U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Master in Electrical and Computer Engineering

Using Codesys

and

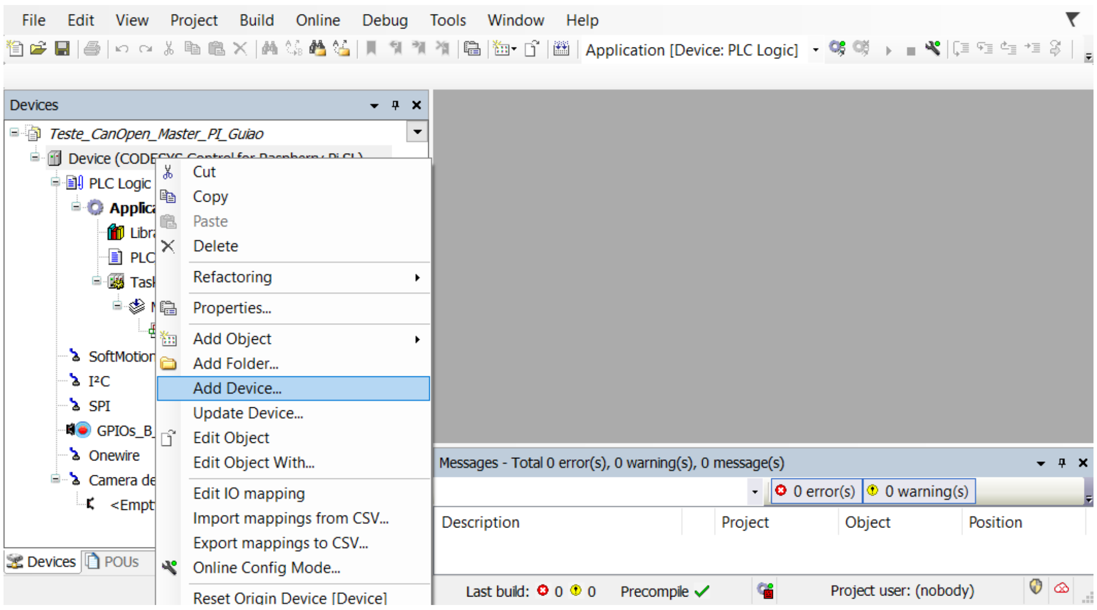
CANopen

Version 1.2 (2021-12-02)

1. Configuring a CANopen Network

We will begin by configuring the CANopen Master that will communicate with the CANopen Slave (running on an STB device).

The first step is to create a **CANbus** network. Right-click on the '*Device'* and choose '*Add Device*'.



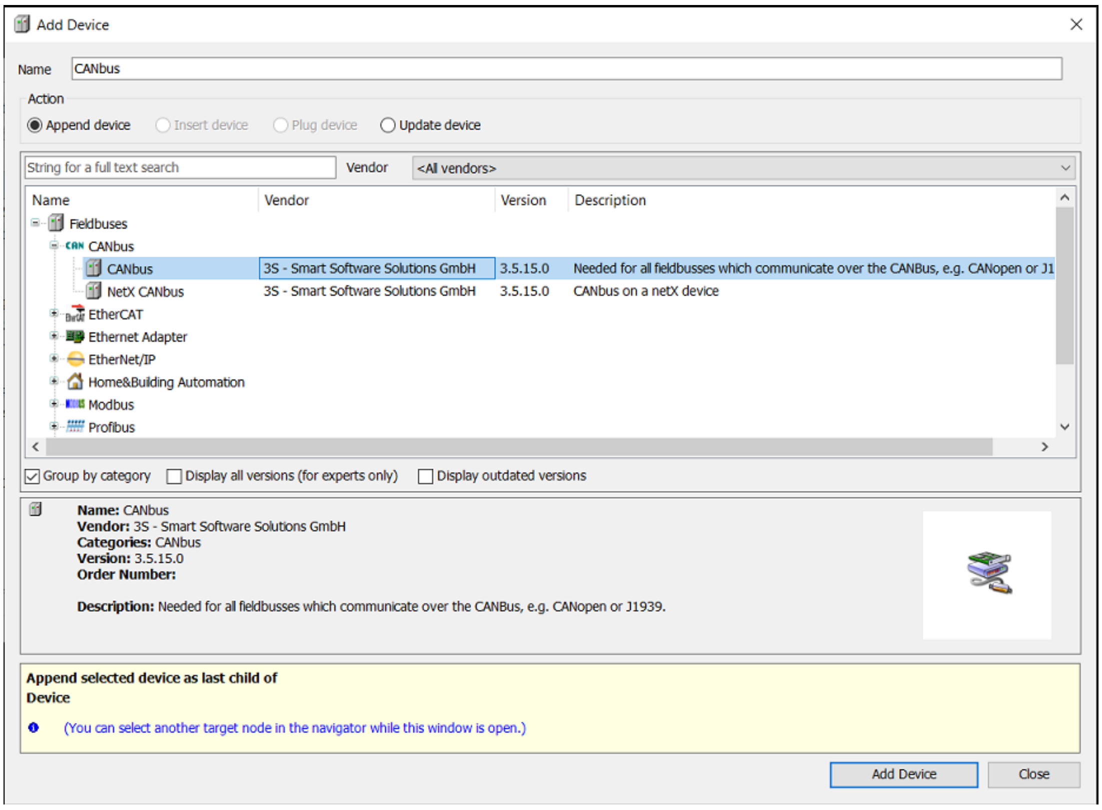
Messages - Total 0 error(s), 0 warning(s), 0 message(s)

0 error(s) 0 warning(s)

Description	Project	Object	Position

Last build: 0 0 Precompile ✓ Project user: (nobody)

You will be shown a window where you may choose the device to add. Start by adding a '**CANbus'** device.


