

CANopen - Network Master

PLC500, PLC500ED, PLC500MC
PLC410

Application Note



Application Note

PLC410, PLC500, PLC500ED, PLC500MC

Document: 10010165216

Revision: 01

Publication Date: 11/2024

The information below describes the reviews made in this manual.

Version	Revision	Description
1.4.0	R01	Information update and inclusion of PLC models PLC410, PLC500ED and PLC500MC.
1.0.0	R00	First edition.

CONTENTS

1	INTRODUCTION	1-1
1.1	TERMS AND DEFINITIONS USED	1-1
1.2	NUMERICAL REPRESENTATION	1-1
1.3	REFERENCE DOCUMENTS	1-2
1.4	IMPORTANT NOTICE ABOUT CYBERSECURITY AND COMMUNICATIONS	1-2
1.5	TRADEMARKS	1-2
2	CAN INTERFACE	2-1
2.1	INDICATION LEDS	2-1
2.2	CAN NETWORK INSTALLATION	2-2
3	PROJECT IN CODESYS	3-1
3.1	PROJECT CREATION	3-1
3.2	ADDING MASTER DEVICE	3-2
3.3	ADDING SLAVE DEVICE	3-3
4	CANOPEN CONFIGURATION	4-1
4.1	TERMINATION RESISTORS	4-1
4.2	NETWORK CONFIGURATION	4-1
4.3	PDO CONFIGURATION	4-3
5	MONITORING	5-1
5.1	VARIABLE MONITORING	5-1
5.2	COMMUNICATION ERRORS	5-2
5.3	MONITORING USING FUNCTION BLOCKS	5-4

1 INTRODUCTION

This Application Note is intended to assist in the use of the **CANopen** protocol in WEG PLCs, models PLC410, PLC500, PLC500ED and PLC500MC. Throughout this document, the PLC500 is used as an example. However, the information presented is applicable to the other PLC models described above.

This document presents information about the CANopen protocol, the description of the CAN interface, installation recommendations and a guide to establish CANopen communication between the PLC500 (master) and the RUW100 remote unit (slave) in CODESYS.

For more information about hardware, interfaces and communication protocols, refer to the User Manual for the respective product, available at [WEG](#) website. For a more in-depth and detailed description of the CANopen network operation, access the online help at [CODESYS Online Help](#).



ATTENTION!

This application note is intended for professionals trained in industrial networks. Installation and configuration of devices must be done according to the manufacturer's manual.



NOTE!

This application note is aimed at the CODESYS programming tool, it is recommended to use version **CODESYS V3.5 SP19** or higher. It is also recommended to use the latest available version of the configuration libraries for the CANopen network.

1.1 TERMS AND DEFINITIONS USED

CAN: Controller Area Network is a communication protocol developed to allow real-time interaction between microcontrollers and devices in automotive and industrial environments.

CANopen: Higher-level communication protocol that operates on top of CAN, offering a set of services to manage communication between devices, including network management and device profiles.

CiA: CAN in Automation is an organization that develops and promotes standards and protocols based on CAN technology.

CODESYS: Programming platform that allows the development, configuration and monitoring of solutions for industrial automation and systems integration.

kbits: kilobits per second.

Mbps: Megabits per second.

m: meters.

min: minute.

ms: millisecond = 0.001 seconds.

ODVA: from the English “*Open DeviceNet Vendor Association*”, an organization that develops and promotes industrial communication standards based on the DeviceNet protocol and other technologies.

Ω: ohms.

mΩ: milliohms.

s: second.

USB: from the English “*Universal Serial Bus*”, a type of serial communication interface designed to work according to the “*Plug and Play*” concept.

1.2 NUMERICAL REPRESENTATION

Decimal numbers are represented by digits without a suffix. Hexadecimal numbers are represented with a prefix of '0x'.

1.3 REFERENCE DOCUMENTS

It is recommended to consult the documents related to the CANopen network presented in [Table 1.1 on page 1-2](#).

Table 1.1: Reference documents.

Document	Version	Source
CiA 106 Connector pin-assignment recommendations	V1.1.0	CiA
CiA 303-1 Device and network design - Part 1: CANopen physical layer	V2.0.1	CiA
CiA 306 CANopen electronic data sheet (ESD)	V1.3.0	CiA
DeviceNet Cable System Planning and Installation Manual	PUB00027R1	ODVA

1.4 IMPORTANT NOTICE ABOUT CYBERSECURITY AND COMMUNICATIONS

WEG PLCs, models PLC410, PLC500, PLC500ED and PLC500MC, have the ability to connect and exchange information through communication networks and protocols. Although they have been designed and tested to ensure proper operation with other automation systems using the protocols mentioned in this manual, it is essential that the customer understands the responsibilities associated with information and cybersecurity when using this equipment.

Therefore, it is the customer's sole responsibility to adopt defense-in-depth strategies and implement policies and measures to ensure the security of the system as a whole, including communications sent and received by the equipment. These measures include, but are not limited to, installing firewalls, antivirus and antimalware programs, data encryption, authentication control and physical control of user access.

WEG and its affiliates are not responsible for damages or losses resulting from cybersecurity breaches, including, but not limited to, unauthorized access, intrusion, leakage and/or theft of data or information, denial of service or any other form of security breach. The use of this product in conditions for which it was not specifically designed is not recommended and may cause damage to the product, the network and the automation system.

In this sense, it is essential that the customer understands that external interventions through third-party programs, such as sniffers or programs with similar actions, have the potential to cause interruptions or restrictions in the functionality of the equipment.

1.5 TRADEMARKS

All other trademarks are the property of their respective owners.

2 CAN INTERFACE

The CAN interface is shown in [Figure 2.1 on page 2-1](#) for PLC500 and PLC410. The CAN interface pinouts are described in [Table 2.1 on page 2-1](#).

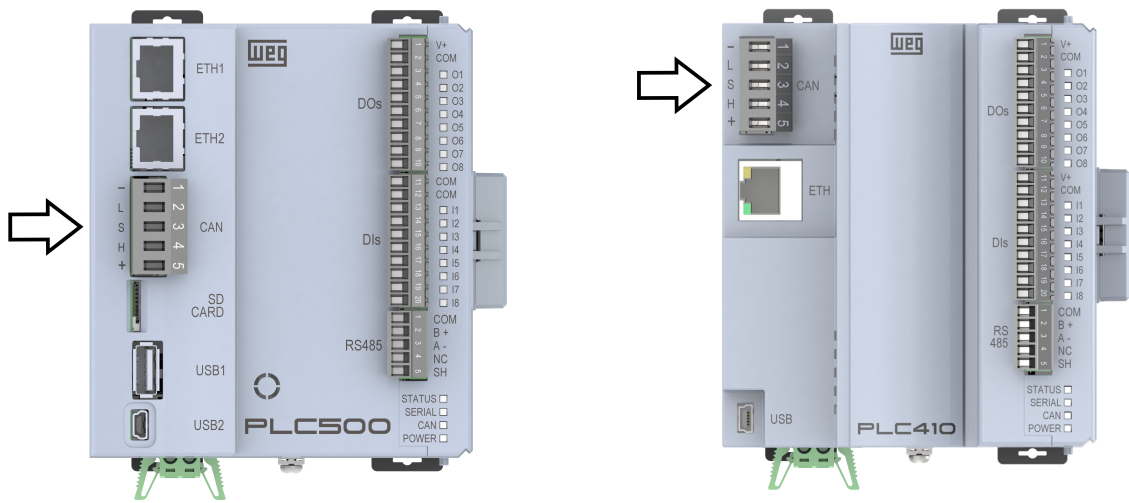


Figure 2.1: Indication of the CAN connector pins on the PLC500 and PLC410.

Table 2.1: Description of the CAN interface.

Pin	Name	Notation	Function
1	COM	-	Common (connected to the negative pole of the CAN network)
2	CAN_L	L	CAN_L communication signal
3	SHIELD	S	Cable shielding
4	CAN_H	H	CAN_H communication signal
5	NC	+	Not connected (can receive the positive pole of the CAN network)

The PLC410, PLC500, PLC500ED and PLC500MC allow the connection of up to 126 network slaves. The CAN interface is isolated with its own internal power supply, so pin 5 does not need to be connected. Despite this, it is recommended that pin 1 (COM) be connected to the master and the other slaves to keep the CAN network at the same voltage reference.

2.1 INDICATION LEDs

The PLC500 and PLC410 have LEDs on the front that are used to indicate the status of the CAN interface. The color indication of the LED shown in [Figure 2.2 on page 2-1](#) is described in [Table 2.2 on page 2-2](#).



Figure 2.2: Indicator LEDs of the PLC500 and PLC410.

Table 2.2: Operation of the LED indicator of the CAN interface.

CAN LED	STATUS	DESCRIPTION
OFF	No configuration.	There is no configuration for the CANopen Master communication in the application.
GREEN	CANopen Master communication without failure.	CANopen Master communication configured in the application, and all elements are running without error/failure.
ORANGE	CANopen Master communication with partial failure.	CANopen Master communication configured in the application with some elements experiencing error/failure.
RED	CANopen Master communication with total failure.	CANopen Master communication configured in the application with all elements experiencing error/failure.

2.2 CAN NETWORK INSTALLATION

The CAN network, like many industrial communication networks, is often used in harsh environments with high exposure to electromagnetic interference, and requires certain precautions to ensure a low rate of communication errors during its operation.



ATTENTION!

For passive network components such as cables, connectors, termination resistors and power supply, it is recommended to use only components certified for industrial applications.

The PLC500 allows the configuration of the baud rate used by the CAN interface. It should be selected according to the amount of data to be transmitted and the length of the cable in the installation. [Table 2.3 on page 2-2](#) shows the communication rates supported by the PLC500 and the maximum recommended cable length, as specified by the protocol.

Table 2.3: Supported communication rates and cable length.

