type
16      OutputMax:= 20,            // Maximum value for the configured type
17      Error=> );
18
19  AIO61_AO1 := REAL_TO_INT(FC_AnalogScaling( // Reading the input value and scaling it to the type selected
20      InputValue:= rAO1_Out_V ,
21      InputMin:= -10,            // Minimum value for the configured type
22      InputMax:= 10,             // Maximum value for the configured type
23      OutputMin:= 0,             // Minimum RAW value
24      OutputMax:= 32767,         // Maximum RAW value
25      Error=> ));
26
27  AIO61_AO2 := REAL_TO_INT(FC_AnalogScaling( // Reading the input value and scaling it to the type selected
28      InputValue:= rAO2_Out_mA ,
29      InputMin:= 4,              // Minimum value for the configured type
30      InputMax:= 20,             // Maximum value for the configured type
31      OutputMin:= 0,             // Minimum RAW value
32      OutputMax:= 32767,         // Maximum RAW value
33      Error=> ));

```

3.3.5 Scaling example

A **Function** is provided to perform value scaling

```
PLC_PRG  FC_AnalogScaling x AIO61
1  FUNCTION FC_AnalogScaling : REAL
2  VAR_INPUT
3      InputValue : REAL;
4      InputMin   : REAL;
5      InputMax   : REAL;
6      OutputMin  : REAL;
7      OutputMax  : REAL;
8
9  END_VAR
10 VAR_OUTPUT
11     Error      : BOOL;
12
13 END_VAR
14 VAR
15 END_VAR
16
17 Error := (InputMin >= InputMax) OR (OutputMin >= OutputMax) OR (InputValue < InputMin) OR (InputValue > InputMax);
18
19 IF NOT Error THEN
20     FC_AnalogScaling := (InputValue - InputMin) / (InputMax - InputMin) * (OutputMax - OutputMin) + OutputMin;
21 ELSE
22     FC_AnalogScaling := 0;
23 END_IF
```

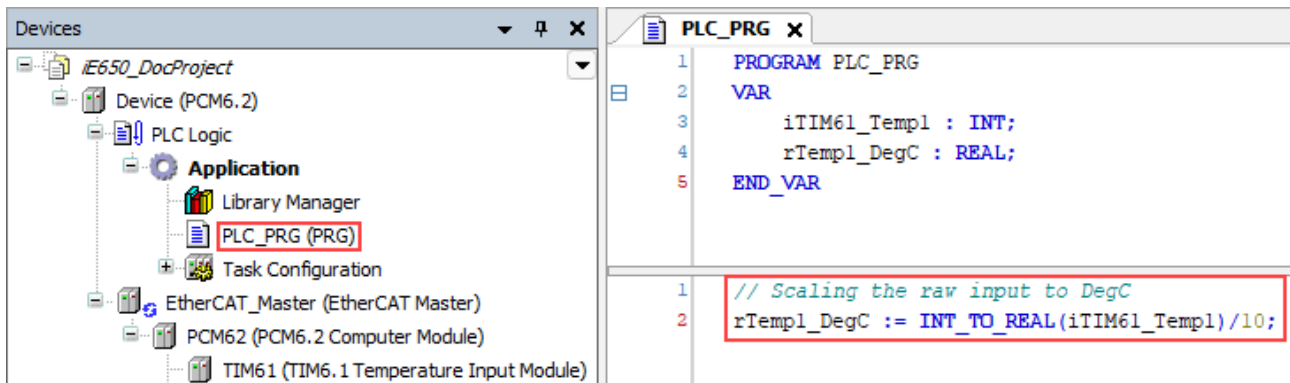
3.4 TIM6.1 Program configuration

TIM6.1 is a temperature input module capable of up to 14 PT100 inputs, dependent on wiring method - For wiring of sensors, please refer to **Installation instructions** document.

After a variable has been declared in the programme, it can be associated with a hardware channel directly in **EtherCAT I/O Mapping**.

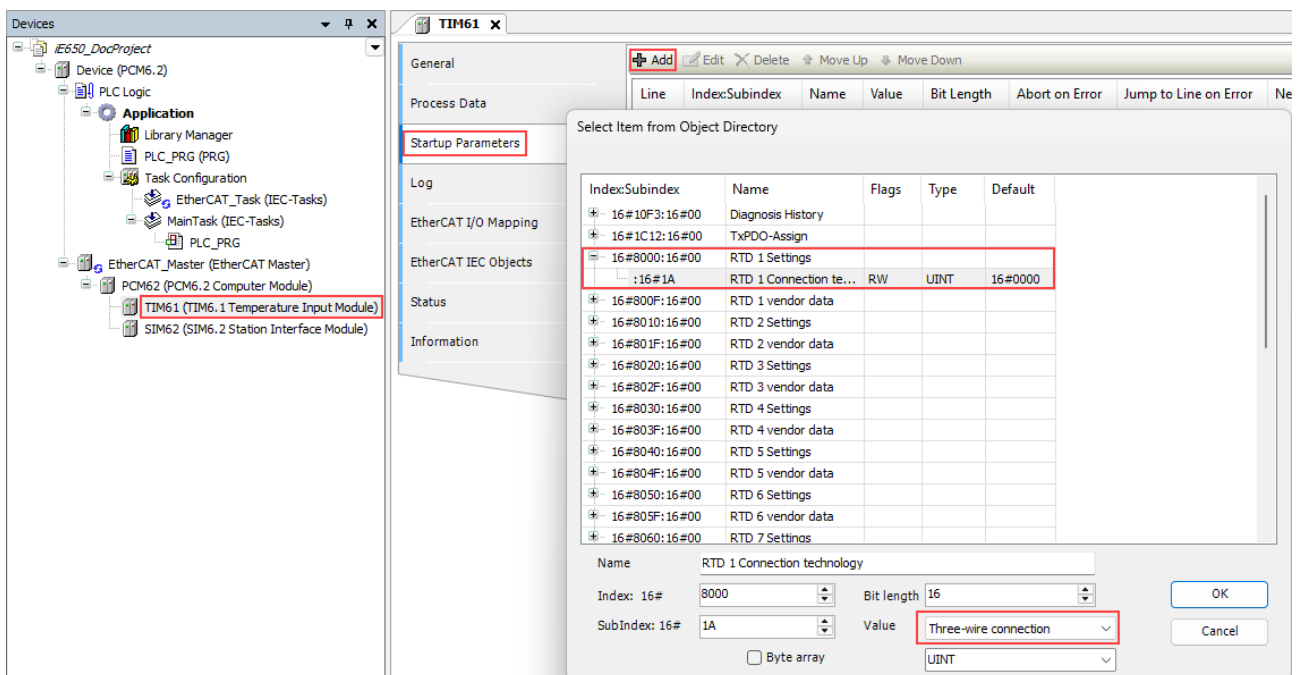
Variable	Mapping	Channel	Address	Type	Unit	Description
		RTD 1 Under range	%IX2.0	BIT		RTD 1 Under range