Add Device

Name: CANbus

Action: [✓ Append device] Insert device Plug device Update device

String for a full text search Vendor: <All vendors>

Name	Vendor	Version	Description
Fieldbuses			
CANbus			
CANbus	3S - Smart Software Solutions GmbH	3.5.15.0	Needed for all fieldbusses which communicate over the CANBus, e.g. CANopen or J1
NetX CANbus	3S - Smart Software Solutions GmbH	3.5.15.0	CANbus on a netX device
EtherCAT			
Ethernet Adapter			
EtherNet/IP			
Home&Building Automation			
Modbus			
Profibus			

[✓ Group by category] Display all versions (for experts only) Display outdated versions

Name: CANbus
Vendor: 3S - Smart Software Solutions GmbH
Categories: CANbus
Version: 3.5.15.0
Order Number:

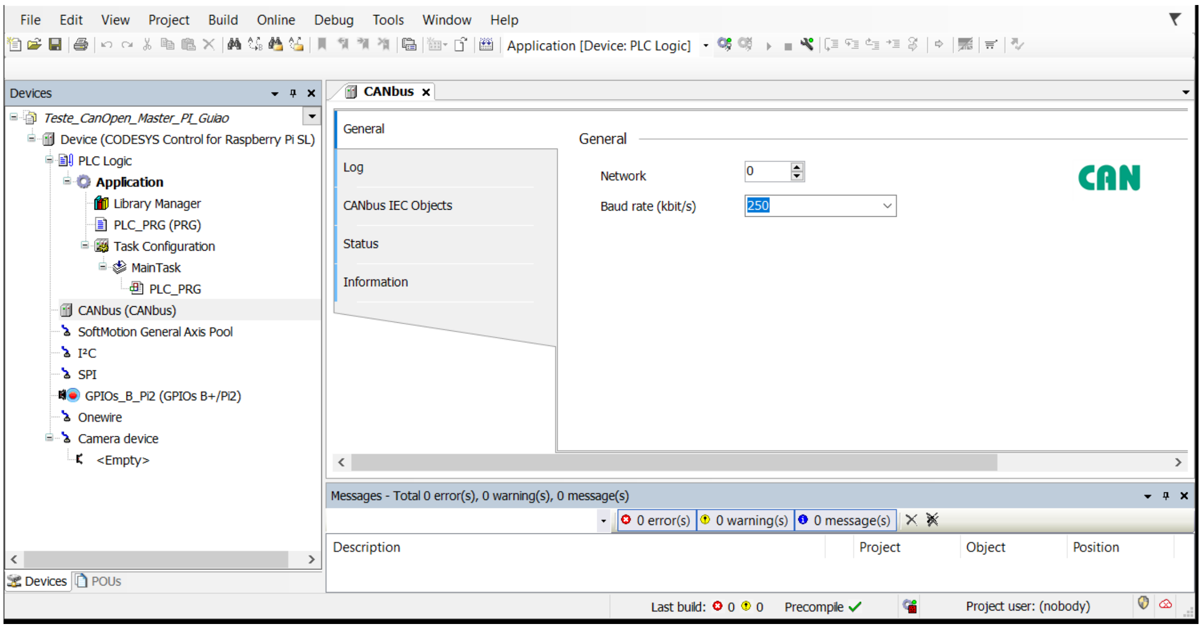
Description: Needed for all fieldbusses which communicate over the CANBus, e.g. CANopen or J1939.

Append selected device as last child of
Device

(You can select another target node in the navigator while this window is open.)

Add Device Close

On the Project browser ('*Devices*') window on the left, double-click on the newly created **CANbus** network**.**



Teste_CanOpen_Master_PI_Guiao
Device (CODESYS Control for Raspberry Pi SL)
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
Task Configuration
MainTask
PLC_PRG
CANbus (CANbus)
SoftMotion General Axis Pool
I²C
SPI
GPIOs_B_Pi2 (GPIOs B+/Pi2)
Onewire
Camera device
<Empty>

General | Log | CANbus IEC Objects | Status | Information

General
Network: 0
Baud rate (kbit/s): 250

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

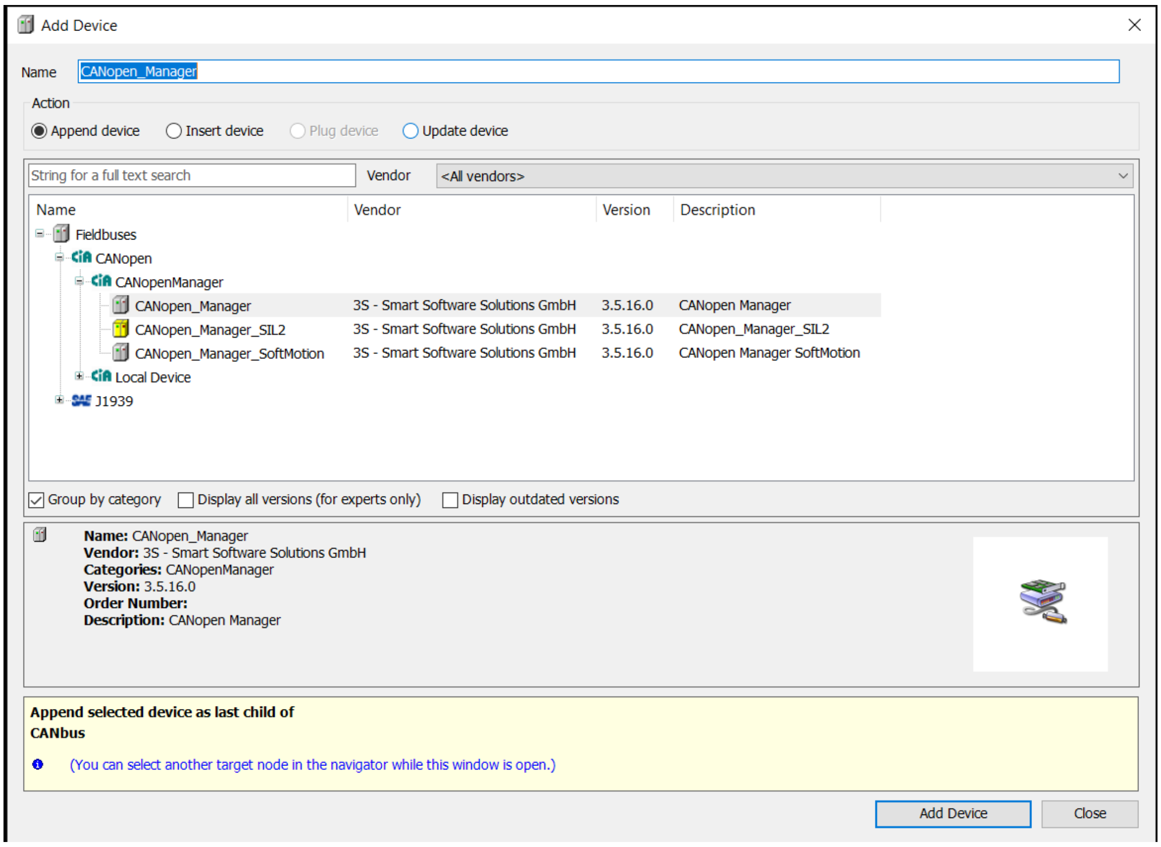
Description	Project	Object	Position

Last build: 0 0 Precompile ✓ Project user: (nobody)

In this window, you can configure the parameters of the CAN network. Use the following configuration:

- Network: 0 – this corresponds to the **can0** interface this will be used to capture CAN/CANopen packets with Wireshark
- Baud Rate: 250 Kbits/s (network transmission rate)

The next step is to add a **CANopen Master.** Once again, select the created **CANbus** in the ‘*Devices’* list and add a ‘**CANopen Manager**’ device.