Communication Rate	Cable Length
10 kbps	1000 m
20 kbps	1000 m
50 kbps	1000 m
100 kbps	600 m
125 kbps	500 m
250 kbps	250 m
500 kbps	100 m
800 kbps	50 m
1 Mbps	25 m

For connecting the **CAN_L** and **CAN_H** signals, a shielded twisted pair must be used. For equipment that requires external power for the CAN interface, the positive and negative pair must also be twisted. [Table 2.4 on page 2-2](#) shows the recommended characteristics for the cable.

Table 2.4: Characteristics of the cable for CAN network.

Cable Length	Resistance per meter	Conductor Area
between 0 to 40 m	70 mΩ/m	between 0.25 to 0.34 mm ²
between 40 m to 300 m	less than 60 mΩ/m	between 0.34 to 0.60 mm ²
between 300 m to 600 m	less than 40 mΩ/m	between 0.50 to 0.60 mm ²
between 600 m to 1000 m	less than 26 mΩ/m	between 0.75 to 0.80 mm ²

To interconnect the various network nodes, it is recommended to connect the equipment directly from the main line, without using branches. When installing the cables, avoid placing them near power cables, as this can lead to errors during transmission due to electromagnetic interference.

To avoid problems with current circulation due to potential differences between different grounds, all devices must be connected to the same ground point.

**DANGER!**

Faulty grounding installations can lead to the death and malfunction of the CAN network.

This application note will use the example of a PLC500 as a network master and a RUW100 remote unit as a network slave. [Figure 2.3 on page 2-3](#) shows the components and architecture of the CANopen network. A computer programs the PLC500 using CODESYS. The RUW100 also has an isolated CAN interface with its own internal power supply, meaning that pin 5 does not need to be connected.

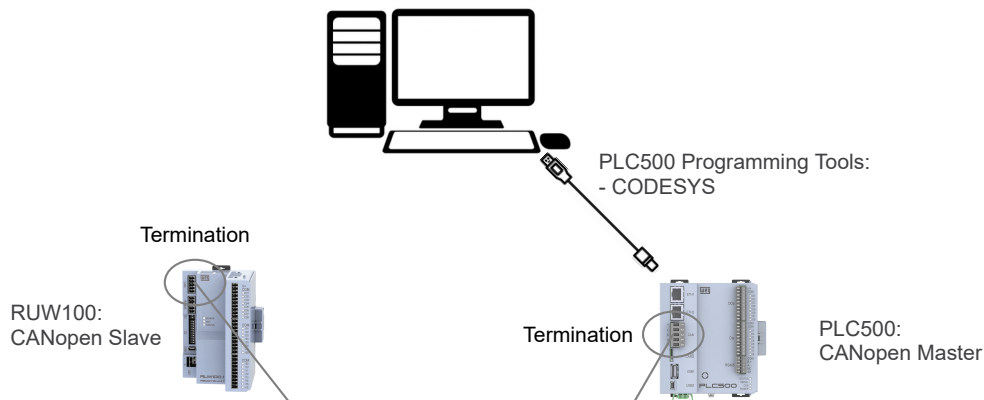


Figure 2.3: Components and architecture of the CANopen network example.

The RUW100 requires a termination resistor worth $120\ \Omega$ | $0.25\ W$ to be connected between pins 2 and 4. The PLC500 termination resistor can be added via CODESYS, the procedure for doing so will be shown in [Chapter 4 CANOPEN CONFIGURATION on page 4-1](#).

**NOTE!**

The RUW100 can be programmed using the **Weg Programming Suite (WPS)** software, but for the example presented in this application note, it is not necessary to use it.

In other situations where network devices require external power, it is recommended that the network be powered at only one point, avoiding problems with voltage differences in the power supply between network devices. If more than one power supply is required, they must be located at points where the voltage is at adequate levels for the slaves to function properly. The power supplies must be properly dimensioned to supply power to these devices and must be referenced to the same point.

**ATTENTION!**

Grounding the negative pole of the PLC500 or RUW100 power supply and pin 1 of the CAN network may cause damage to the devices due to loss of power supply isolation.

**NOTE!**

For more detailed information on the physical interface of the CANopen network, we recommend consulting the document CiA 303-1 Device and network design - Part 1: CANopen physical layer, available at: can-cia.org. For general installation guidance in an industrial environment, we recommend the DeviceNet Cable System Planning and Installation Manual available at: odva.org.

3 PROJECT IN CODESYS

In this chapter, the steps for creating a project and adding the master and slave devices of the CANopen network in CODESYS are presented

3.1 PROJECT CREATION

To configure and define a CANopen network, you must first create the project and include the PLC500 programmable controller. In the CODESYS software, create a new project using the **Standard project** template, choose the directory and the name of the application. Then, select the PLC500 device and the desired programming language, as shown in [Figure 3.1 on page 3-1](#).

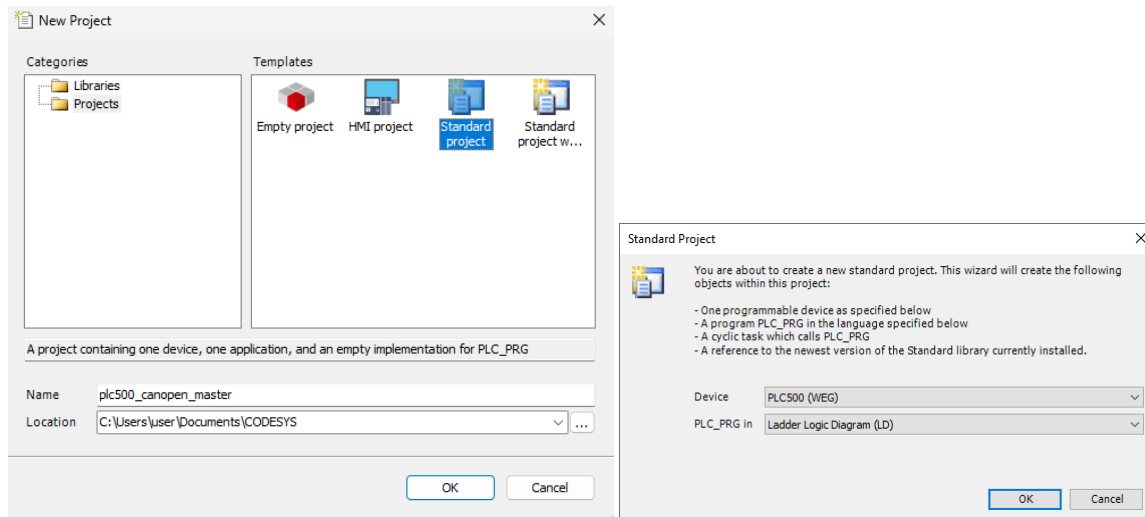


Figure 3.1: Project creation in CODESYS.



NOTE!

If the PLC410, PLC500, PLC500ED or PLC500MC device is not yet available in the CODESYS options, you must download and install the configuration file. Check the **Product Manual** to find the necessary steps and settings.

With the PLC500 device selected, it will result in the project tree as shown in [Figure 3.2 on page 3-1](#).

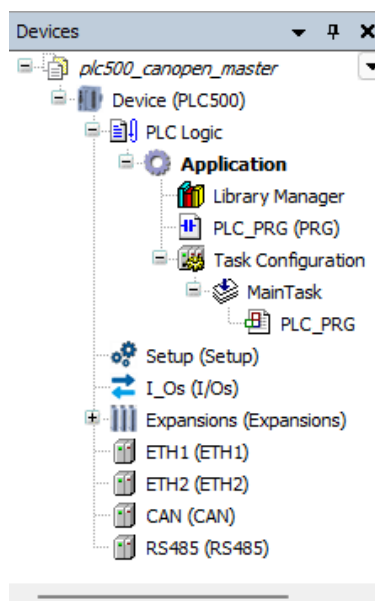


Figure 3.2: Initial project tree of PLC500.

3.2 ADDING MASTER DEVICE

In the next step, you must add the **CANopen_Manager** device. Right-click on the CAN network icon and select **Add Device**; a new window will open. Expand the CANopen and CANopenManager icons using the **+** symbol until the **CANopen_Manager** device appears, as shown in [Figure 3.3 on page 3-2](#). With it selected, click **Add Device** in the window. The device will be added to the project tree just below the CAN interface as shown in [Figure 3.4 on page 3-2](#).

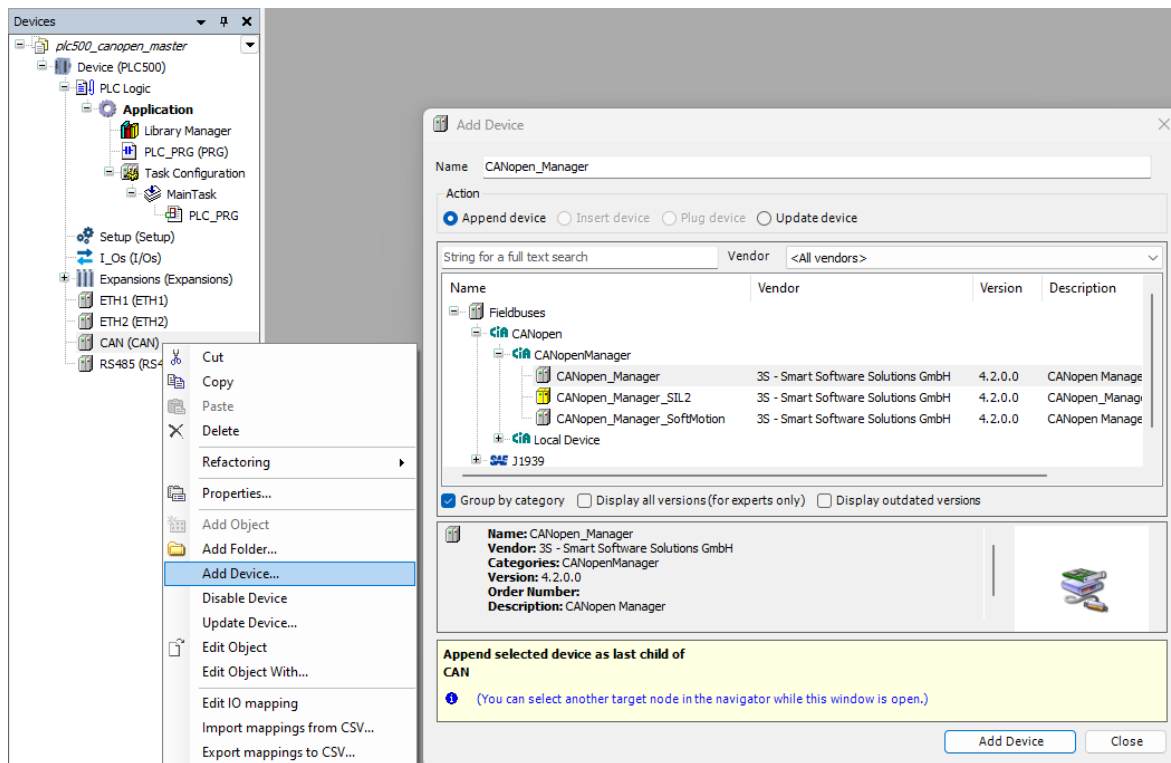


Figure 3.3: Adding CANopen_Manager master device.

