# User Manual
# Nexto XF

MU218600 Rev. A

April 23, 2026

www.altus.com.br

According to current legislation in Brazil, the Consumer Defense Code, we are giving the following information to clients who use our products, regarding personal safety and premises.

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. This can even put human lives at risk. The user should consider the possible consequences of the defects and should provide additional external installations for safety reasons. This concern is higher when in initial commissioning and testing.

The equipment manufactured by Altus does not directly expose the environment to hazards, since they do not issue any kind of pollutant during their use. However, concerning the disposal of equipment, it is important to point out that built-in electronics may contain materials which are harmful to nature when improperly discarded. Therefore, it is recommended that whenever discarding this type of product, it should be forwarded to recycling plants, which guarantee proper waste management.

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use. The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user training and improve experience with the products and their features.

Altus warrants its equipment as described in General Conditions of Supply, attached to the commercial proposals.

Altus guarantees that their equipment works in accordance with the clear instructions contained in their manuals and/or technical characteristics, not guaranteeing the success of any particular type of application of the equipment.

Altus does not acknowledge any other guarantee, express or implied, mainly when end customers are dealing with third-party suppliers. The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing. Altus is not responsible for supplying information about its equipment without formal request. These products can use EtherCAT® technology (www.ethercat.org).

## COPYRIGHTS



## OPEN SOURCE SOFTWARE NOTICE

To obtain the source code under GPL, LGPL, MPL and other open source licenses, that is contained in this product, please contact opensource@altus.com.br. In addition to the source code, all referred license terms, warranty disclaimers and copyright notices may be disclosed under request.

# Contents

# 1. Introduction

The Nexto XF is a powerful Programmable Logic Controller (PLC), belonging to the family of controllers and I/O modules of the Nexto Series. The Nexto XF provides high-speed processing power in a compact design with embedded I/O, easily expandable with external I/O modules. There are several options available, allowing you to choose the best solution regardless of the complexity of the application.

This product portfolio targets small, medium and distributed control systems, offering models with a range of options of embedded I/O points that includes digital and analog inputs and outputs concentrated in a single controller. For additional I/O needs, the system can be easily expanded through expansion modules coupled to the high-speed bus (refer to Related Products section). Additionally, the number of I/O points can be further increased through remote (distributed) I/O devices communicating via protocols such as CANopen, EtherNet/IP, PROFINET, EtherCAT, and MODBUS.

The Nexto XF can be used in applications such as infrastructure, marine, building and factory automation (food, textile, packaging, general machinery), water and wastewater treatment and numerous OEM solutions. Integrated solutions of firewall and VPN add security to ensure data integrity and reduce risks associated to cybersecurity threats. Additionally, the controller is an ideal solution to complement large applications along with the Nexto Series portfolio, extending the range of applications using the same technology and engineering environment. This is a significant advantage for OEMs and system integrators with a wide range of applications needs and sizes.


Figure 1: Nexto XF325

## 1.1. Documents Related to this Manual

In order to obtain additional information regarding the Nexto Series, other documents (manuals and technical features) besides this one, may be accessed. These documents are available in its last version on the site https://www.altusautomation.com.

Each product has a document designed by Technical Features (CE), where the product features are described. Furthermore, the product may have Utilization Manuals (the manuals codes are listed in the CE).

The following documents are recommended as additional sources of information:

| Code | Description | Language |
|---|---|---|
| **CE118100** | Nexto XF Technical Characteristics | English |
| **CT118100** | Características Técnicas Nexto XF | Portuguese |
| **MU214000** | Manual de Utilização Série Nexto | Portuguese |
| **MU214600** | Nexto Series User Manual | English |
| **MU214622** | LibSparkPlug User Manual | English |
| **MU218600** | Nexto XF User Manual | English |
| **MU218000** | Manual de Utilização Nexto XF | Portuguese |
| **MU299611** | Mastertool X User Manual | English |
| **MU299049** | Manual de Utilização Mastertool X | Portuguese |
| **MP399609** | IEC 61131 Programming Manual | English |
| **MP399048** | Manual de Programação IEC 61131 | Portuguese |
| **MU214606** | MQTT User Manual | English |
| **MU214609** | OPC UA Server for Altus Controllers User Manual | English |
| **MU214610** | PID - Advanced Control Functions User Manual | English |
| **MU214621** | Nexto Series PROFINET Manual | English |
| **NAP169** | RSTP in Nexto CPUs | English |

Table 1: Related Documents

## 1.2. Visual Inspection

Before resuming the installation process, it is advised to carefully visually inspect the equipment, verifying the existence of transport damage. Verify that all parts requested are in good condition. In case of damages, inform the transport company or Altus distributor closest to you.

**CAUTION**

Before taking the modules off the case, it is important to discharge any possible static energy accumulated in the body. For that, touch (with bare hands) on any metallic grounded surface before handling the modules. Such procedure guarantees that the module static energy limits are not exceeded.

It's important to register each received equipment serial number, as well as software revisions, in case they exist. This information is necessary, in case the Altus Technical Support is contacted.