Name: CANopen_Manager

Action
[✓ Append device] Insert device Plug device Update device

String for a full text search | Vendor: <All vendors>

Name	Vendor	Version	Description
Fieldbuses			
CANopen			
CANopenManager			
CANopen_Manager	3S - Smart Software Solutions GmbH	3.5.16.0	CANopen Manager
CANopen_Manager_SIL2	3S - Smart Software Solutions GmbH	3.5.16.0	CANopen_Manager_SIL2
CANopen_Manager_SoftMotion	3S - Smart Software Solutions GmbH	3.5.16.0	CANopen Manager SoftMotion
Local Device			
J1939			

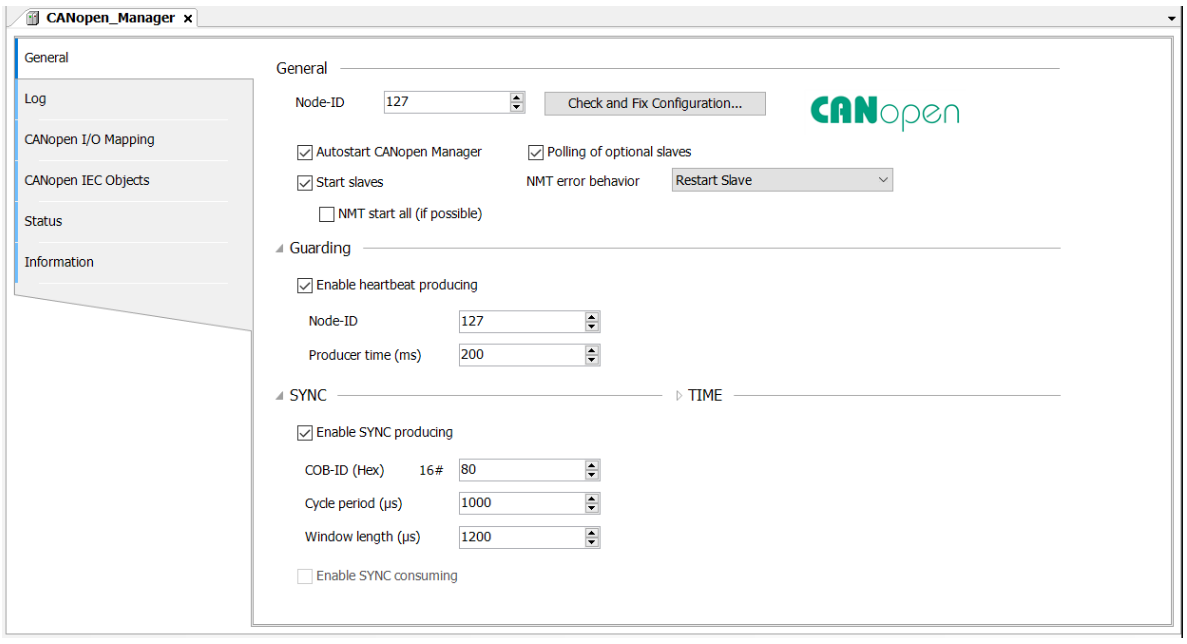
[✓] Group by category [ ] Display all versions (for experts only) [ ] Display outdated versions

Name: CANopen_Manager
Vendor: 3S - Smart Software Solutions GmbH
Categories: CANopenManager
Version: 3.5.16.0
Order Number:
Description: CANopen Manager

Append selected device as last child of
CANbus

(You can select another target node in the navigator while this window is open.)

On the Project browser (‘*Devices*’) window on the left, double-click on the newly created ‘**CANbus Manager’**.



General

Node-ID: 127

☑ Autostart CANopen Manager
☑ Polling of optional slaves
☑ Start slaves
NMT error behavior: Restart Slave
☐ NMT start all (if possible)

Guarding

☑ Enable heartbeat producing
Node-ID: 127
Producer time (ms): 200

SYNC

TIME

☑ Enable SYNC producing
COB-ID (Hex) 16#: 80
Cycle period (µs): 1000
Window length (µs): 1200
☐ Enable SYNC consuming

In this window, you can configure the parameters of the **CANopen Master**. By default, some parameters are already selected. For now, do not change these values. Select the ‘**SYNC’** checkbox (if unchecked).

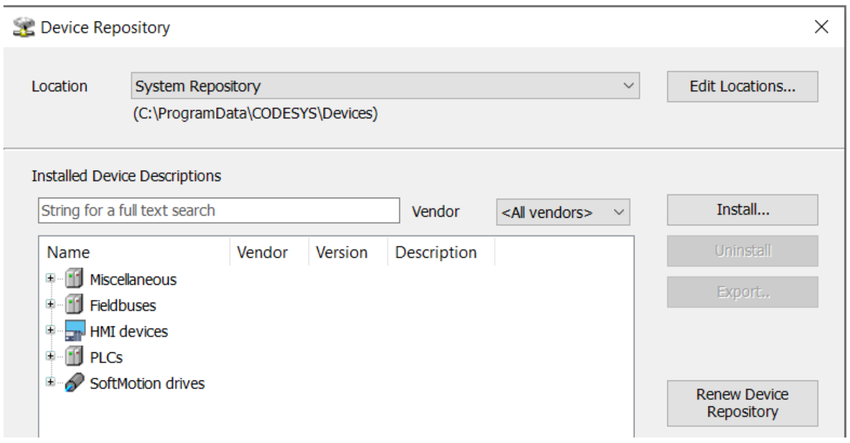
The meaning of these parameters is as follows:

- *Node-ID* (127) – the Master address
- *Autostart CANopen Manager* – must be checked to start the Master automatically
- *Start Slaves* – must be checked to start the Slaves automatically (the Master does this task)
- *Guarding* – defines which node sends ‘Heartbeat messages’ and their period. By default, it is the master (127) with a 200ms period.
- *SYNC* – enables the Master to send SYNC messages. When selected, the default configuration is the following:
  - COB-ID (80) – this is the CAN identifier for SYNC messages
  - Cycle period (1000us) – the period of SYNC messages
  - Window length (1200us) – the window length associated to SYNC messages