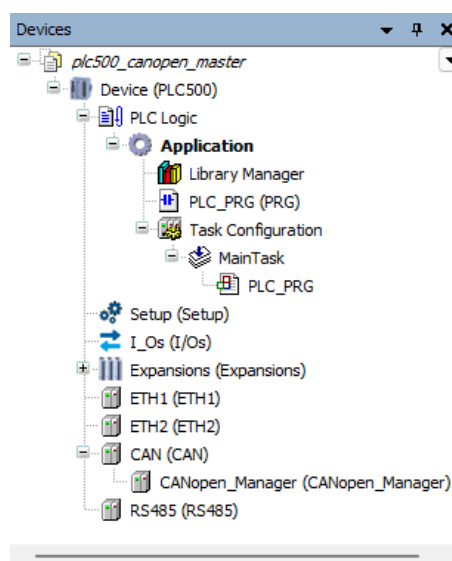


Figure 3.4: CANopen_Manager master device added to the project tree.

3.3 ADDING SLAVE DEVICE

Left-clicking on the **CANopen_Manager** icon adds the slave device to the CANopen network. In this case, the RUW100 with its corresponding firmware version is selected. [Figure 3.5 on page 3-3](#) shows the steps for this configuration.

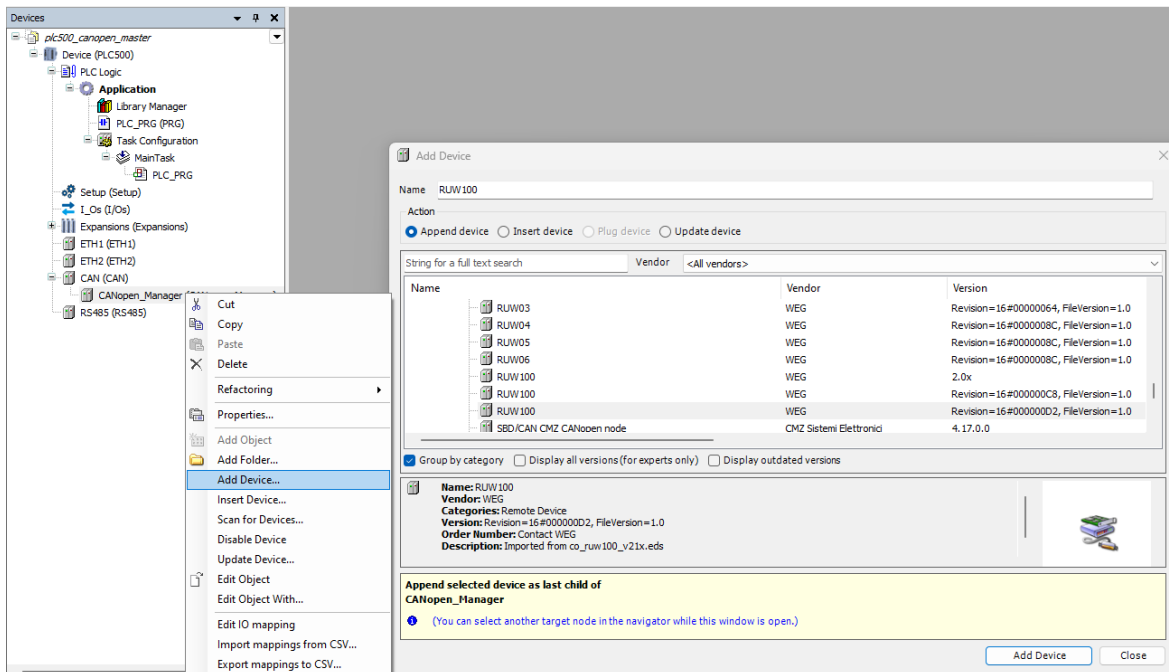


Figure 3.5: Adding slave device to the network.

If this is the first time that a device is added as a CANopen network slave device, you must obtain the respective EDS file for the product and import it into CODESYS. For the RUW100, the EDS file can be found on the product page at [WEG](#) website. Also check if the EDS file version is compatible with the product firmware version. The EDS file installation must be done in **Tools** in the **Device Repository** window, as shown in [Figure 3.6 on page 3-3](#).

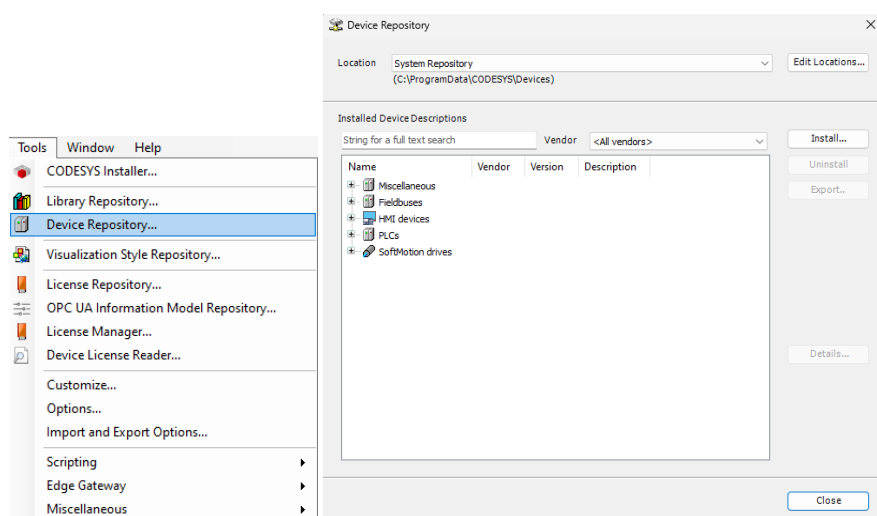


Figure 3.6: Installing a new device in Device Repository in CODESYS.

Click on **Install...** and select the EDS file as shown in the [Figure 3.7 on page 3-4](#). After selecting the EDS, the Device Repository window will display a message that the device has been installed and its icon in the device list.

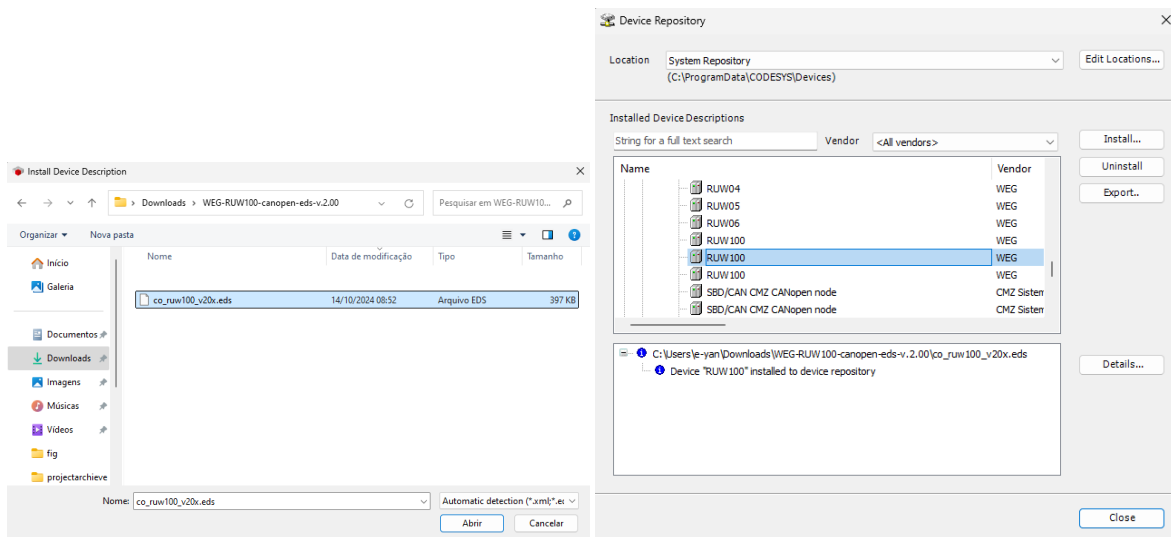


Figure 3.7: Selecting the EDS file and successful device installation.



NOTE!

Electronic Device Description (EDS) is the file required to describe the slave device to the CANopen network master. The manufacturer must provide this file to allow the correct configuration of this device when the network slave option is available.

At this point, the CAN interface must have the items indicated in [Figure 3.8 on page 3-4](#). More slave devices can be added following the same procedures adopted.