		RTD 1 Over range	%IX2.1	BIT		RTD 1 Over range
		RTD 1 Error	%IX2.6	BIT		RTD 1 Error
		RTD 1 TxPDO State	%IX3.6	BIT		RTD 1 TxPDO State
		RTD 1 TxPDO Toggle	%IX3.7	BIT		RTD 1 TxPDO Toggle
Application.PLC_PRG.TIM61_Temp1		RTD 1 Value	%IW2	INT		RTD 1 Value
		RTD 2 Under range	%IX6.0	BIT		RTD 2 Under range
		RTD 2 Over range	%IX6.1	BIT		RTD 2 Over range
		RTD 2 Error	%IX6.6	BIT		RTD 2 Error
		RTD 2 TxPDO State	%IX7.6	BIT		RTD 2 TxPDO State

The raw value of the temperature input channel needs to be **multiplied by 0.1** to convert to the actual temperature value.



3.4.1 3-Wire PT100

When an input is wired with a 3-wire sensor, add a startup parameter to change the default settings. Go to **Startup parameter > Add**. Maximize settings-row for the specific RTD. Highlight the RTD Connection-row, it is then possible to change the connection value to three-wire for this input. When **OK** is pressed, the new startup parameter is visible on the list.



3.5 IFM 6.1 Program Configuration

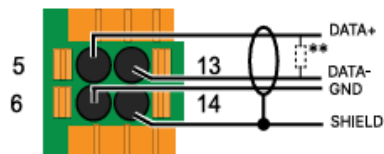
3.5.1 IFM6.1 - Modbus RTU Client

IFM6.1 modules enable the iE 650 PLC to add multiple Modbus RTU Clients.

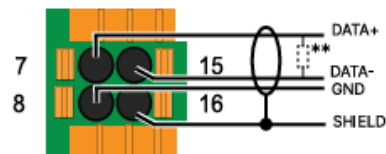
It is very important to ensure the physical installation is using **shielded cable and end-resistors** no matter if it is a small test system or an actual installation. Without correct installation it is easy to end up in CRC and timeout errors.

The IFM6.1 contains two COM ports for RS-485:

COM - RS-485 wiring



COM - RS-485 port 1

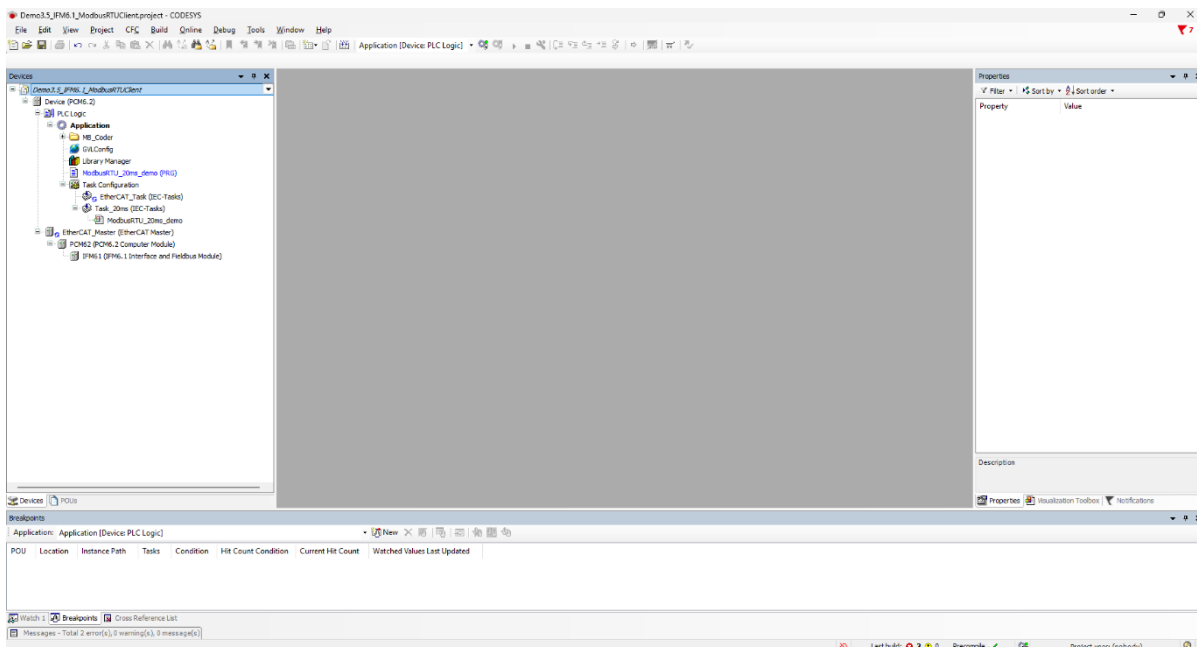


COM - RS-485 port 2

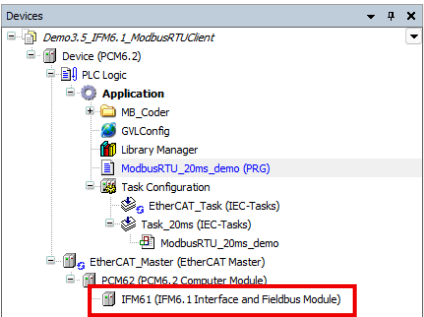
NOTE ** SW configurable (on/off) termination (internally 120 Ω for RS-485). SW configurable (on/off) bias (pull up/pull down, 500 Ω). GND decoupled to Chassis through 1.5 M Ω || 1.5 nF.

3.5.1.1 Configure CODESYS setup

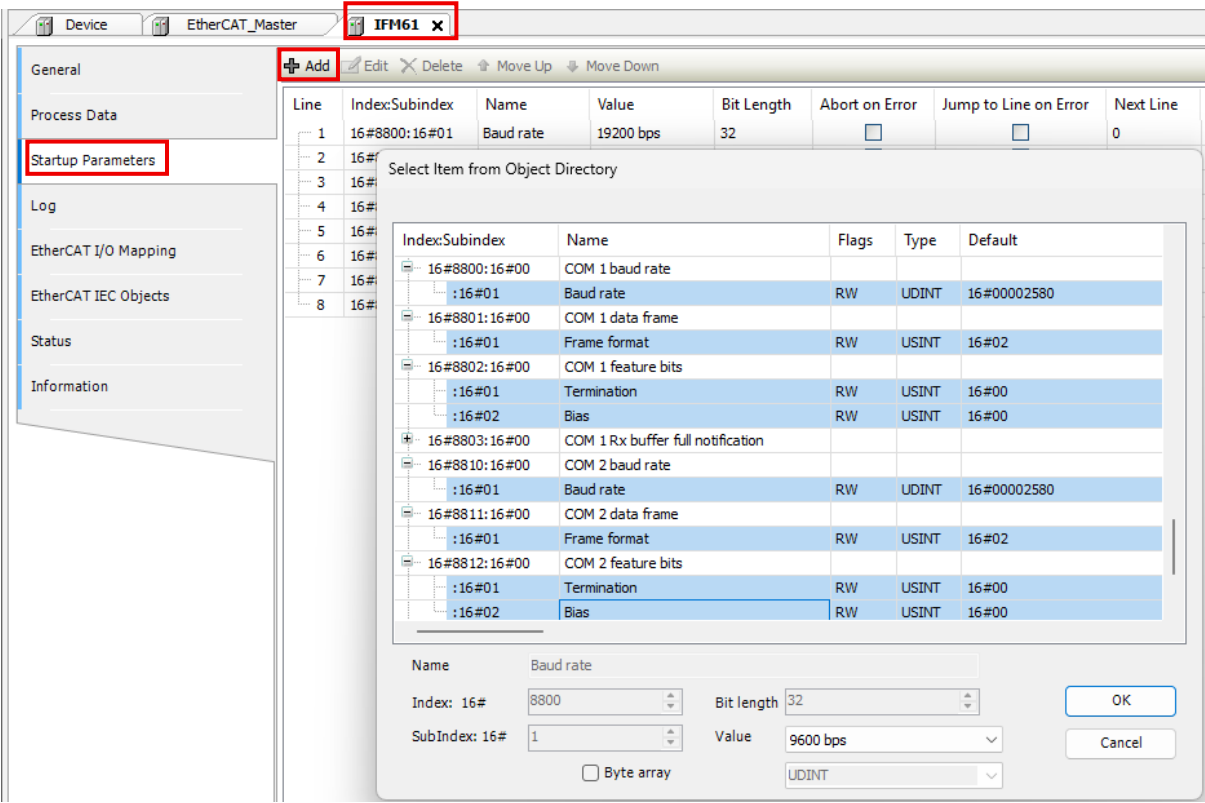
1. For the following procedure use the demo project or copy objects to your own project:
Demo3.5_IFM6.1_ModbusRTUClient.project.



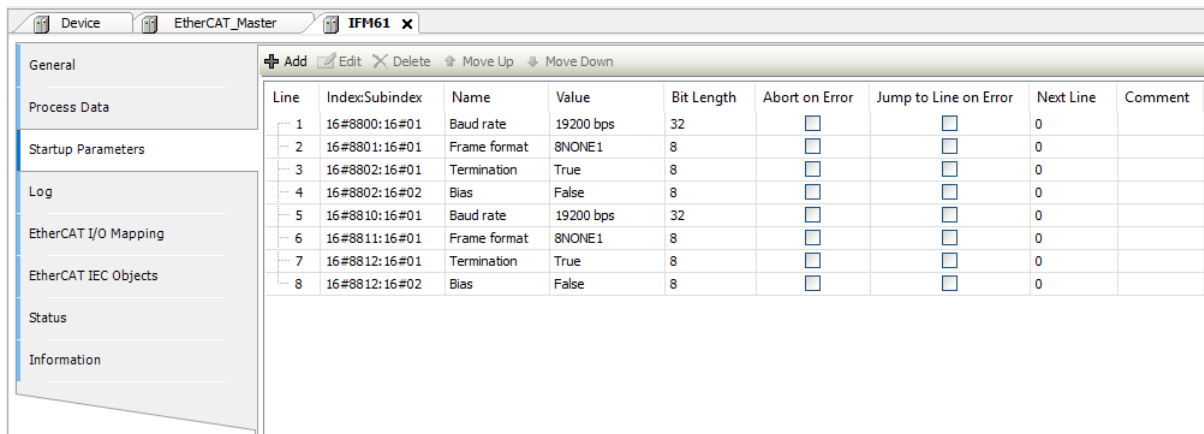
2. The IFM6.1 module is added under the **EtherCAT Master** in the device tree.