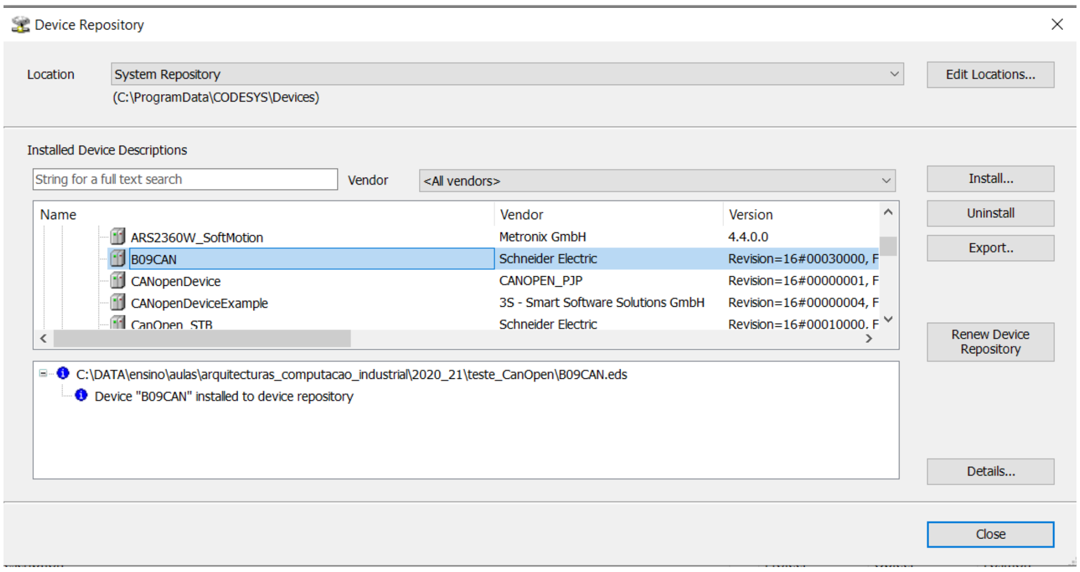
Confirm that the selected options and values are according to the previous values (check the figure).

The next step is to add a **CANopen Slave** to the Master’s configuration. For that, it is necessary to insert the slave's EDS file into the Codesys device repository (a database). Proceed as follows:

1. Download the EDS file of the STB device that you are using in your workbench. You can find this file on this link (ftp://labs-automacao.fe.up.pt/I005/aulas/ACI/EDS/).
   (NOTE: The URL uses the ftp protocol. Newer browsers no long support this protocol. To open the URL you may need to use another application, such as ‘filemanager’ in Windows.)
2. In the *Tools* menu select *Device Repository*. It opens the following window:



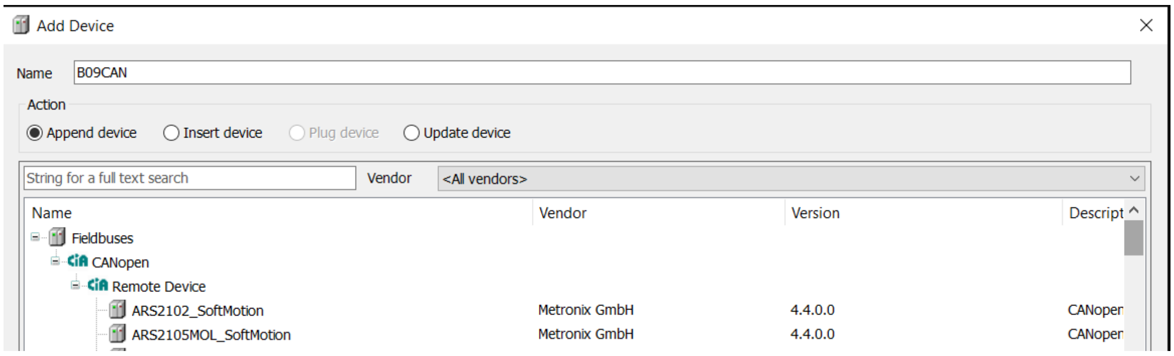
3. Click on the “Install...” button
4. On the new window select the type of file as ‘*EDS and DFC files*’ (combo box at bottom right).
5. Find the EDS file that you have downloaded and selected it. Click on ‘*Open’*.
6. The Slave configuration is now included in the repository. A window should appear like the following one (the name of the device should be different).



Name	Vendor	Version
ARS2360W_SoftMotion	Metronix GmbH	4.4.0.0
B09CAN	Schneider Electric	Revision=16#00030000, F
CANopenDevice	CANOPEN_PJP	Revision=16#00000001, F
CANopenDeviceExample	3S - Smart Software Solutions GmbH	Revision=16#00000004, F
CanOpen_STB	Schneider Electric	Revision=16#00010000, F

```
C:\DATA\ensino\aulas\arquitecturas_computacao_industrial\2020_21\teste_CanOpen\B09CAN.eds
    Device "B09CAN" installed to device repository
```

Now, select the **CANopen Manager** in the ‘*Devices’* list and add the device that you included in the repository. The device that you inserted can be found on the tree-like structure: ‘*Fieldbuses → CANopen → Remote Device’*

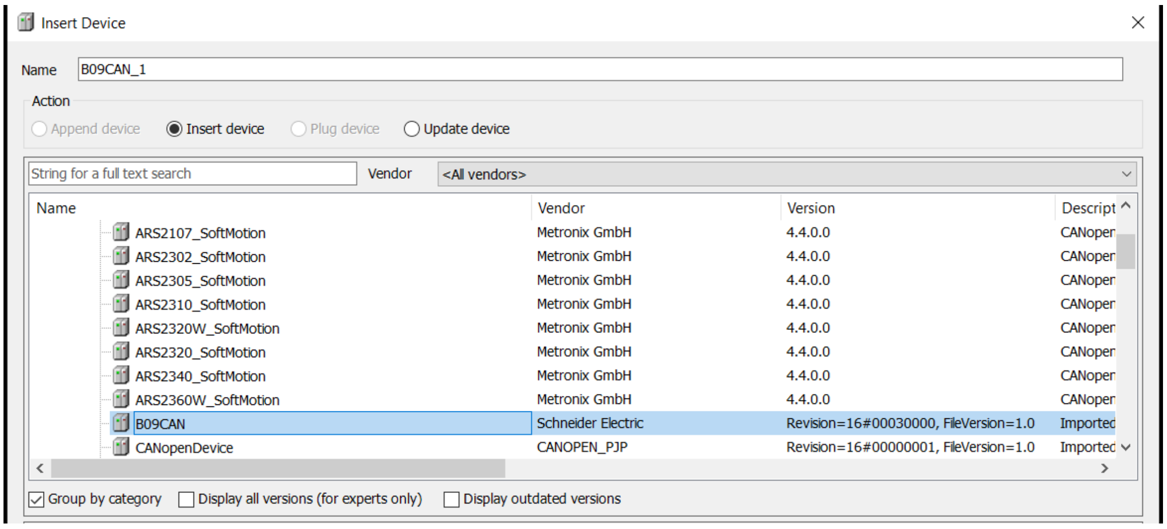


Name: B09CAN

Action: [✓ Append device] Insert device Plug device Update device

Name	Vendor	Version	Descript
Fieldbuses			
CANopen			
Remote Device			
ARS2102_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2105MOL_SoftMotion	Metronix GmbH	4.4.0.0	CANoper