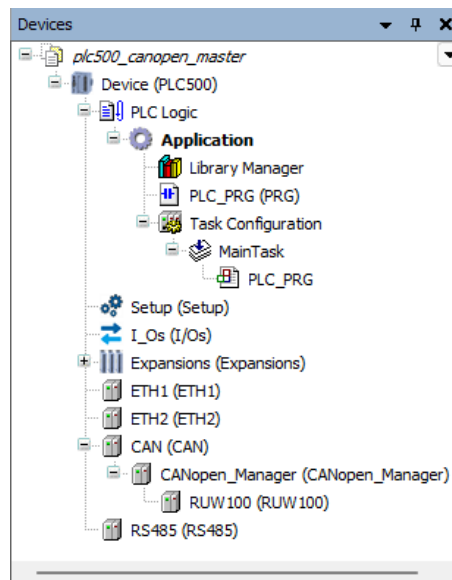


Figure 3.8: CAN interface with RUW100 master and slave.

4 CANOPEN CONFIGURATION

In this chapter, the steps for configuring a CANopen network with a PLC500 master and a RUW100 slave in CODESYS are presented.

4.1 TERMINATION RESISTORS

For the CAN network to function correctly, it is essential to use termination resistors. You can choose an external resistor or enable the internal resistor directly through CODESYS. In PLC500, the internal resistor can be configured through the **Setup** → **Setup Parameters** → **Termination Resistors** → **CAN** object, as shown in Figure 4.1 on page 4-1.

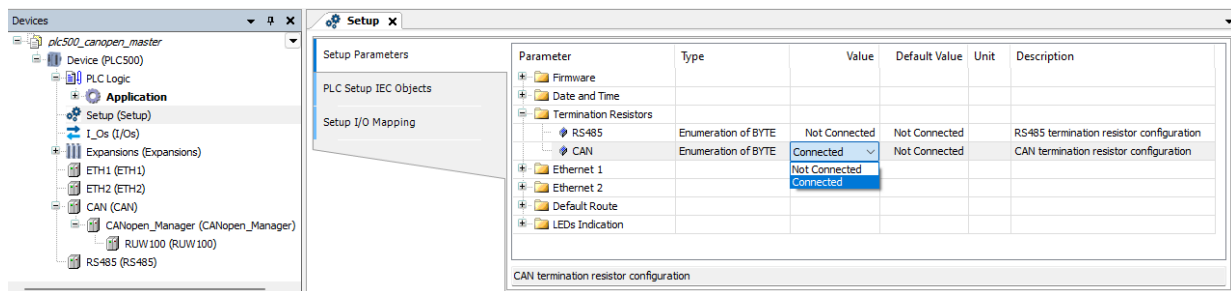


Figure 4.1: Termination resistors configuration page.



NOTE!

For the RUW100, the CANopen network termination resistor must be physically added to pins 2 and 4 of the connector.

4.2 NETWORK CONFIGURATION

The network must be configured in the master to meet the addresses defined in the network slaves. To do this, the parameters and addresses that will be transmitted from the network master must be defined. Each slave device must be configured with a unique address and at the same defined baud rate, otherwise the network will present problems.

Double-clicking on the CAN interface in the project tree opens a window as shown in Figure 4.2 on page 4-1. In this window, the network of the CAN interface used and the baud rate are defined. The **Network** parameter refers to the assigned CAN interface that is being used. The PLC500 has only one CAN interface available, so its number is kept as 0. The communication rate can be changed by the **Baud rate** parameter. In the example, the baud rate is set to 250 kbps.

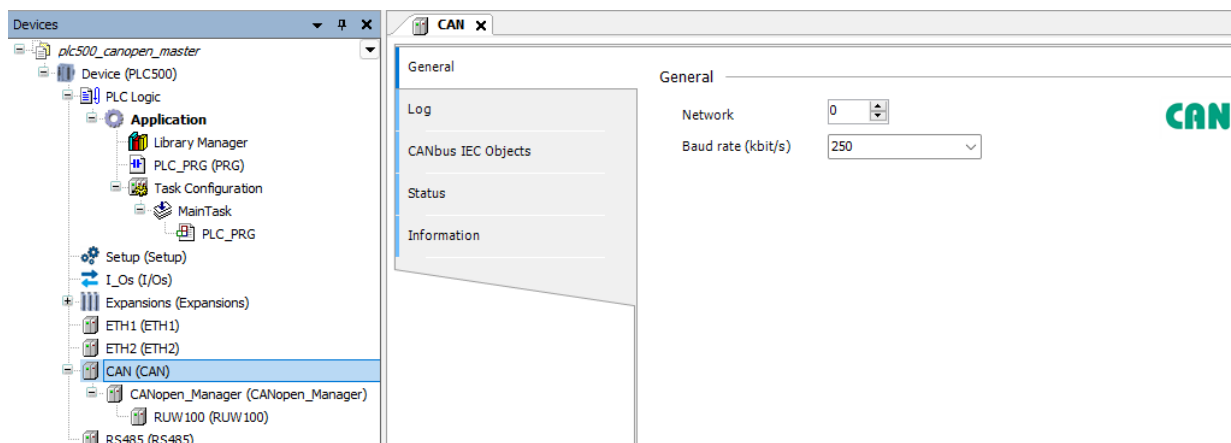


Figure 4.2: CANopen network baud rate configuration.

CANOPEN CONFIGURATION

By double-clicking on the **CANopen_Manager** icon, the screen shown in [Figure 4.3 on page 4-2](#) is displayed. This screen is used to define the settings for the master device of the CANopen network. The master is addressed using the **Node ID** parameter, which must be between 1 and 127. In the **Check and fix configuration** option, CODESYS checks whether the CANopen network configuration is correct and suggests automatic corrections. For the example in this application, the **Node ID** was set to 127. The remaining settings were left as default.

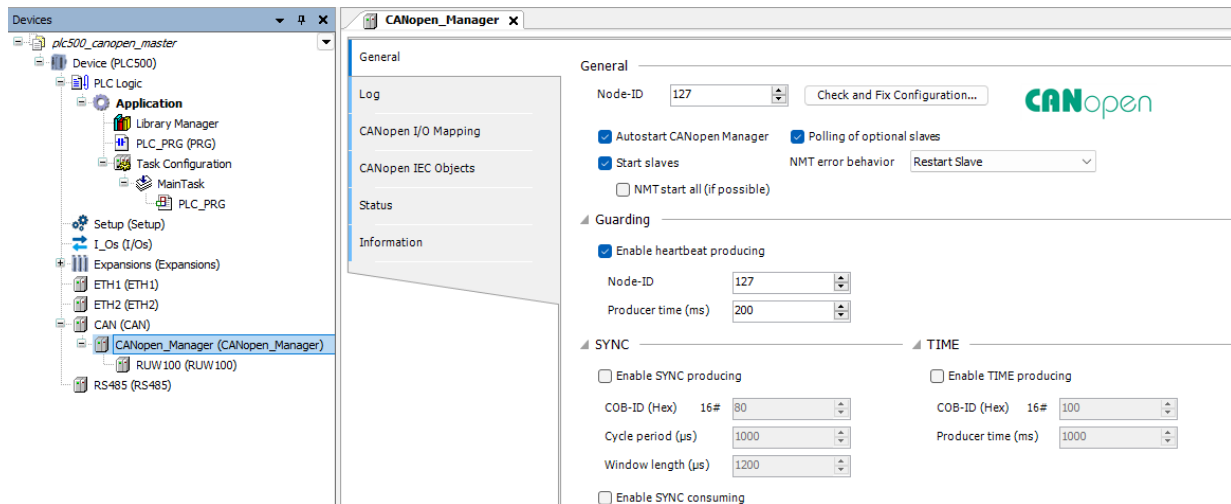


Figure 4.3: CANopen configuration of the master device.

To configure the slave device, double-click on the RUW100 icon and a window opens as shown in [Figure 4.4 on page 4-2](#). The **Enable expert settings** option is enabled and the other settings are left as default.

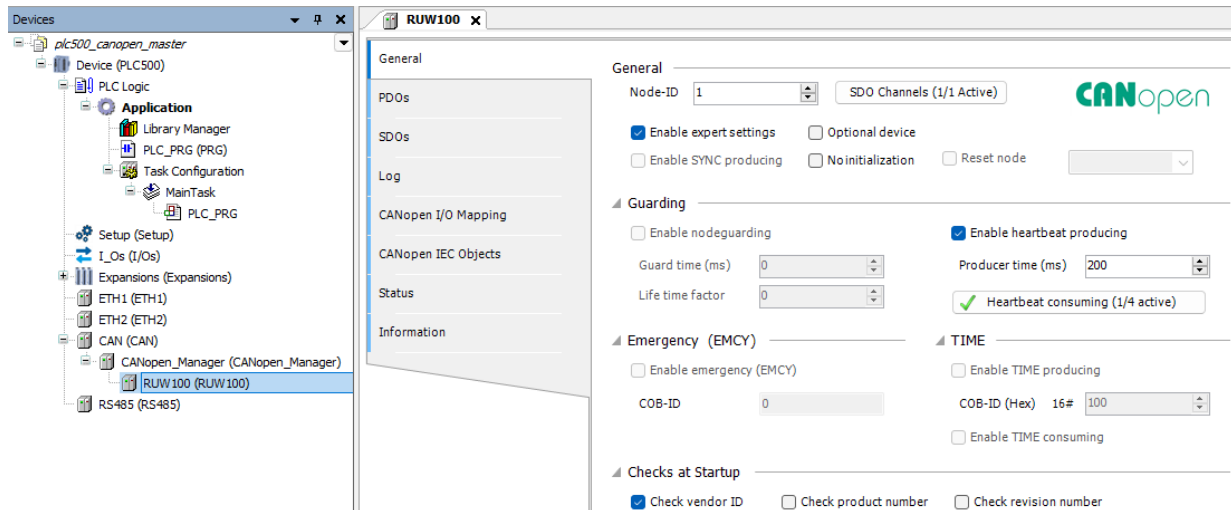


Figure 4.4: CANopen configuration of the slave device.



NOTE!

For more information about all available CANopen network settings in CODESYS, see the online help available at [CODESYS Online Help](#).