3. Configuration of COM ports are handled under **IFM6.1 > Startup Parameters** by adding the different settings.



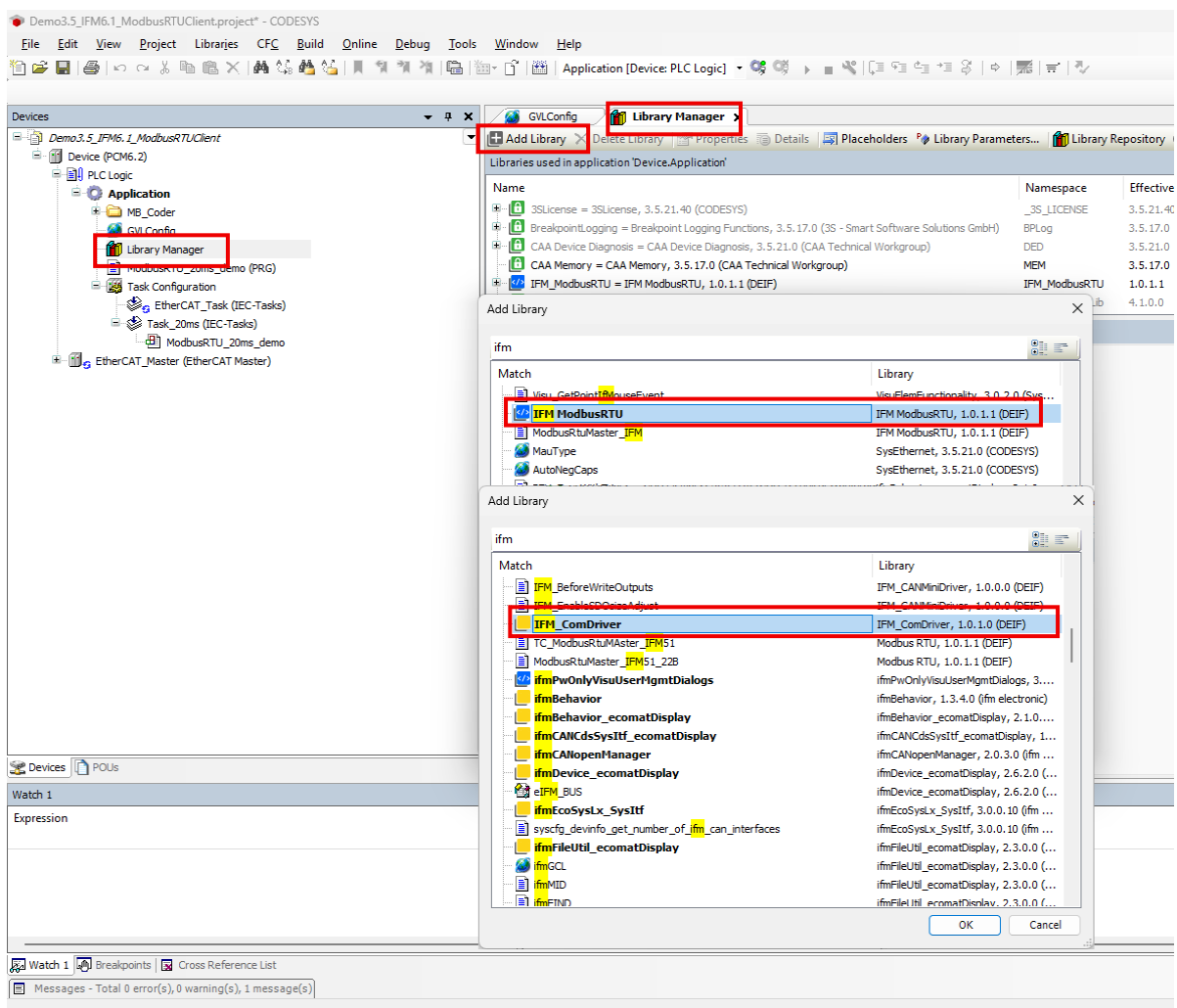
3.1 This is how it looks after adding and configuration.



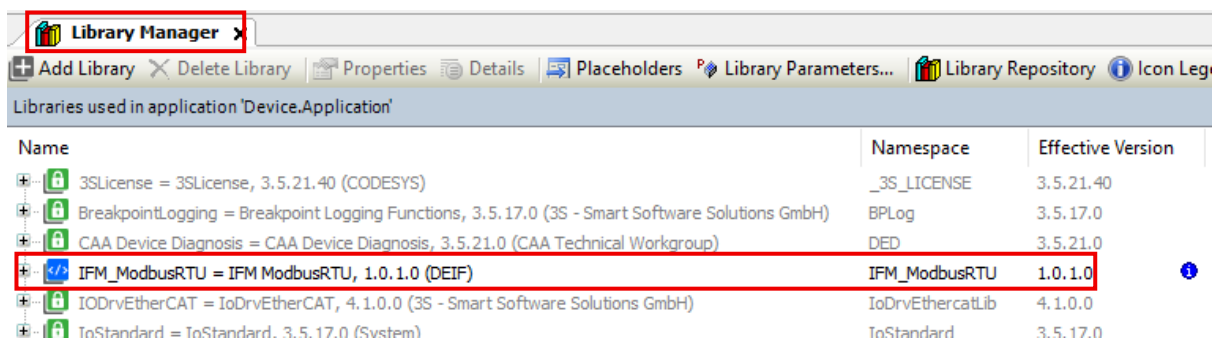
Line	Index/Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#8800:16#01	Baud rate	19200 bps	32	☐	☐	0	
2	16#8801:16#01	Frame format	8NONE1	8	☐	☐	0	
3	16#8802:16#01	Termination	True	8	☐	☐	0	
4	16#8802:16#02	Bias	False	8	☐	☐	0	
5	16#8810:16#01	Baud rate	19200 bps	32	☐	☐	0	
6	16#8811:16#01	Frame format	8NONE1	8	☐	☐	0	
7	16#8812:16#01	Termination	True	8	☐	☐	0	
8	16#8812:16#02	Bias	False	8	☐	☐	0	

Hardware is now ready for use. Moving on to implementing the software.

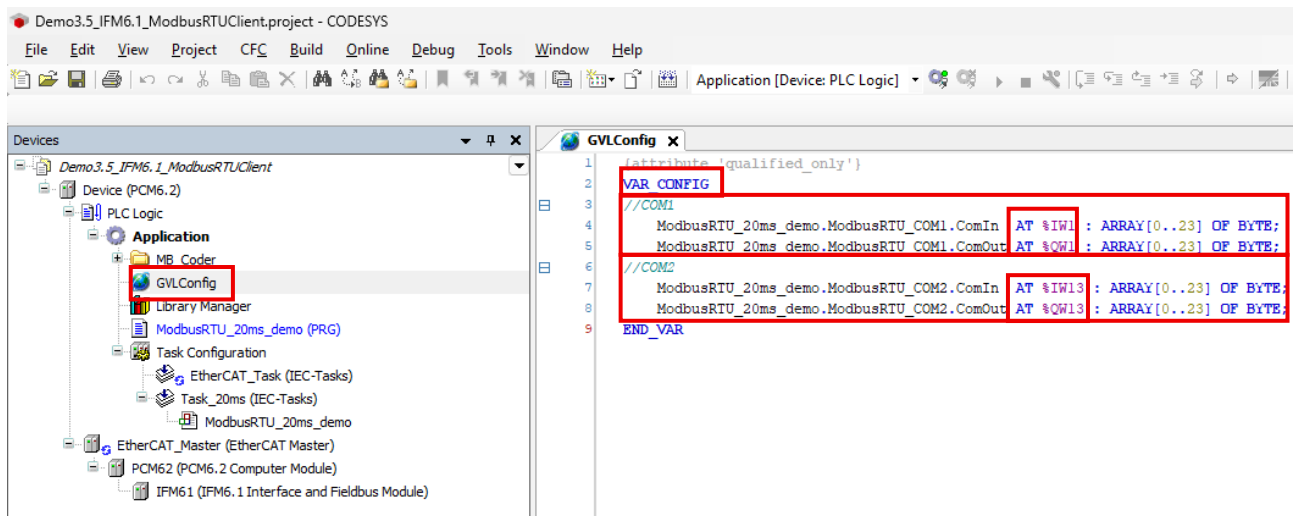
- First step is to ensure the libraries are included in the library repository - if not, install them: **IFM_ModbusRTU.compiled-library** & **IFM_ComDriver.compiled-library**. Add the libraries to the **Library Manager**.



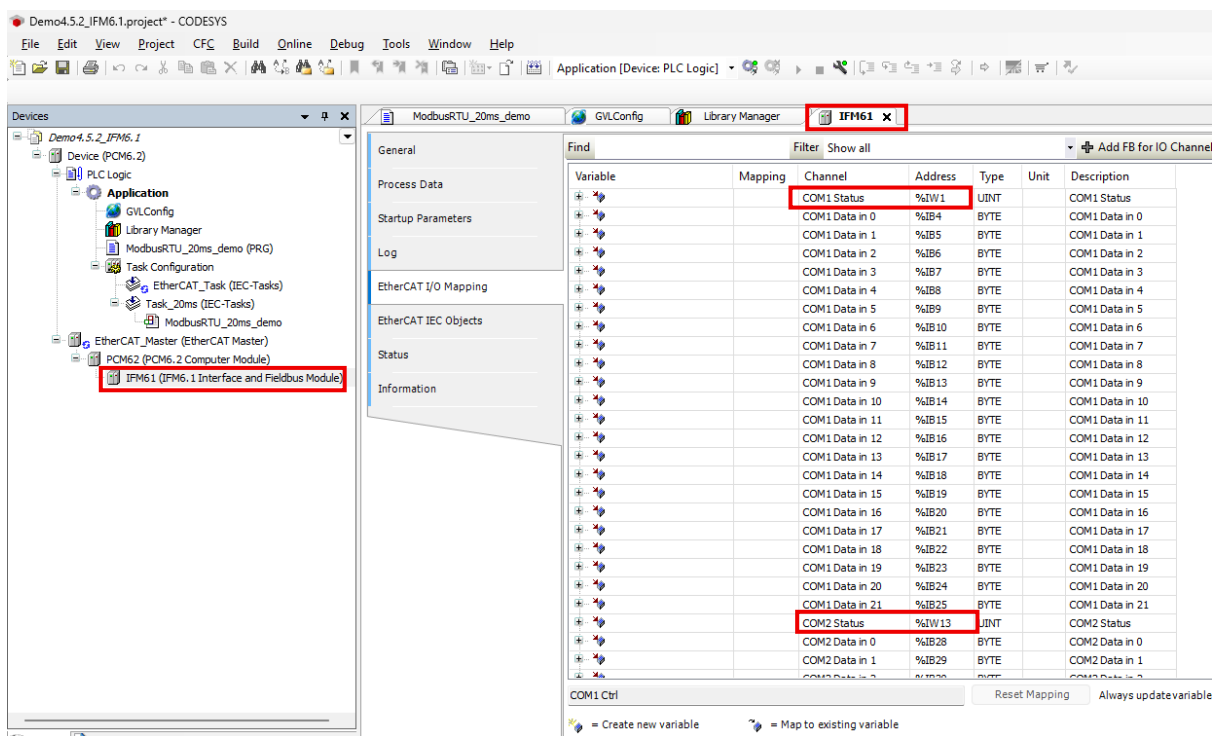
4.1 This how it looks after the library is added.

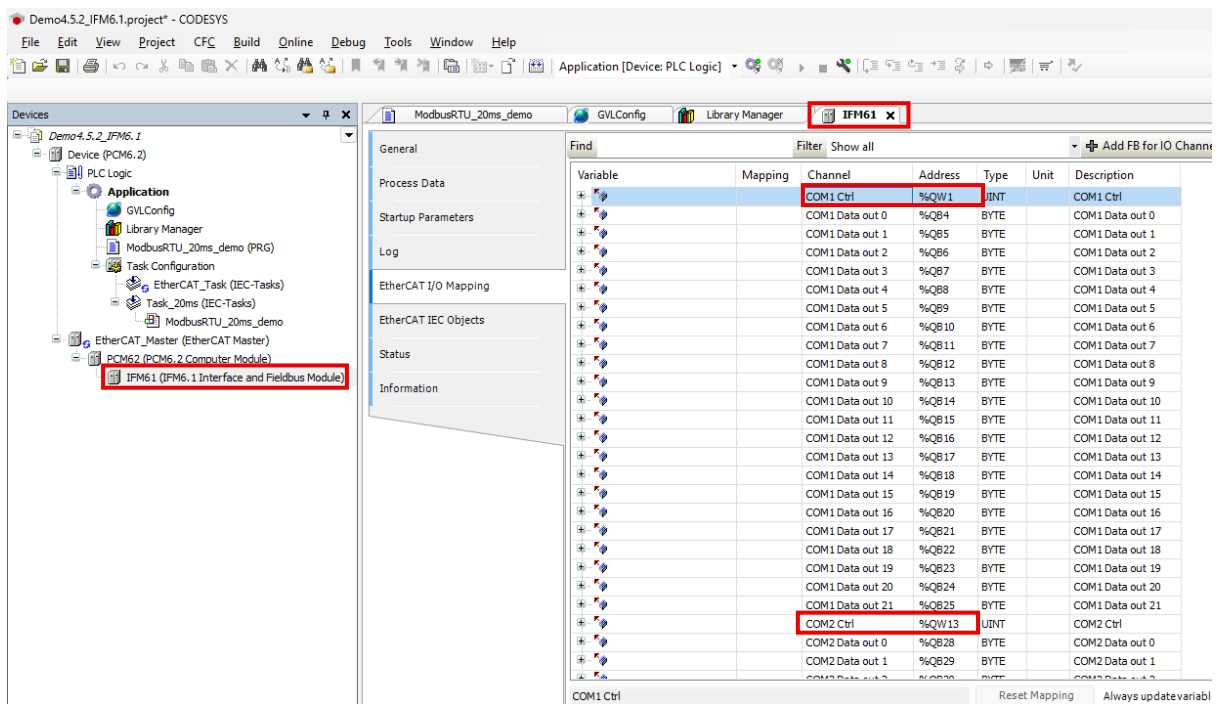


5. Navigate to or create **GVLConfig (VAR_CONFIG)** and point the addresses to the unique destination on the IFM.



6. Addresses are found in **EtherCAT I/O Mapping** under **IFM6.1 device**.





The project is now ready to use Modbus.

3.5.1.2 Test - Verify configuration

The project shows how to use **ReadHoldingRegisters** and **WriteMultipleRegister**. It will read and write F03/F10 address 0-122 and read F03 123-245.

To verify application connect a device (e.g. Modbus Slave) to COM1 with below settings:

Connection: RS-485 2-wire.

Baud rate: 19200 BPS.

Data bits: 8.

Parity: None.

Stop bits: 1.

Data on server:

Function code: F03 - Holding registers.

Start address: 0.

Quantity: 246.

To read the bytes in the right order the function blocks under **MB_Coder** can be used to map the data.

3.6 IFM 6.2 Program Configuration

3.6.1 SSI program configuration

IFM6.2 communication module is equipped with 2 SSI ports.

Associate variables with hardware channels in **EtherCAT I/O Mapping**.

- Click the device **IFM62**, select **Startup Parameters**, and click **Add** to create new startup parameters.