Select the device and click on '*Add Device*'


Insert Device

Name: B09CAN_1

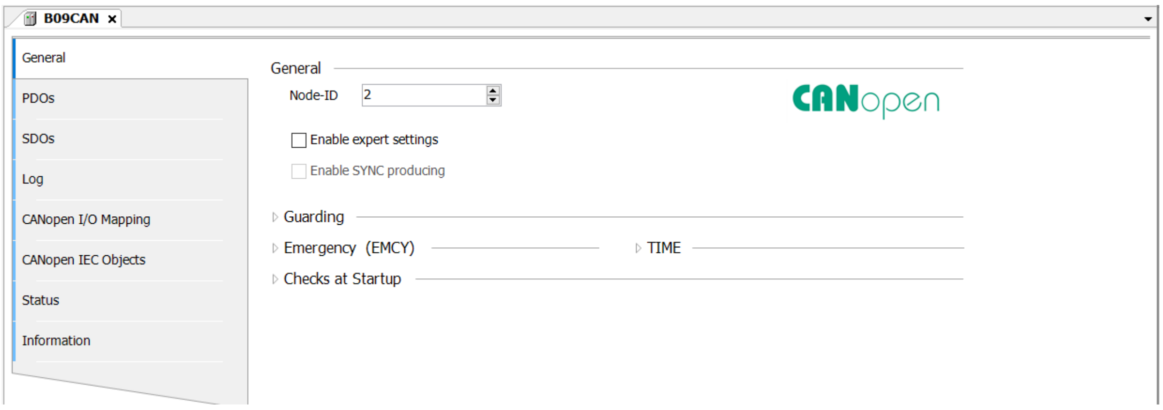
Action: ( ) Append device [● Insert device] ( ) Plug device ( ) Update device

String for a full text search Vendor <All vendors>

Name	Vendor	Version	Descript
ARS2107_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2302_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2305_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2310_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2320W_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2320_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2340_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
ARS2360W_SoftMotion	Metronix GmbH	4.4.0.0	CANoper
B09CAN	Schneider Electric	Revision=16#00030000, FileVersion=1.0	Importec
CANopenDevice	CANOPEN_PJP	Revision=16#00000001, FileVersion=1.0	Importec

[✓ Group by category] ☐ Display all versions (for experts only) ☐ Display outdated versions

Now, you should have a **CAN network** containing a **CANopen Master**, which in turns contains a **CANopen Slave**. By double-clicking on the **CANopen** slave, its configuration window will be shown on the right.


B09CAN

General / PDOs / SDOs / Log / CANopen I/O Mapping / CANopen IEC Objects / Status / Information

General

Node-ID: 2

☐ Enable expert settings

☐ Enable SYNC producing

Guarding

Emergency (EMCY)

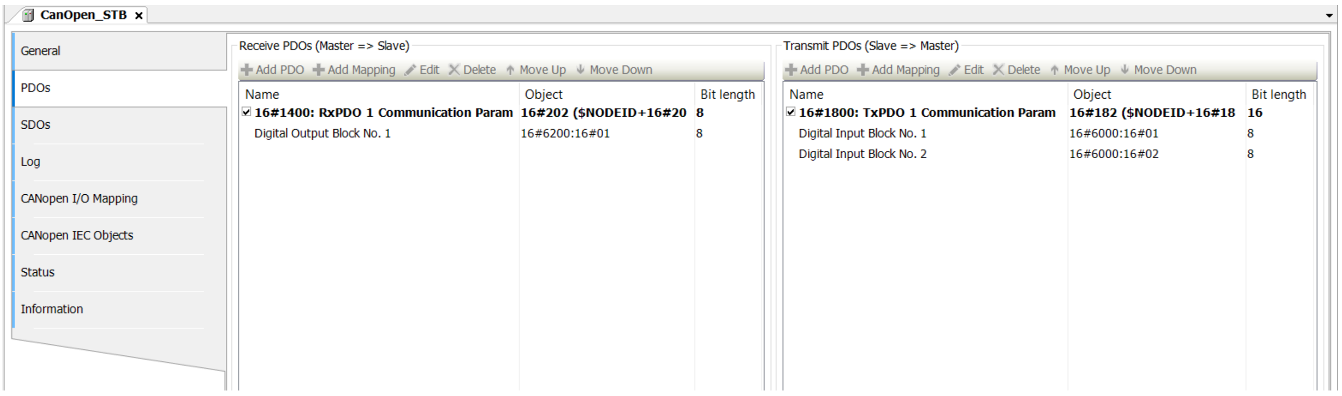
TIME

Checks at Startup

This window has several parameters. <u>For now, is not necessary to modify them</u>. Confirm only that the Slave address (Node-ID) is **2**.

The next step is to analyze the PDOs (Process Data Objects) that are exchanged between the Master and the Slave. For that select the '**PDOs**' tab on the Slave device.

Note: since the slaves used in this assignment are not identical between workbenches, it may be possible that the PDO mapping is slightly different from that shown in the document.


CanOpen_STB

General / PDOs / SDOs / Log / CANopen I/O Mapping / CANopen IEC Objects / Status / Information

Receive PDOs (Master => Slave)

Name	Object	Bit length
✓ 16#1400: RxPDO 1 Communication Param	16#202 ($NODEID+16#20	8
Digital Output Block No. 1	16#6200:16#01	8

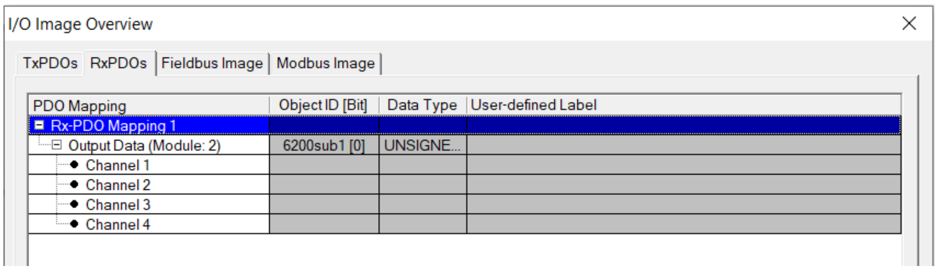
Transmit PDOs (Slave => Master)

Name	Object	Bit length
✓ 16#1800: TxPDO 1 Communication Param	16#182 ($NODEID+16#18	16
Digital Input Block No. 1	16#6000:16#01	8
Digital Input Block No. 2	16#6000:16#02	8

In this configuration, the PDOs are the following:

- one PDO transmitted by the Master to the Slave (**RxPDO 1**, left on the figure). This PDO represents data that is produced by the Master and is consumed by the Slave. The PDO has a single block (8 bits) that represents the Digital Outputs of the slave (Digital Output Block No.1).
- one PDO transmitted by the Slave to the Master (**TxPDO 1**, right on the figure). This PDO represents data that is produced by the Slave and is consumed by the Master. The PDO has 2 blocks (each 8 bits long) that represent the Digital Inputs and the Slave Diagnostics (Digital Input Blocks No.1 and 2, respectively).