After completing the previous step in CODESYS, configure address 1 and the baud rate of 250 kbps directly on the DIP switches on the exterior of the RUW100. Detailed information can be found in the product manual available at [WEG](#) website.

**NOTE!**

The method of configuring the addressing and baud rate may be different for each slave device, depending on the type of product and the programming software. Configure the slave device according to the manual provided by the manufacturer.

4.3 PDO CONFIGURATION

Variables and addresses configured for a slave must be programmed taking into account the master of that CANopen network. After importing the CANopen network slave into CODESYS, the set of PDOs (Process Data Object) will be available according to each device as shown in [Figure 4.5 on page 4-3](#).

Name	Object	Bit length
☒ 16#1400: Receive PDO Communication Parameter 1	16#201 (\$NODEID+16#200)	16
Write Output 1 to 8	16#6200sub01	8
Write Output 9 to 16	16#6200sub02	8
☒ 16#1401: Receive PDO Communication Parameter 2	16#301 (\$NODEID+16#300)	16
Write Output 1 to 16	16#6300sub01	16
☒ 16#1402: Receive PDO Communication Parameter 3	16#401 (\$NODEID+16#400)	64
Write Analogue Output 1	16#6411sub01	16
Write Analogue Output 2	16#6411sub02	16
Write Analogue Output 3	16#6411sub03	16
Write Analogue Output 4	16#6411sub04	16
☒ 16#1403: Receive PDO Communication Parameter 4	16#501 (\$NODEID+16#500)	64
Write Analogue Output 5	16#6411sub05	16
Write Analogue Output 6	16#6411sub06	16
Write Analogue Output 7	16#6411sub07	16
Write Analogue Output 8	16#6411sub08	16
☐ 16#1404: Receive PDO Communication Parameter 5	16#0	0
☐ 16#1405: Receive PDO Communication Parameter 6	16#0	0
☐ 16#1406: Receive PDO Communication Parameter 7	16#0	0
☐ 16#1407: Receive PDO Communication Parameter 8	16#0	0
☐ 16#1408: Receive PDO Communication Parameter 9	16#0	0
☐ 16#1409: Receive PDO Communication Parameter 10	16#0	0
☐ 16#140A: Receive PDO Communication Parameter 11	16#0	0
☐ 16#140B: Receive PDO Communication Parameter 12	16#0	0

Name	Object	Bit length
☒ 16#1800: Transmit PDO Communication Parameter 1	16#181 (\$NODEID+16#180)	16
Read Input 1 to 8	16#6000sub01	8
Read Input 9 to 16	16#6000sub02	8
☒ 16#1801: Transmit PDO Communication Parameter 2	16#281 (\$NODEID+16#280)	16
Read Input 1 to 16	16#6100sub01	16
☒ 16#1802: Transmit PDO Communication Parameter 3	16#381 (\$NODEID+16#380)	64
Read Analogue Input 1	16#6401sub01	16
Read Analogue Input 2	16#6401sub02	16
Read Analogue Input 3	16#6401sub03	16
Read Analogue Input 4	16#6401sub04	16
☒ 16#1803: Transmit PDO Communication Parameter 4	16#481 (\$NODEID+16#480)	64
Read Analogue Input 5	16#6401sub05	16
Read Analogue Input 6	16#6401sub06	16
Read Analogue Input 7	16#6401sub07	16
Read Analogue Input 8	16#6401sub08	16
☐ 16#1804: Transmit PDO Communication Parameter 5	16#0	0
☐ 16#1805: Transmit PDO Communication Parameter 6	16#0	0
☐ 16#1806: Transmit PDO Communication Parameter 7	16#0	0
☐ 16#1807: Transmit PDO Communication Parameter 8	16#0	0
☐ 16#1808: Transmit PDO Communication Parameter 9	16#0	0
☐ 16#1809: Transmit PDO Communication Parameter 10	16#0	0
☐ 16#180A: Transmit PDO Communication Parameter 11	16#0	0
☐ 16#180B: Transmit PDO Communication Parameter 12	16#0	0

Figure 4.5: List of PDOs available in the RUW100.

By default, the RUW100 has 4 PDOs enabled after its addition to the network for reception (master → slave) and transmission (slave → master). The equipment provides a total of 32 PDOs for sending and another 32 PDOs for receiving data, which can be individually configured to transmit up to 8 bytes of data.

**NOTE!**

For each application, specific configurations and addresses may arise. Therefore, the manufacturer must inform the availability of PDOs according to the equipment. It is recommended to check the specific product manual for more details.

For each application, the transmission addresses and which data are relevant can be defined. For example, the PLC500 writes a value to the digital outputs and reads the status of the digital inputs of the RUW100 every 100 milliseconds. In [Figure 4.6 on page 4-3](#) there is a schematic representation of this network configuration.

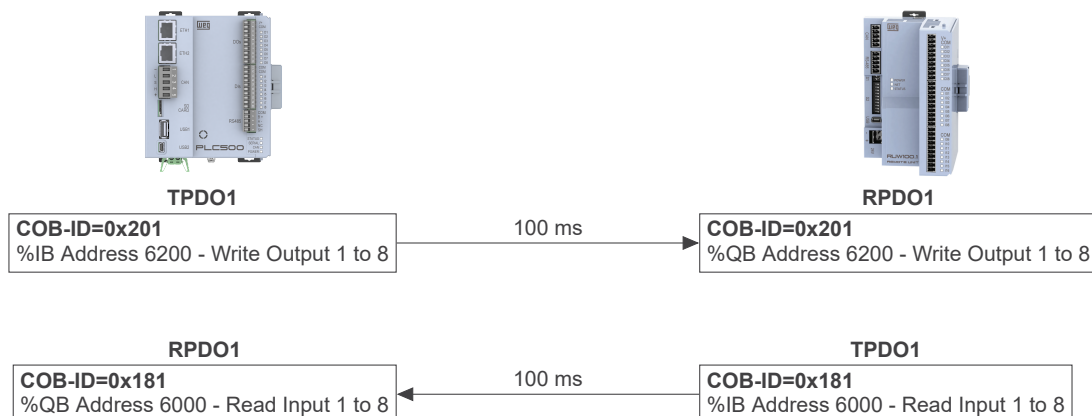


Figure 4.6: Schematic representation of PDO configuration.

CANOPEN CONFIGURATION

New PDOs can be enabled and configured with their respective COB-ID and transmission type. There are different types of PDO transmission that can be configured as follows:

- **Acyclic – synchronous (Type 0):** indicates that transmission should occur immediately after reception of a SYNC telegram, but not periodically.
- **Cyclic – synchronous (Type 1 - 240):** the PDO should be transmitted for each detected SYNC telegram or multiple occurrences of SYNC.
- **Synchronous – RTR only (Type 252):** indicates that the message content should be updated (but not sent) after reception of a SYNC telegram. The message should be sent after reception of a remote frame (RTR frame).
- **Asynchronous – RTR only (Type 253):** the PDO should update and send a message as soon as it receives an RTR.
- **Asynchronous – manufacturer-specific (Type 254):** the object must be transmitted periodically according to a scheduled time.
- **Asynchronous – device profile-specific (Type 255):** the PDO is transmitted according to the CiA device profile.



NOTE!

The RUW100 follows the device profile described by CiA DPS 401 - Device Profile for Generic I/O Modules. Therefore, the PDO configured for transmission type 255 will be transmitted automatically when the value of any of the mapped objects is changed or periodically according to a programmed time. It is necessary to check which device profile the slave follows to know how the PDO transmission is performed for type 255.

The enabled PDOs of the RUW100 shown in [Figure 4.5 on page 4-3](#) come preconfigured with transmission type 255 for reception telegrams and type 254 for transmission telegrams. The transmission types can be changed individually when the PDO is selected by the **Edit** button or by left-clicking. As an example, the parameters of the receiving PDO 1 (master → slave) are edited as shown in [Figure 4.7 on page 4-4](#).

Name	Object	Bit length
☒ 16#1400: Receive PDO Communication Parameter 1	16#201 (\$NODEID+16#200)	16
Write Output 1 to 8	16#6200sub01	8
Write Output 9 to 16	16#6200sub02	8
☒ 16#1401: Receive PDO Communication Parameter 2	16#301 (\$NODEID+16#300)	16
Write Output 1 to 16	16#6300sub01	16
☒ 16#1402: Receive PDO Communication Parameter 3	16#401 (\$NODEID+16#400)	64
Write Analogue Output 1	16#6411sub01	16
Write Analogue Output 2	16#6411sub02	16
Write Analogue Output 3	16#6411sub03	16
Write Analogue Output 4	16#6411sub04	16
☒ 16#1403: Receive PDO Communication Parameter 4	16#501 (\$NODEID+16#500)	64
Write Analogue Output 5	16#6411sub05	16
Write Analogue Output 6	16#6411sub06	16
Write Analogue Output 7	16#6411sub07	16
Write Analogue Output 8	16#6411sub08	16
☐ 16#1404: Receive PDO Communication Parameter 5	16#0	0
☐ 16#1405: Receive PDO Communication Parameter 6	16#0	0
☐ 16#1406: Receive PDO Communication Parameter 7	16#0	0
☐ 16#1407: Receive PDO Communication Parameter 8	16#0	0
☐ 16#1408: Receive PDO Communication Parameter 9	16#0	0

Figure 4.7: Editing the parameters of PDO 1.

A new screen will open as shown in [Figure 4.8 on page 4-5](#). The PDOs must have a unique COB-ID address in the CANopen network. To facilitate the addressing of several slaves in a CANopen network, the **\$NODEID** function acquires the address configured in the **General** tab of the device. For the example, **\$NODEID** is equal to 1 and added to 200 in hexadecimal (16#200) gives the COB-ID 16#201 (0x201). The **Inhibit time** is set

to 2000 microseconds, transmission type (**Transmissiontype**) to 255 and transmission period (**Event time**) to 100 milliseconds. The same configuration is done for the receiving PDO 2, except for its COB-ID, which is maintained.