- In the pop-up dialog box, select and configure the required parameters.

Only one parameter can be added each time you click **Add**. To add multiple startup parameters, you need to click **Add** repeatedly.

Variable	Mapping	Channel	Address	Type	Unit	Description
		CAN 2 Rx overflow	%IX481.5	BIT		CAN 2 Rx overflow
		CAN 2 Warning limit reached	%IX481.6	BIT		CAN 2 Warning limit reached
		CAN 2 Bus-off	%IX481.7	BIT		CAN 2 Bus-off
		CAN 2 Rx error counter	%IB482	UDINT		CAN 2 Rx error counter
		CAN 2 Tx error counter	%IB483	UDINT		CAN 2 Tx error counter
		SSI 1 Data error	%IX484.0	BIT		SSI 1 Data error
		SSI 1 Frame error	%IX484.1	BIT		SSI 1 Frame error
		SSI 1 Data mismatch	%IX484.2	BIT		SSI 1 Data mismatch
		SSI 1 TxPDO State	%IX485.6	BIT		SSI 1 TxPDO State
Application_IFM62_SSI_1_CounterValue	SSI 1 Counter value	SSI 1 Counter value	%ID489	UDINT		SSI 1 Counter value
		SSI 2 Data error	%IX492.0	BIT		SSI 2 Data error
		SSI 2 Frame error	%IX492.1	BIT		SSI 2 Frame error
		SSI 2 Data mismatch	%IX492.2	BIT		SSI 2 Data mismatch
		SSI 2 TxPDO State	%IX493.6	BIT		SSI 2 TxPDO State
Application_IFM62_SSI_2_CounterValue	SSI 2 Counter value	SSI 2 Counter value	%ID494	UDINT		SSI 2 Counter value
		SSI 2 TxPDO Toggle	%IX493.7	BIT		SSI 2 TxPDO Toggle
		FREQ 1 State	%IX500.0	BIT		FREQ 1 State
		FREQ 1 TxPDO State	%IX501.6	BIT		FREQ 1 TxPDO State
		FREQ 1 TxPDO Toggle	%IX501.7	BIT		FREQ 1 TxPDO Toggle
		FREQ 1 Counter value	%ID126	UDINT		FREQ 1 Counter value
		FREQ 1 Period value	%ID127	UDINT		FREQ 1 Period value
		FREQ 2 State	%IX512.0	BIT		FREQ 2 State
		FREQ 2 TxPDO State	%IX513.6	BIT		FREQ 2 TxPDO State
		FREQ 2 TxPDO Toggle	%IX513.7	BIT		FREQ 2 TxPDO Toggle
		FREQ 2 Counter value	%ID129	UDINT		FREQ 2 Counter value
		FREQ 2 Period value	%ID130	UDINT		FREQ 2 Period value

- Three startup parameters need to be configured for the SSI port, namely **Frame size** and **Data length**.

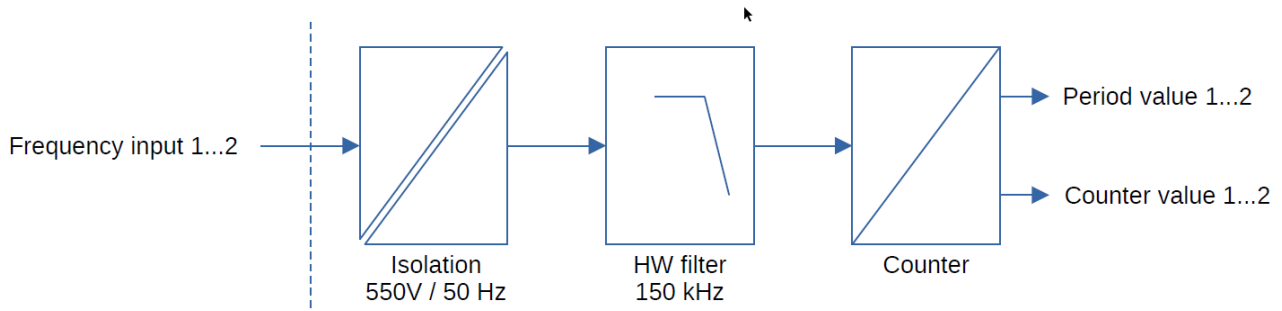
Taking the encoder Baumer GM400.Z103 (8192 × 4096, 25-bit Gray code) as an example, since this encoder adopts 25-bit Gray code, set the Frame size to 25, based on its specification of 8192 × 4096, set the Data length to 25.