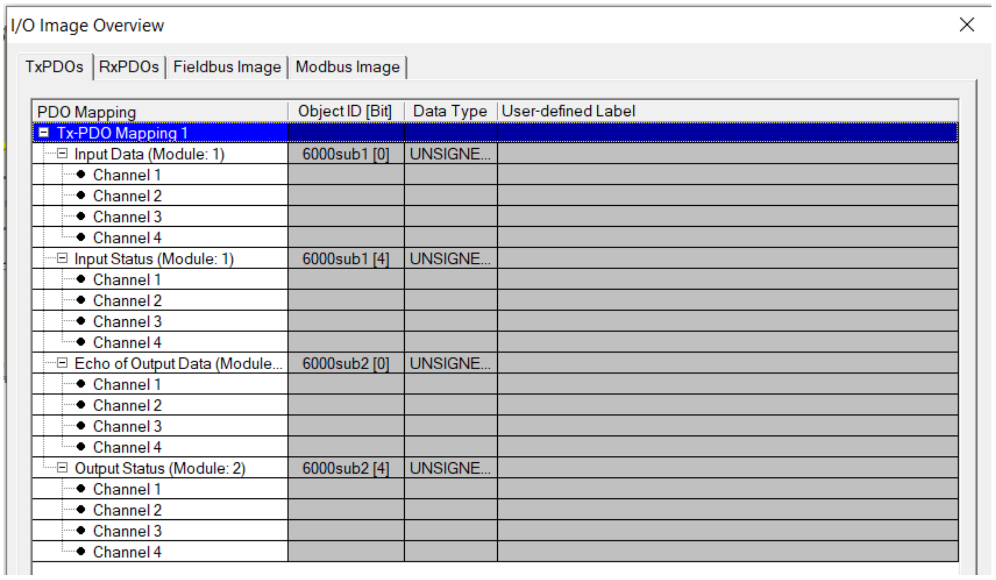
The **Advantys** tool can be used to inspect how these PDO are mapped. With this tool open the STB configuration and select the following menu: '*Island'* → *'IO Image Overview*'


I/O Image Overview

TxPDOs | RxPDOs | Fieldbus Image | Modbus Image

PDO Mapping	Object ID [Bit]	Data Type	User-defined Label
Rx-PDO Mapping 1			
Output Data (Module: 2)	6200sub1 [0]	UNSIGNE...	
Channel 1			
Channel 2			
Channel 3			
Channel 4			

The **RxPDO** has 1 bit allocated for each Digital Output (4 bits total). The remaining 4 bits (most significant) are unused.


I/O Image Overview

TxPDOs | RxPDOs | Fieldbus Image | Modbus Image

PDO Mapping	Object ID [Bit]	Data Type	User-defined Label
Tx-PDO Mapping 1			
Input Data (Module: 1)	6000sub1 [0]	UNSIGNE...	
Channel 1			
Channel 2			
Channel 3			
Channel 4			
Input Status (Module: 1)	6000sub1 [4]	UNSIGNE...	
Channel 1			
Channel 2			
Channel 3			
Channel 4			
Echo of Output Data (Module...	6000sub2 [0]	UNSIGNE...	
Channel 1			
Channel 2			
Channel 3			
Channel 4			
Output Status (Module: 2)	6000sub2 [4]	UNSIGNE...	
Channel 1			
Channel 2			
Channel 3			
Channel 4			

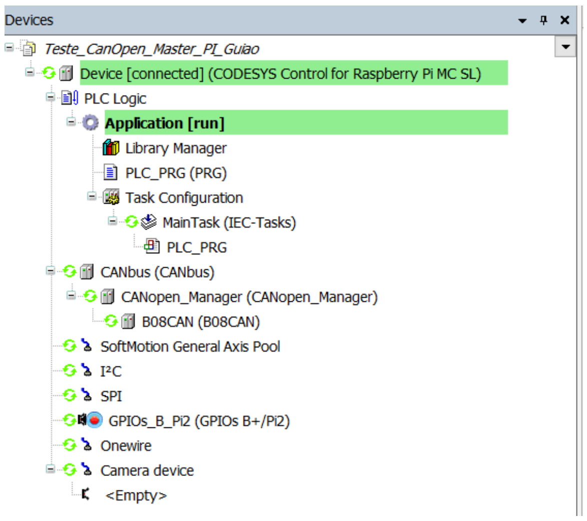
The **TxPDO** has the following mapping:

- 1 bit for each Digital Input (4 bits)
- 1 bit for the diagnostic status of each Digital Input (4 bits)
- 1 bit for each Digital Output (a copy of the outputs) (4 bits)
- 1 bit for the diagnostic status of each Digital Output (4 bits)

The configuration is now complete. Compile and download this configuration to the Raspberry PI.

Go '*Online*' and make RUN.

If everything is correct the network will be working as expected (you must wait a few seconds for the network configuration to take effect). You should observe that **CANbus**, **CANopen Manager** and **CANopen Device** are highlighted in green (otherwise they aren't working…in this case review the previous procedures). The next figure presents an example:



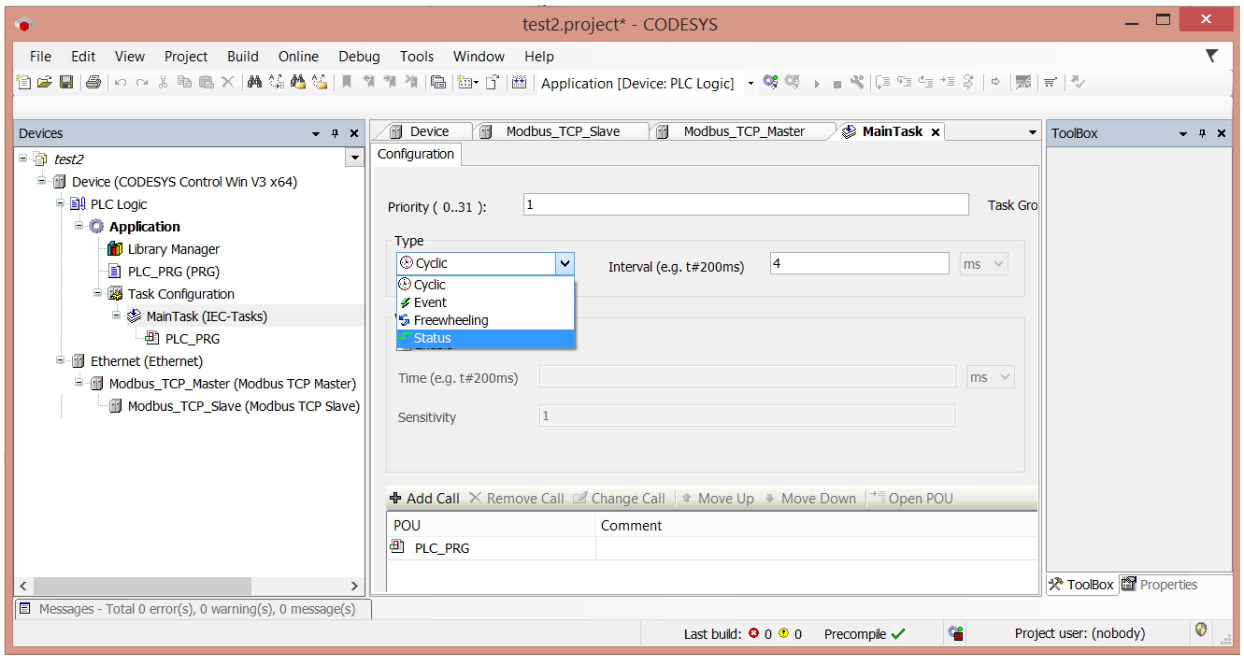
```
Devices
Teste_CanOpen_Master_PI_Guiao
  Device [connected] (CODESYS Control for Raspberry Pi MC SL)
    PLC Logic
      Application [run]
        Library Manager
        PLC_PRG (PRG)
        Task Configuration
          MainTask (IEC-Tasks)
            PLC_PRG
    CANbus (CANbus)
      CANopen_Manager (CANopen_Manager)
        B08CAN (B08CAN)
    SoftMotion General Axis Pool
    I²C
    SPI
    GPIOs_B_Pi2 (GPIOs B+/Pi2)
    Onewire
    Camera device
      <Empty>
```

2. Changing the configuration parameters

2.1 Changing Program Parameters

By double-clicking on the '*MainTask*' on the project explorer panel (window on the left) you can access the task (and program) activation parameters.

- Cyclic: The task/program (PLC scan cycle) is activated periodically, using the user defined period ('interval'). Appropriate delays are inserted at the end of each scan cycle to guarantee periodic activation.
- Freewheeling: The task/program is executed continuously. As soon as it finishes executing one scan cycle, the next is started again.
- Status/Event: The execution of the task/program is activated by an event on a user configured variable.



test2
- Device (CODESYS Control Win V3 x64)
  - PLC Logic
    - Application
      - Library Manager
      - PLC_PRG (PRG)
      - Task Configuration
        - MainTask (IEC-Tasks)
          - PLC_PRG
  - Ethernet (Ethernet)
    - Modbus_TCP_Master (Modbus TCP Master)
      - Modbus_TCP_Slave (Modbus TCP Slave)

Priority (0..31): 1

Type: Cyclic (Cyclic, Event, Freewheeling, Status)

Interval (e.g. t#200ms): 4 ms

Time (e.g. t#200ms): ms

Sensitivity: 1

POU	Comment
PLC_PRG	

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

Last build: 0 0 Precompile ✓ Project user: (nobody)

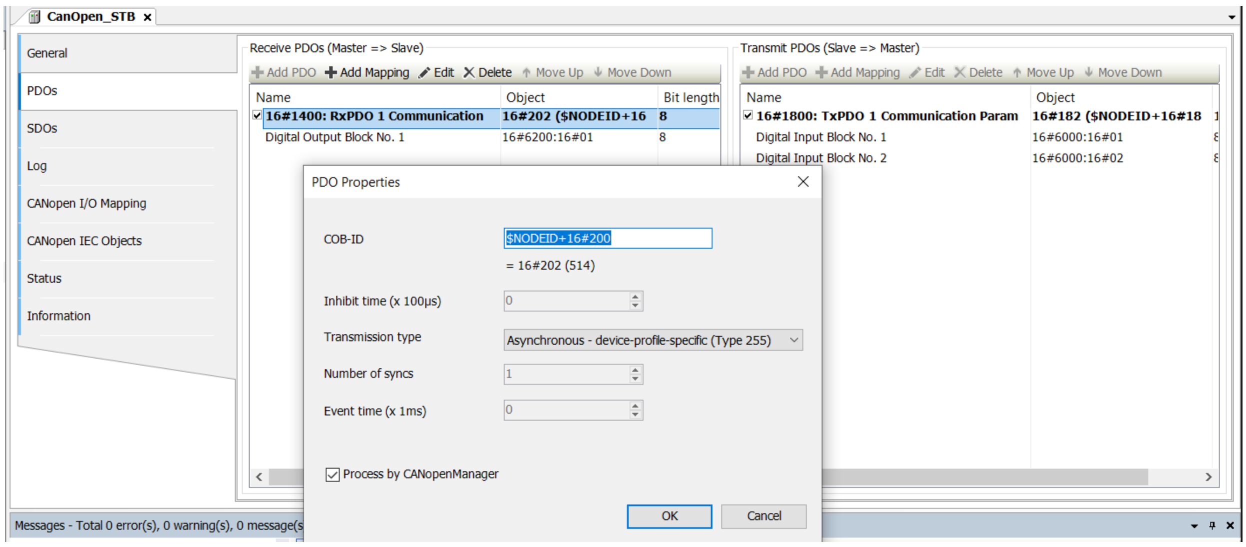
2.2 Changing CANopen parameters

For this assignment, we will need to change the type of PDOs that are used in the CANopen network.

There are several types of PDOs that define how they are transmitted:

- **Synchronous – acyclic (Type 0)**. The PDO transmission is synchronized with a SYNC message (i.e. transmitted after a SYNC), but not cyclically. An **RxPDO** is only evaluated (i.e. takes effect) after the next SYNC message has been received. A device whose **TxPDO** is configured in this way acquires its input data when it receives the SYNC and then transmits it if the data corresponds to an event (i.e., change in value, timer expiration, …).
- **Synchronous – cyclic (Type 1-240)**. The PDO is transmitted cyclically every n SYNC messages (n = 1...240). An **RxPDO** is set to valid (i.e. takes effect) only when the next SYNC is received. A device whose **TxPDO** is configured in this way acquires its input data when it receives the SYNC and then transmits it.
- **Synchronous – RTR only (Type 252)**. These PDOs are transmitted exclusively on request by a remote frame (RTR), thus these are **TxPDO**s, only. Once an RTR frame is received, the next SYNC triggers the process data acquisition and transmission.
- **Asynchronous – RTR only (Type 253)**. Same as the previous, but now asynchronously. The corresponding **TxPDO** is sent immediately after the reception of the RTR request frame.
- **Asynchronous – manufacturer specific (Type 254)**. The PDO is transmitted when an event specific of the manufacturer occurs (e.g. change of an input value)
- **Asynchronous – device-profile specific (Type 255)**. The PDO is transmitted when an event specific of the device profile occurs (e.g. change of an input value)

To change how a PDO is transmitted double click on the PDO name and you will be shown a window. In this window use the '*Transmission Type*' field to change the type of transmission according to your needs (do not change the COB-ID, i.e., the identifier of the PDO).