ATTENTION!

Setting the **Inhibit time** to zero in a CAN network can lead to increased collisions, message loss, network congestion, and potential disruptions in system timing and performance. It is typically advisable to configure a suitable inhibit time based on the specific application requirements to ensure reliable communication.

Figure 4.8: Editing the configuration of the receiving PDO 1.

The same procedure is performed for the transmitting PDOs 1 and 2 (slave → master), their COB-IDs are maintained and their transmission modes are type 254, **Inhibit time** is configured as 2000 microseconds and **Event time** as 100 milliseconds for both.



NOTE!

You must pay attention to the COB-ID range occupied, especially when several slaves are inserted in the same CANopen network. If a PDO in the CANopen network has the same COB-ID as another, CODESYS will report an error. You can choose to automatically correct the configuration error using the **Check and Fix Configuration** option in the **General** tab of **CANopen_Manager**.

The other PDOs can be disabled by removing the checkbox, as indicated in [Figure 4.9 on page 4-5](#).

Receive PDOs (Master => Slave)				Transmit PDOs (Slave => Master)			
Name	Object	Bit length		Name	Object	Bit length	
☒ 16#1400: Receive PDO Communication Parameter 1	16#201 (\$NODEID+16#200)	16		☒ 16#1800: Transmit PDO Communication Parameter 1	16#181 (\$NODEID+16#180)	16	
Write Output 1 to 8	16#6200sub01	8		Read Input 1 to 8	16#6000sub01	8	
Write Output 9 to 16	16#6200sub02	8		Read Input 9 to 16	16#6000sub02	8	
☒ 16#1401: Receive PDO Communication Parameter 2	16#301 (\$NODEID+16#300)	16		☒ 16#1801: Transmit PDO Communication Parameter 2	16#281 (\$NODEID+16#280)	16	
Write Output 1 to 16	16#6300sub01	16		Read Input 1 to 16	16#6100sub01	16	
☐ 16#1402: Receive PDO Communication Parameter 3	16#401 (\$NODEID+16#400)	64		☐ 16#1802: Transmit PDO Communication Parameter 3	16#381 (\$NODEID+16#380)	64	
Write Analogue Output 1	16#6411sub01	16		Read Analogue Input 1	16#6401sub01	16	
Write Analogue Output 2	16#6411sub02	16		Read Analogue Input 2	16#6401sub02	16	
Write Analogue Output 3	16#6411sub03	16		Read Analogue Input 3	16#6401sub03	16	
Write Analogue Output 4	16#6411sub04	16		Read Analogue Input 4	16#6401sub04	16	
☐ 16#1403: Receive PDO Communication Parameter 4	16#501 (\$NODEID+16#500)	64		☐ 16#1803: Transmit PDO Communication Parameter 4	16#481 (\$NODEID+16#480)	64	
Write Analogue Output 5	16#6411sub05	16		Read Analogue Input 5	16#6401sub05	16	
Write Analogue Output 6	16#6411sub06	16		Read Analogue Input 6	16#6401sub06	16	
Write Analogue Output 7	16#6411sub07	16		Read Analogue Input 7	16#6401sub07	16	
Write Analogue Output 8	16#6411sub08	16		Read Analogue Input 8	16#6401sub08	16	
☐ 16#1404: Receive PDO Communication Parameter 5	16#0	0		☐ 16#1804: Transmit PDO Communication Parameter 5	16#0	0	
☐ 16#1405: Receive PDO Communication Parameter 6	16#0	0		☐ 16#1805: Transmit PDO Communication Parameter 6	16#0	0	
☐ 16#1406: Receive PDO Communication Parameter 7	16#0	0		☐ 16#1806: Transmit PDO Communication Parameter 7	16#0	0	
☐ 16#1407: Receive PDO Communication Parameter 8	16#0	0		☐ 16#1807: Transmit PDO Communication Parameter 8	16#0	0	
☐ 16#1408: Receive PDO Communication Parameter 9	16#0	0		☐ 16#1808: Transmit PDO Communication Parameter 9	16#0	0	
☐ 16#1409: Receive PDO Communication Parameter 10	16#0	0		☐ 16#1809: Transmit PDO Communication Parameter 10	16#0	0	
☐ 16#140A: Receive PDO Communication Parameter 11	16#0	0		☐ 16#180A: Transmit PDO Communication Parameter 11	16#0	0	
☐ 16#140B: Receive PDO Communication Parameter 12	16#0	0		☐ 16#180B: Transmit PDO Communication Parameter 12	16#0	0	
☐ 16#140C: Receive PDO Communication Parameter 13	16#0	0		☐ 16#180C: Transmit PDO Communication Parameter 13	16#0	0	

Figure 4.9: PDOs configured for the example with RUW100.



NOTE!

For larger applications where a large data flow is required in the network, it is recommended to leave enabled only the PDOs that will be used.

The receive and transmit PDOs 5...32 can be configured to map other objects. To map new objects in a PDO, use the **Add mapping** option, as shown in [Figure 4.10 on page 4-6](#).

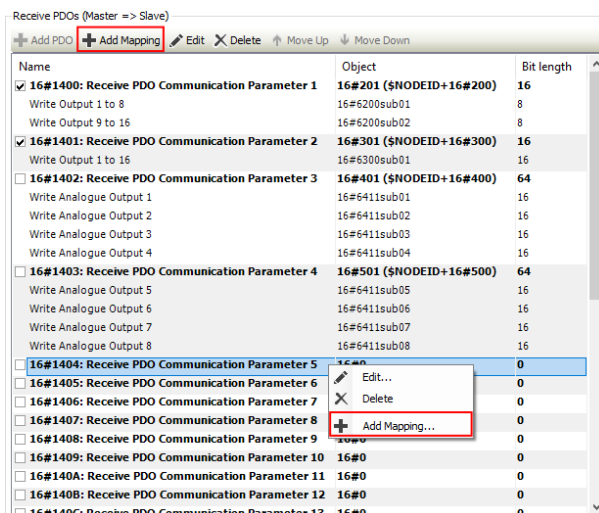


Figure 4.10: Adding new objects to a PDO.

After that, a window opens with the list of available objects according to the slave device. Using the + buttons, you can open the sub parameters that can be selected as shown in [Figure 4.11 on page 4-6](#). If the number of objects exceeds the 64-bit size, an error will appear, as indicated in [Figure 4.12 on page 4-7](#).

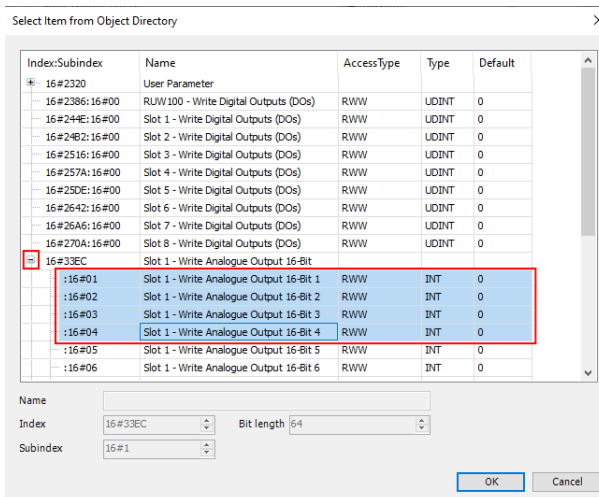


Figure 4.11: Selecting new objects to a PDO.

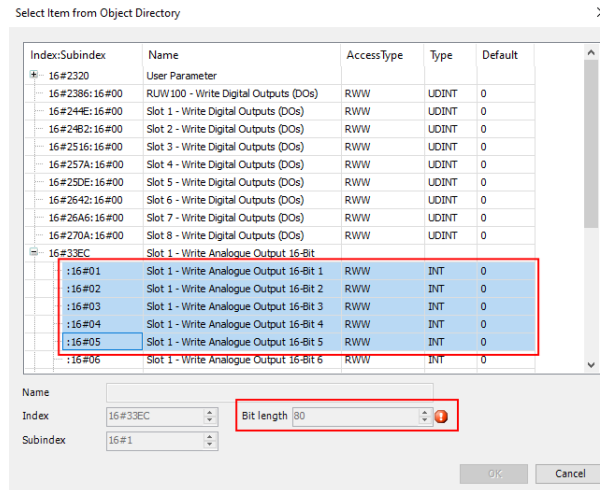


Figure 4.12: Error selecting objects to a PDO due to exceeding the 64-bit size.

To change the mapped objects of PDOs 1...4, you must first remove the objects using **Delete** or by left-clicking, as shown in Figure 4.13 on page 4-7.

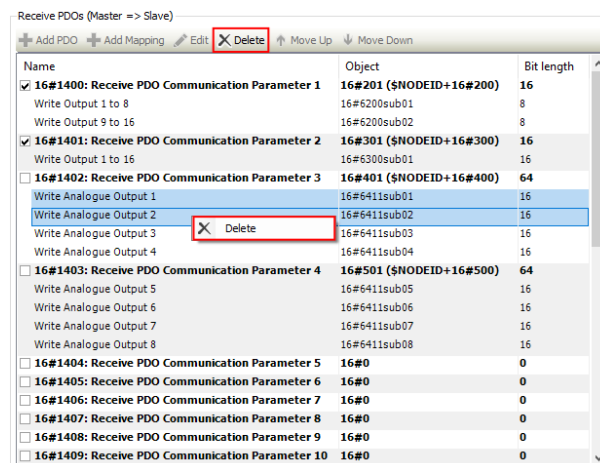


Figure 4.13: Deleting objects from a PDO.



NOTE!

PDOs can be deleted, but only those that are enabled through their checkbox will be used in the CANopen network.

If the PDO of the slave device does not appear in the interface, it is possible to add it using the **Add PDO** button as shown in Figure 4.14 on page 4-8. If all PDOs are already included in the list, this option will not be available, as is the case with the RUW100 when the device is added as a slave in the CAN interface in CODESYS.