Line	Index/Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#8020:16#06	Coding	Gray code	8	☐	☐	0	
2	16#8020:16#12	Data length	25	16	☐	☐	0	
3	16#8020:16#11	Frame size	25	16	☐	☐	0	
4	16#8030:16#06	Coding	Gray code	8	☐	☐	0	
5	16#8030:16#11	Frame size	25	16	☐	☐	0	
6	16#8030:16#12	Data length	25	16	☐	☐	0	

Messages - Total 0 error(s), 0 warning(s), 1 message(s)

3.6.2 FI program configuration

IFM6.2 has **two high-speed counters / frequency inputs** and allow measurement of period values for frequencies in the range from **1/42 Hz to 150 kHz**.



Each high-speed input channel contains four pieces of data

- **State** will be **1** when incoming pulse input is high.
- **TxPDO State** and **TxPDO Toggle** can be used to determine the validity of the received data.
- **TxPDO State** will be **1** if the counter value data is invalid.
- **TxPDO Toggle** will toggle between **0** and **1** each time the counter value data is updated.

"**Counter value**" will increase on rising edge for each incoming pulse. This can also be used to track if pulse input has stopped.

"**Period value**" will show period time between two pulses. The scaling of this value is 10 ns.

* Application.FrequencyInput.xInput	?	FREQ 1 State	%IX336.0	BIT	TRUE
* Application.FrequencyInput.xInput		FREQ 1 TxPDO State	%IX337.6	BIT	FALSE
* Application.FrequencyInput.xInput		FREQ 1 TxPDO Toggle	%IX337.7	BIT	FALSE
* Application.FrequencyInput.udIPulseCount	?	FREQ 1 Counter value	%ID85	UDINT	64292
* Application.FrequencyInput.udIPulsePeriod	?	FREQ 1 Period value	%ID86	UDINT	5207995
* Application.FrequencyInput.xInput		FREQ 2 State	%IX348.0	BIT	FALSE
* Application.FrequencyInput.xInput		FREQ 2 TxPDO State	%IX349.6	BIT	TRUE
* Application.FrequencyInput.xInput		FREQ 2 TxPDO Toggle	%IX349.7	BIT	FALSE
* Application.FrequencyInput.udIPulseCount		FREQ 2 Counter value	%ID88	UDINT	0
* Application.FrequencyInput.udIPulsePeriod		FREQ 2 Period value	%ID89	UDINT	0

Device.Application.FrequencyInput					
Expression	Type	Value	Prepared value	Address	Comment
xInput	BOOL	TRUE			
udiPulseCount	UDINT	59982			
udiPulsePeriod	UDINT	5196956			
rFrequency	LREAL	19.242033220985515			