CanOpen_STB

General | PDOs | SDOs | Log | CANopen I/O Mapping | CANopen IEC Objects | Status | Information

Receive PDOs (Master => Slave)

Name	Object	Bit length
☑ 16#1400: RxPDO 1 Communication	16#202 ($NODEID+16	8
Digital Output Block No. 1	16#6200:16#01	8

Transmit PDOs (Slave => Master)

Name	Object
☑ 16#1800: TxPDO 1 Communication Param	16#182 ($NODEID+16#18
Digital Input Block No. 1	16#6000:16#01
Digital Input Block No. 2	16#6000:16#02

PDO Properties

COB-ID	$NODEID+16#200
	= 16#202 (514)
Inhibit time (x 100µs)	0
Transmission type	Asynchronous - device-profile-specific (Type 255)
Number of syncs	1
Event time (x 1ms)	0

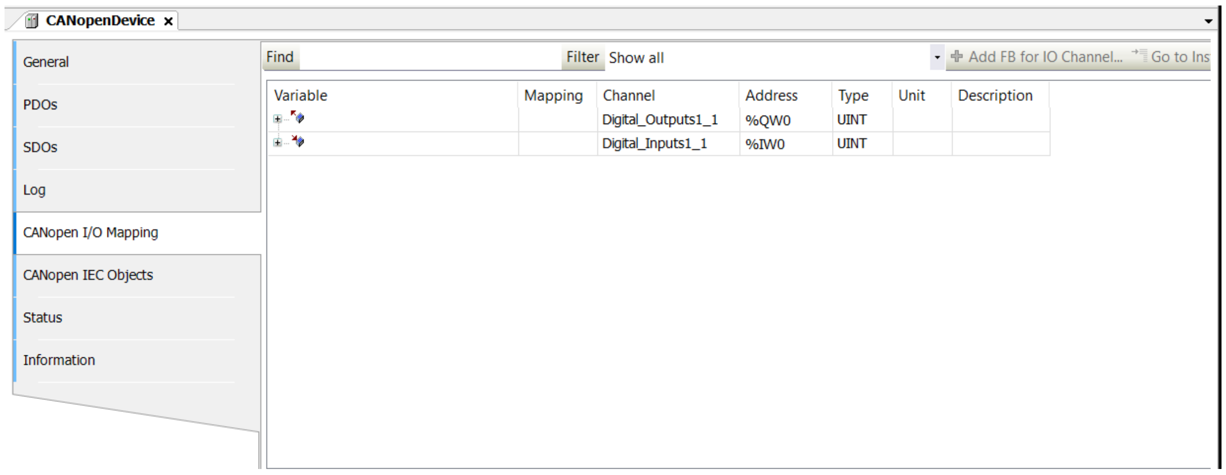
☑ Process by CANopenManager

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

3. Writing a simple program

Go offline from the runtime device.

Select the '**CANopen I/O Mapping**' tab on the Slave device.



CANopenDevice

General | PDOs | SDOs | Log | CANopen I/O Mapping | CANopen IEC Objects | Status | Information

Find | Filter Show all | Add FB for IO Channel... | Go to Ins

Variable	Mapping	Channel	Address	Type	Unit	Description
		Digital_Outputs1_1	%QW0	UINT		
		Digital_Inputs1_1	%IW0	UINT		

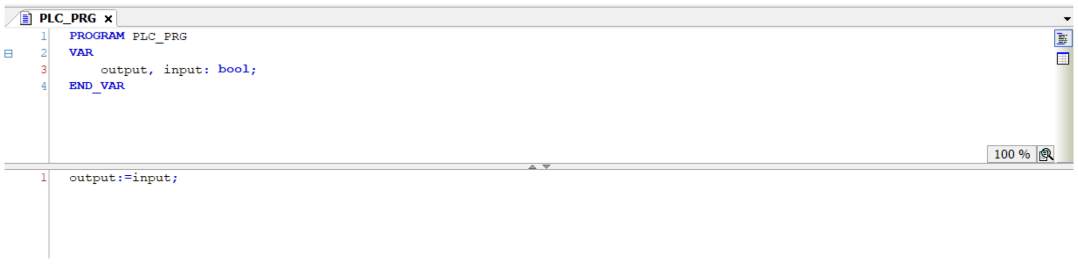
As you can check, Codesys has already mapped the PDOs to addresses. In this case:

- The **RxPDO 1** (Digital_Outputs1_1) is mapped on **%QW0**
- The **TxPDO 1** (Digital_Inputs1_1) is mapped on **%IW0**

Now, we are going to create a program that copies the status of the first digital input to the first digital of output, both on the slave.

Edit the program (**PLC_PRG** automatically inserted when the project was created) and create 2 variables, **output** and **input**, of type BOOL. Write the following code:

```
output:=input;
```



PLC_PRG

```
PROGRAM PLC_PRG
VAR
    output, input: bool;
END_VAR
```

```
output:=input;
```

Now, we need to map these variables on the PDOs. Select again the ‘**CANopen I/O Mapping**’ tab on the Slave device. Expand ‘*Digital_Outputs1_1*’ (click on ‘+’) and select the first output bit (BIT0). Double click the cell under the '*variable*' column, select $\boxed{\dots}$ and follow the path until find the variable **output**: *Application→PLC_PRG→Output*. The variable **output** is now mapped on the first bit of this PDO.

Repeat the previous procedure for the variable **input** (that must be mapped in the first bit (BIT0) of ‘*Digital_Inputs1_1*’)

Compile, download, and run this program.

You can check if this program is working properly using the Arduino-based experiment to measure the network TRR. The input should be periodically activated and the output should be activated correspondingly with a small delay (might be too small to be visible).

You can also test this configuration by using the “mirror” of output bits (**TxPDO 1**), which you can read and use their values to change the output. For that, you have only to make the following modification:

- Map the **input** variable on the 9th bit **TxPDO 1** (‘Echo Output Data’ channel 1)
- Change the program to **output:= not input;**

Compile, download, and run this program.

You should observe that the **output** variable is flipping its value.