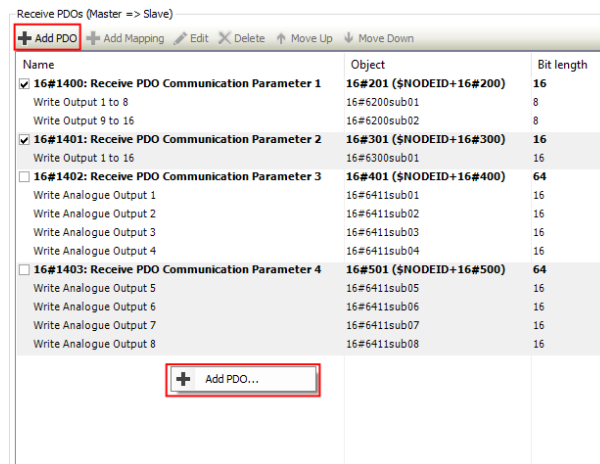


Figure 4.14: Adding a new reception PDO.

The alignment of default settings between a CANopen slave device and its EDS file is crucial for reliable communication. If the default values specified in the EDS do not match those in the device's firmware, it can lead to operational conflicts, such as PDOs not functioning correctly or communication failures. Checking the **write complete PDO configuration** checkbox as show in Figure 4.15 on page 4-8 in the **SDO** tab of the slave device ensures consistency between the device's behavior and its EDS, thereby minimizing potential issues and facilitating smooth data exchange in the network.

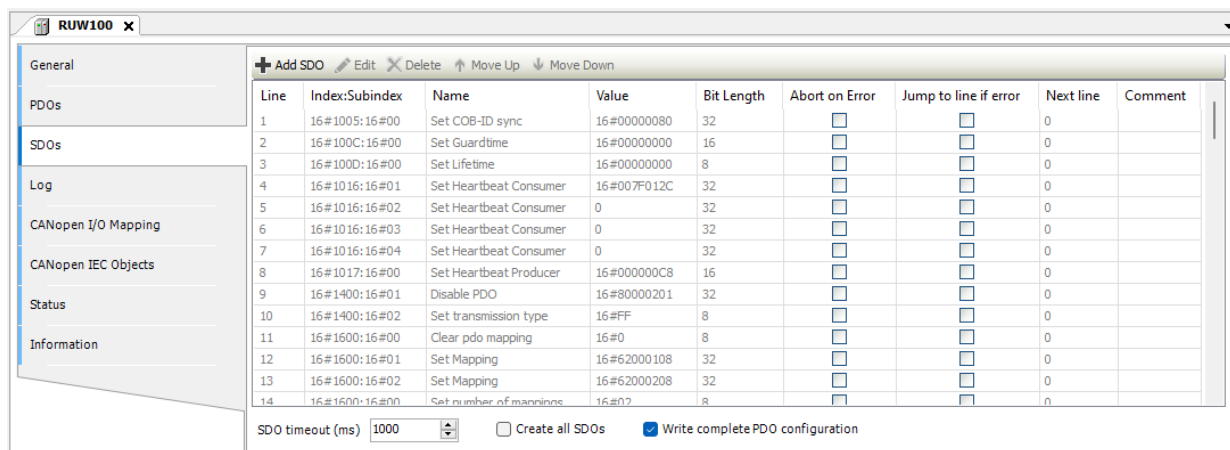


Figure 4.15: Activating Write complete PDO configuration option.

This way, the configurations to start the communication between the master and slave devices are complete.

5 MONITORING

In this chapter, some methods for monitoring the CANopen network variables and diagnosing communication errors in CODESYS are presented.

5.1 VARIABLE MONITORING

You can monitor the variables declared in the network through two methods. The first possibility would be to add them to the CODESYS program and monitor the values of the variables online. The second method only requires enabling **Always Update Variables** in the footer of the previous page, as shown in the red arrow in [Figure 5.1 on page 5-1](#).

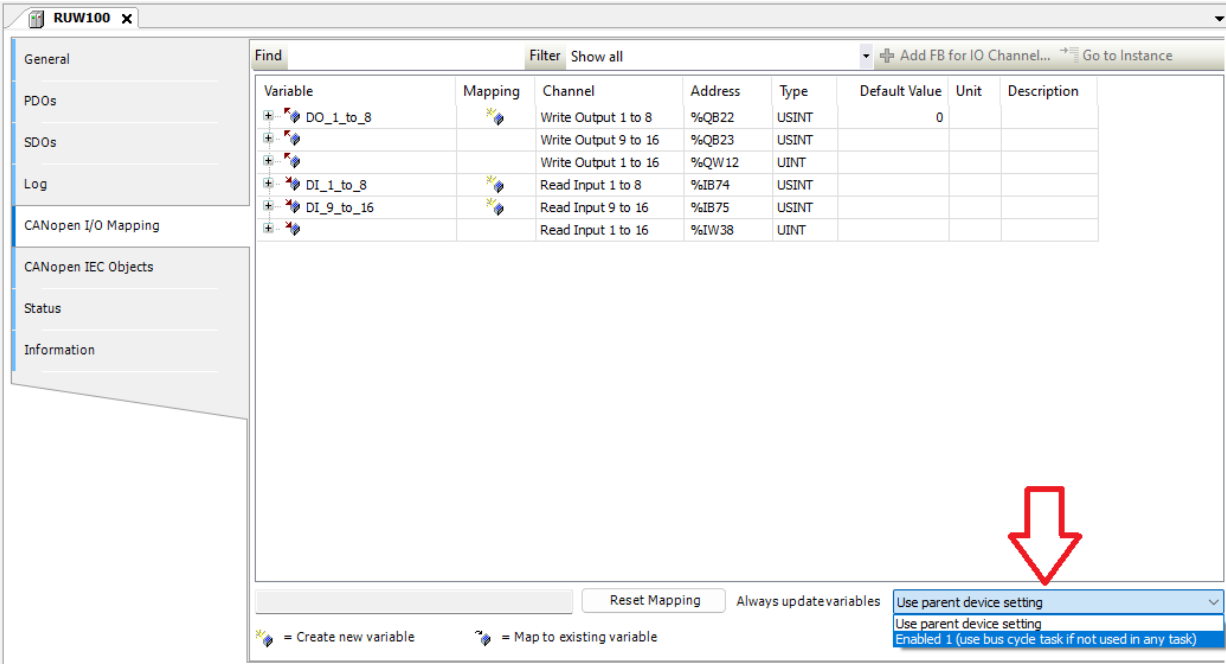


Figure 5.1: Configuration to monitor CANopen variables through the bus cycle.

[A Figure 5.2 on page 5-1](#) shows the CODESYS variable display screen for the network slave used through online monitoring.

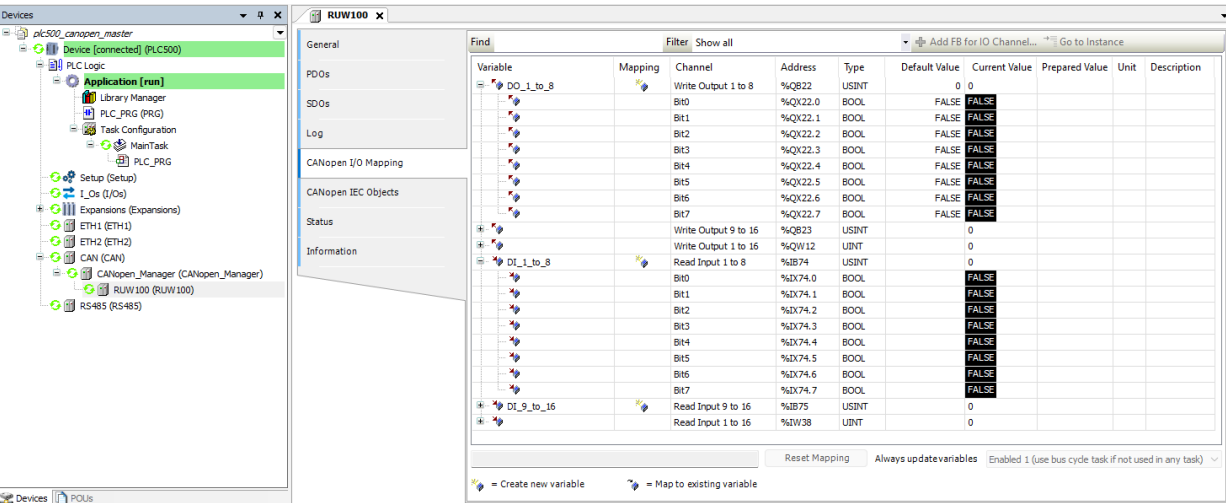


Figure 5.2: Monitoring variables in CODESYS.

5.2 COMMUNICATION ERRORS

Network status monitoring can be done in **Devices**, indicating the status of each communication stage and reporting the status (**Status**). When encountering connection problems, as shown in [Figure 5.3 on page 5-2](#), check if the cables are properly connected, what is the status of the **CAN LED** on the PLC410, PLC500, PLC500ED, PLC500MC, and then review the settings made.

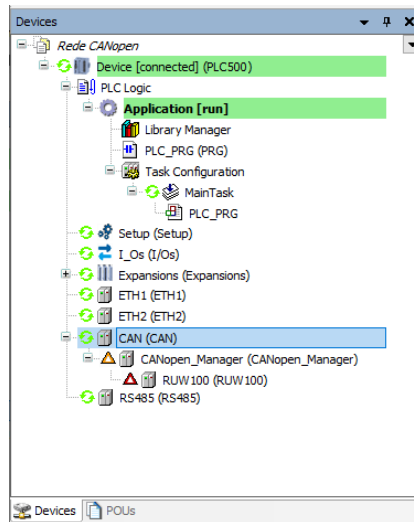


Figure 5.3: Indication of the error communicating with the slave device.