```

1 //1s = 1000000000ns, scale = 10ns.
2 //      |-----|      |-----|      |-----|      |-----|      1.
3 //      |-----|      |-----|      |-----|      |-----|      0.
4 //      |---25ms---|---25ms---|---25ms---|---25ms---|      .
5 rFrequency := 1000000000 / (UDINT_TO_LREAL(udiPulsePeriod) * 10);
6 .
7 .
8 .
  
```

3.7 SIM6.4 program configuration

The SIM6.4 module allows **EtherCAT redundancy** for the interconnection of multiple racks in a system via electrical connections. The Network Interface (NIC) is used for the EtherCAT master.

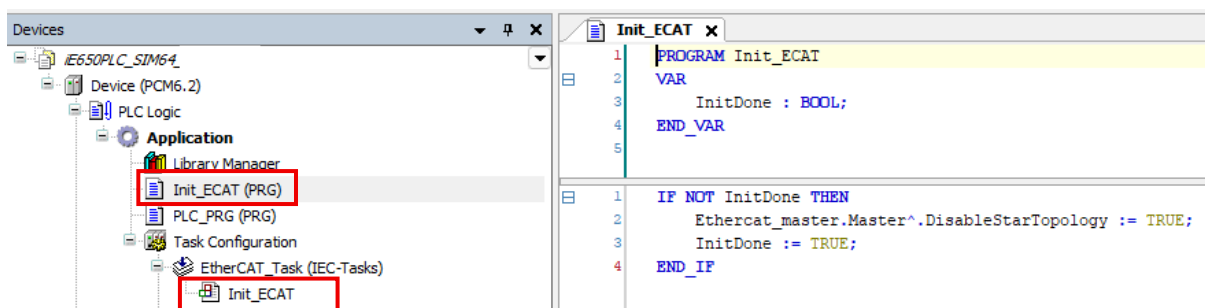
The module has two EtherCAT OUT ports (RJ45) - Port 3 and Port 1.

Main function of the module is to enable redundancy, ring connection, for remote units, but it does also support connection of one or two individual EtherCAT circuits.

3.7.1 EtherCAT redundancy

For the EtherCAT redundancy to work there needs to be added a small piece of code, that needs to be **initialized startup**.

Init_ECAT (PRG) can be called under the **EtherCAT_Task**.



```
PROGRAM Init_ECAT
```

```
VAR
```

```
    InitDone : BOOL;
```

```
END_VAR
```

```
IF NOT InitDone THEN
```

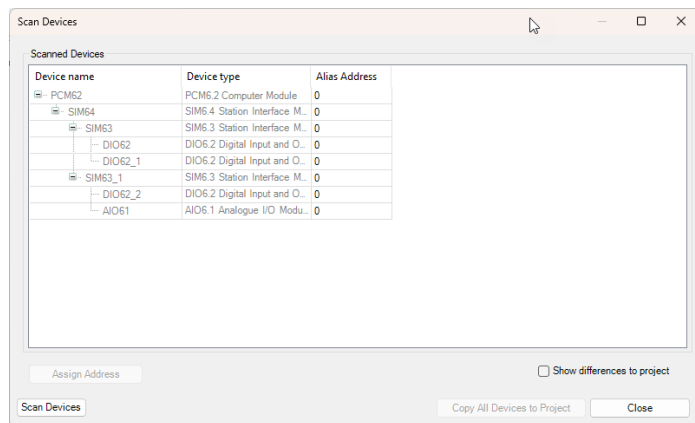
```
    Ethercat_master.Master^.DisableStarTopology := TRUE;
```

```
    InitDone := TRUE;
```

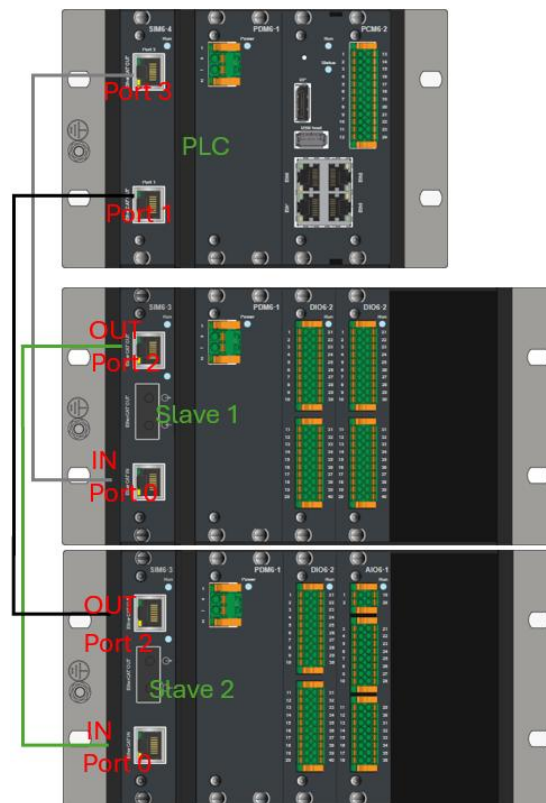
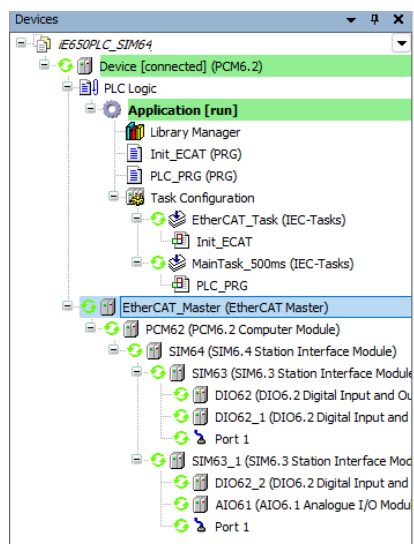
```
END_IF
```

Physical connections can be configured as the picture on the right.

When scanning for devices, the result will look like the example below.



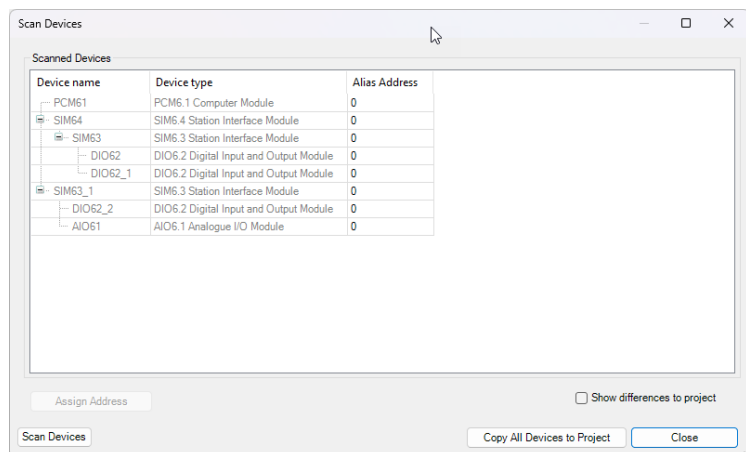
Device tree will look like the image below.



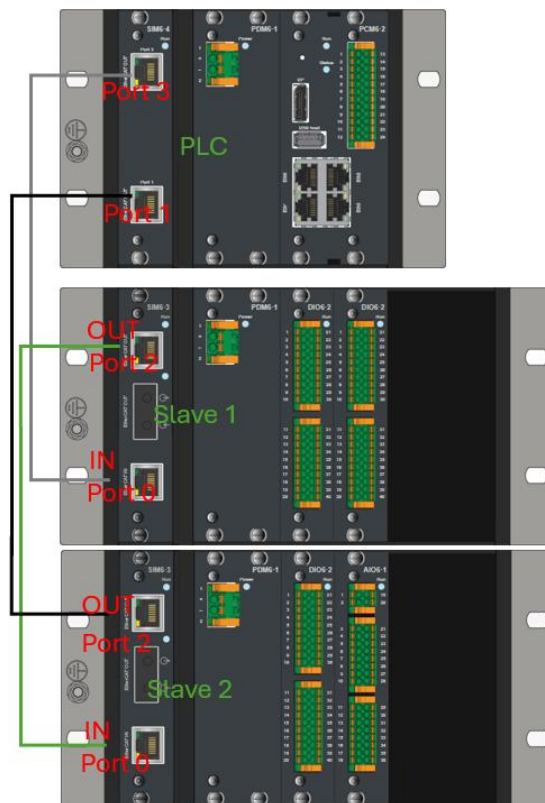
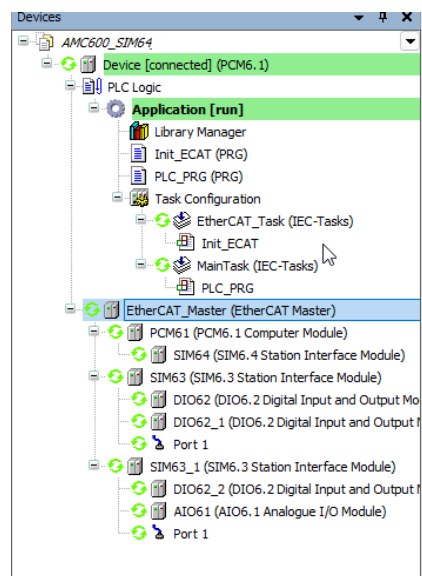
3.7.2 Two individual EtherCAT circuits

Physical connections can be configured as the picture on the right.

When scanning for devices, the result will look like the example below.



Device tree will look like the image below.



3.8 SIM6.5 program configuration

The SIM6.5 module allows **EtherCAT redundancy** for the interconnection of multiple racks in a system via fiber optic connections. The Network Interface (NIC) is used for the EtherCAT master.

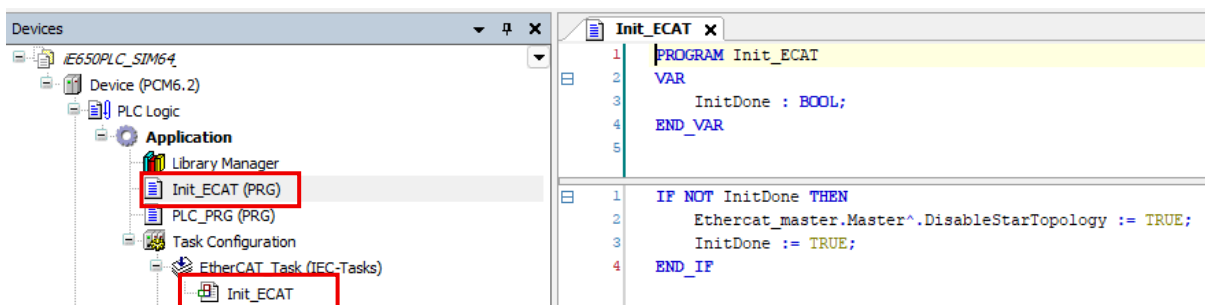
The module has two EtherCAT OUT ports (optical) - Port 2 and Port 1.

Main function of the module is to enable redundancy, ring connection, for remote units, but it does also support connection of two individual EtherCAT circuits.

3.8.1 EtherCAT redundancy

For the EtherCAT redundancy to work a small piece of code needs to be **initialized at startup**.

Init_ECAT (PRG) can be called under the **EtherCAT_Task**.