Connected to the PLC500, access the **Status** and **Log** tabs indicated in [Figure 5.4 on page 5-2](#). There, CODESYS will inform you of the problem it is encountering in order to perform the communication. In this case the cable was not properly connected to the slave device.

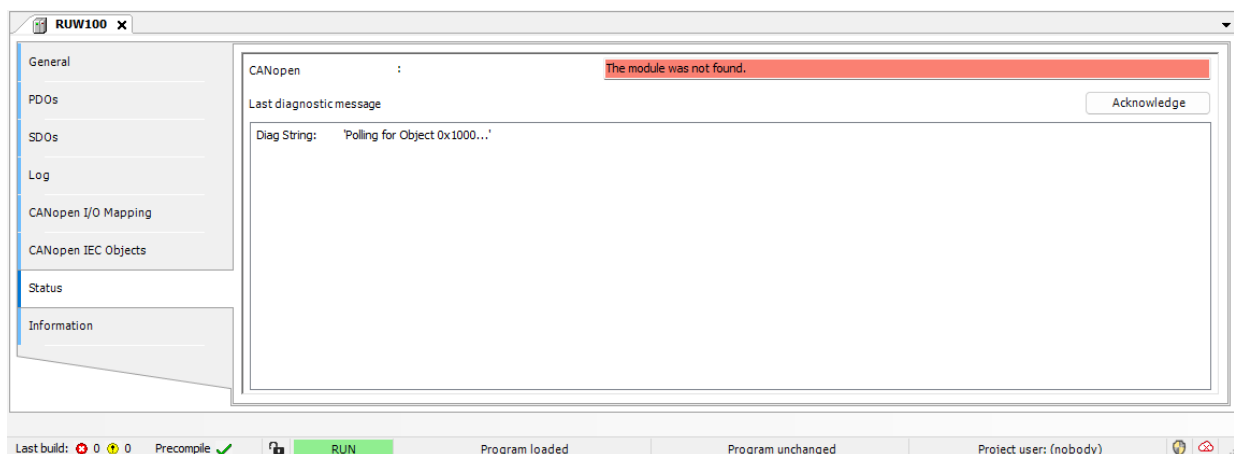


Figure 5.4: Status and Log pages contained in each of the communication interface items.

Reconnecting the cable and waiting a while restores network communication and the device icons turn green. An exclamation icon is displayed on devices indicating that an error has been declared, as shown in [Figure 5.5 on page 5-3](#).

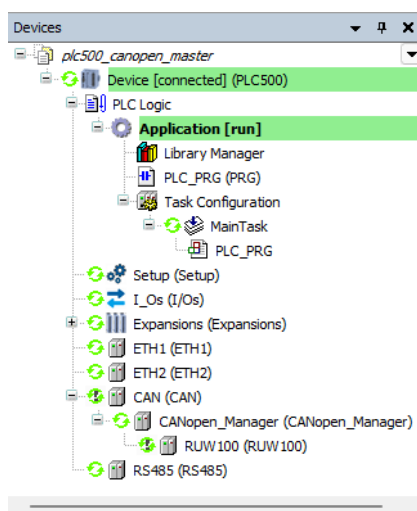


Figure 5.5: Indication of declared error on the CANopen network.

To remove the warning, simply click on **Acknowledge**, as shown in Figure 5.6 on page 5-3.

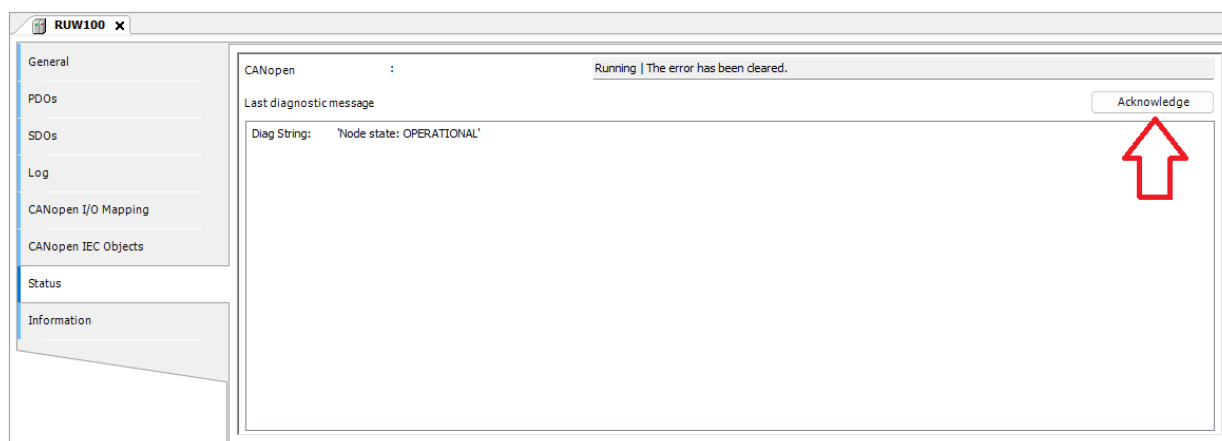


Figure 5.6: Recognizing declared error of CANopen Network.

After acknowledging the warning also in the **CAN** tab and good network functioning, the device icons will be represented as in Figure 5.7 on page 5-3.

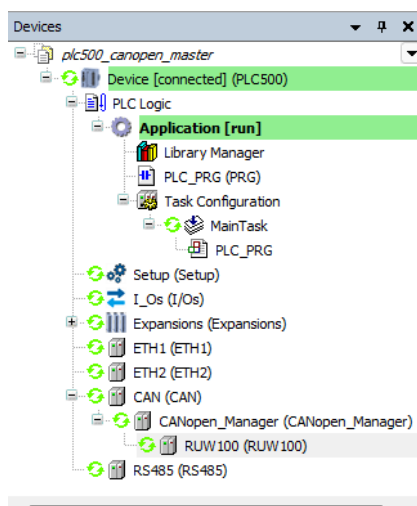


Figure 5.7: CANopen network running without errors and warnings between master and slave.

5.3 MONITORING USING FUNCTION BLOCKS

The CANopen network can be monitored using the Function Blocks available in the **CAA CiA 405** and **3S CANopenStack** libraries in CODESYS.

In the **3S CANopenStack** library, the **CANopenState** Function Block obtains the CANopen network device status at each call. [Figure 5.8 on page 5-4](#) shows the variable declaration and [Figure 5.9 on page 5-4](#) shows the structured text code for the ST_CANopenState application. In this example, **RUW100.CANopenState** is used to obtain the device state. It is worth noting that the name RUW100 can be replaced by any other CANopen device name present in the project tree.

ST_CANopenState
<pre>PROGRAM ST_CANopenState VAR RUW100_DeviceState_3SCOS: _3SCOS.DEVICE_STATE; END_VAR</pre>

Figure 5.8: Variable declaration for ST_CANopenState program.

ST_CANopenState - Structured text (ST)
<pre>// Gets state from RUW100 slave device RUW100_DeviceState_3SCOS := RUW100.CANopenState;</pre>

Figure 5.9: ST_CANopenState program in structured text.

In the CAA CiA 405 library the **GetState** Function Block obtains the state of the CANopen network device when enabled. [Figure 5.10 on page 5-4](#) shows the variable declaration and [Figure 5.11 on page 5-4](#) shows the structured text code for the ST_CANopen_GetState application.

ST_CANopen_GetState
<pre>PROGRAM ST_CANopen_GetState VAR GetState : CIA405.GET_STATE; xEnable : BOOL; // Set this variable to true to start the example code. Variable will be set to FALSE if GetState finished execution. State : CiA405.DEVICE_STATE; // Result: CANopen State. END_VAR</pre>

Figure 5.10: Variable declaration for ST_CANopen_GetState program.

ST_CANopen_GetState - Structured text (ST)
<pre>IF GetState.CONFIRM = TRUE THEN // GetState successful. Output STATE contains result. State := GetState.STATE; xEnable := FALSE; ELSIF GetState.ERROR <> CIA405.CANOPEN_KERNEL_ERROR.CANOPEN_KERNEL_NO_ERROR THEN // an error occurred while processing GetState e.g. wrong NodeID or NetworkID. xEnable := FALSE; END_IF // GetState needs the CAN network and the device NodeID (input DEVICE). GetState (ENABLE := xEnable, NETWORK := CAN.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. TIMEOUT := 0, // no timeout for GetState needed because it is processed locally. DEVICE := RUW100.NodeID, // Note: If you want to get the state of the CANopenManager, you can use the CANopenManager NodeID or NodeID 0. // A CANopenManager goes to OPERATIONAL if all mandatory (= not optional) slaves are operational.);</pre>

Figure 5.11: ST_CANopen_GetState program in structured text.

The `DEVICE_STATE` variable is of type `ENUM` whose states are described in [Table 5.1 on page 5-5](#).

Table 5.1: Description of the states of the `DEVICE_STATE` variable.

Name	Value	Description
INIT	0	Device is in boot up phase.
RESET_COMM	1	Device performs a reset communication.
RESET_APP	2	Device performs a reset application.
PRE_OPERATIONAL	3	Device is in PREOPERATIONAL state (default state after boot up).
STOPPED	4	Device is in STOPPED state.
OPERATIONAL	5	Device is in OPERATIONAL state. I/O messaging is active.
UNKNOWN	6	Device state is unknown. For example if no guarding mechanism is active.
NOT_AVAIL	7	Device was not found on bus.

**NOTE!**

Other function blocks and methods are available in the CAA CiA 405 and 3S CANopenStack libraries, see the documentation available in the **Library Manager** or in the online help available at [CODESYS Online Help](#).



BRAZIL

WEG DRIVES & CONTROLS - AUTOMAÇÃO LTDA.

Av. Prefeito Waldemar Grubba, 3000

89256-900 - Jaraguá do Sul - SC

Phone: 55 (47) 3276-4000

Fax: 55 (47) 3276-4060

www.weg.net/br