```
PROGRAM Init_ECAT
```

```
VAR
```

```
    InitDone : BOOL;
```

```
END_VAR
```

```
IF NOT InitDone THEN
```

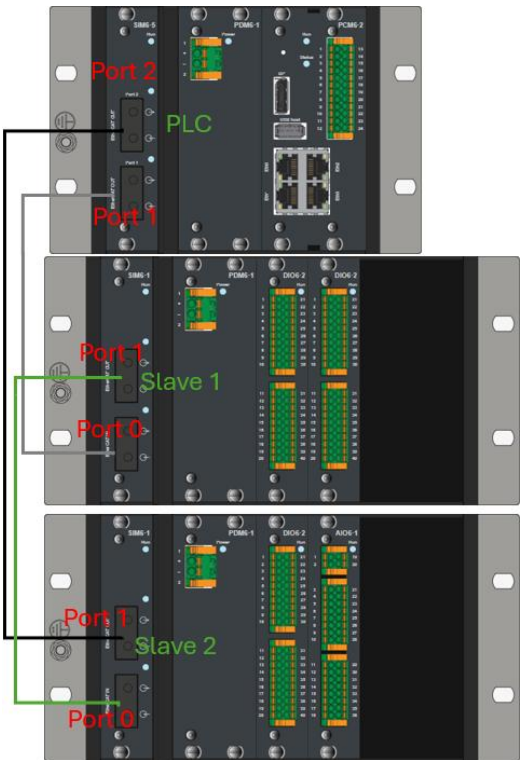
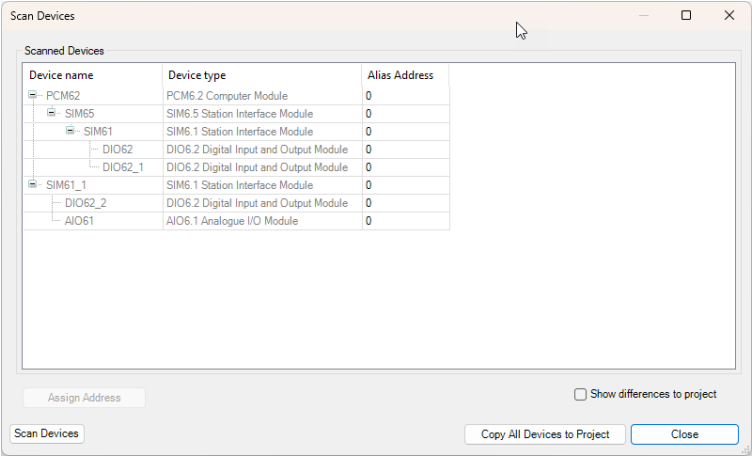
```
    Ethercat_master.Master^.DisableStarTopology := TRUE;
```

```
    InitDone := TRUE;
```

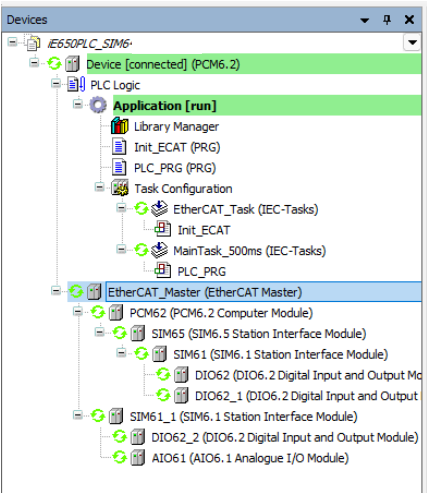
```
END_IF
```

Physical connections can be configured as the picture on the right.

When scanning for devices, the result will look like the example below.



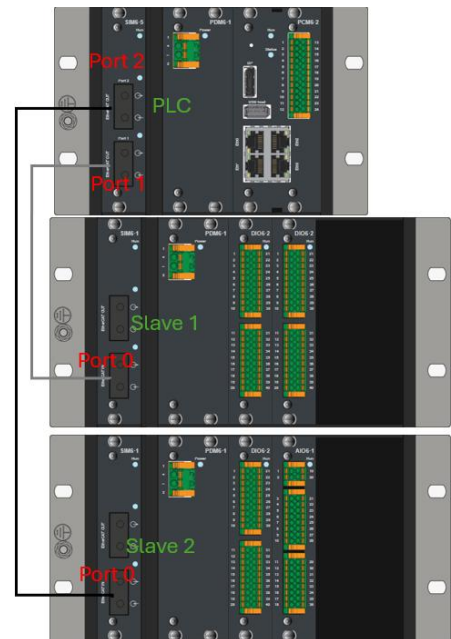
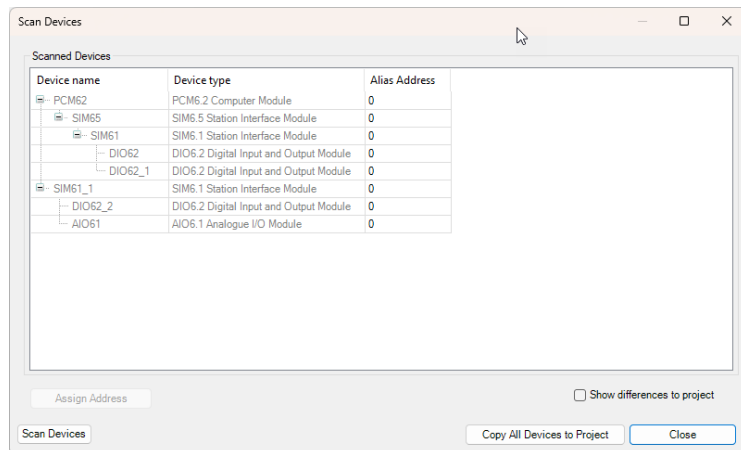
Device tree will look like the image below.



3.8.2 Two individual EtherCAT circuits

Physical connections can be configured as the picture on the right.

When scanning for devices, the result will look like the example below.



Device tree will look like the image below.