## 1.3. Technical Support

To contact Altus Technical Support, send an email to support@altusautomation.com or visit our website at www.altusautomation.com/support. If the equipment is already installed, please have the following information available when requesting assistance:

- Model of the equipment used and the installed system configuration
- Product serial number
- Equipment revision and executive software version, indicated on the label attached to the side of the product
- CPU operating mode information, obtained using Mastertool
- Application software content, obtained using Mastertool
- Mastertool version used

## 1.4. Warning Messages Used in this Manual

In this manual, the warning messages will be presented in the following formats and meanings:

**DANGER**

Reports potential hazard that, if not detected, may be harmful to people, materials, environment and production.

**CAUTION**

Reports configuration, application or installation details that must be taken into consideration to avoid any instance that may cause system failure and consequent impact.

**ATTENTION**

Identifies configuration, application and installation details aimed at achieving maximum operational performance of the system.

# 2. Technical Description

This chapter presents all technical features of Nexto XF controllers.

## 2.1. Panels and Connections

The following figure shows the XF325 front panel:

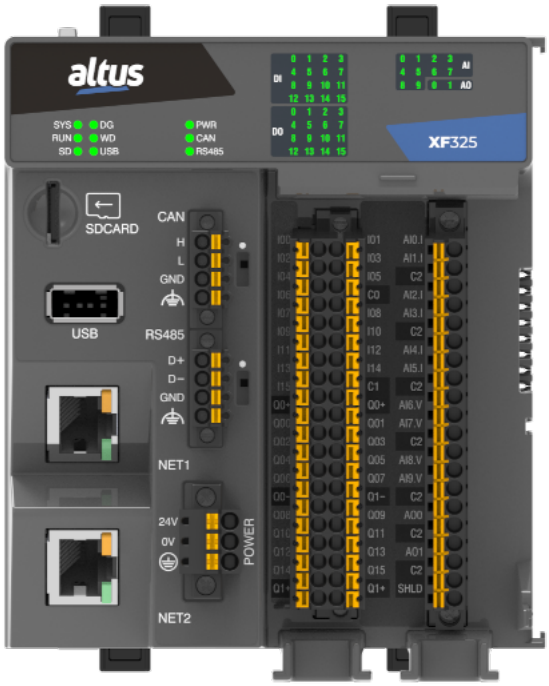



Figure 2: XF325 front panel

The front panel contains the identification of the overall system status, I/O and communication interfaces available on Nexto XF controllers. The I/O interfaces have one LED for each point to indicate the logic state, while the communication interfaces have one LED each to indicate activity. The availability of these interfaces on each model is described on next section. The following table shows the LEDs description. For further information regarding the LEDs status and meaning, see Maintenance chapter.

| LED | Description |
|---|---|
| **SYS** | PLC Status |
| **DG** | Diagnostic Status |
| **RUN** | Running application status |
| **WD** | Watchdog status |
| **SD** | SD Card interface status |
| **USB** | USB interface status |
| **PWR** | Internal power supply status |
| **CAN** | CAN interface status |
| **RS485** | RS-485 serial interface status |
| **DI 0-6** | Standard digital inputs logic state |
| **DI 7-15** | Standard/Fast digital inputs logic state |
| **DO 0-7** | Standard digital outputs logic state |
| **DO 8-15** | Standard/Fast digital outputs logic state |

| LED | Description |
|---|---|
| **AI 0-9** | Analog input logic state<br>(The respective LED turns on when the IO is configured) |
| **AO 0-1** | Analog output logic state<br>(The respective LED turns on when the IO is configured) |
| **ETH** | Ethernet interface link and activity status<br>(Mounted on the RJ45 connector jack) |

Table 2: LEDs Description

## 2.2. Product Features

### 2.2.1. General Features

| | XF300-B | XF300 | XF315 | XF325 | XF325-W |
|---|---|---|---|---|---|
| **Digital inputs** | 16 (9 fast) | | | | |
| **Digital outputs** | 16 (8 fast) | | | | |
| **Max. number of high-speed counters** | 3 | | | | |
| **Max. number of external interruptions** | 9 | | | | |
| **Max. number of PTO outputs** | 3 | | | | |
| **Max. number of VFO/PWM outputs** | 8 | | | | |
| **Analog voltage inputs** | - | | 4 | 4 | 4 |
| **Analog current inputs** | - | | 6 | 6 | 6 |
| **Analog V/I outputs** | - | | - | 2 | 2 |
| **Ethernet TCP / IP interface** | 2 | | | | |
| **Ethernet interface redundancy support** | Yes | | | | |
| **RS-485 serial interface** | 1 | | | | |
| **CAN interface** | 1 | | | | |
| **USB Host port** | 1 | | | | |
| **Max number of I/O expansion modules supported** | 10 | | | | |
| **User web pages (Webvisu)** | No | | | | Yes, 2048 tags |
| **Docker engine** | No | Yes | | | |
| **I/O Channels** | 128 | 512 | | | |
| **Communication Tags** | 512 | 2048 | | | 4096 |
| **Mapped points** | 20480 | | | | |
| **Communication Protocols** | See table Protocols | | | | |
| **FTP** | Yes | | | | |
| **Firewall** | Yes | | | | |
| **VPN** | Yes | | | | |
| **Programming languages** | Structured Text (ST)<br>Ladder Diagram (LD)<br>Sequential Function Chart (SFC)<br>Function Block Diagram (FBD)<br>Continuous Function Chart (CFC) | | | | |
| **Online changes** | Yes | | | | |
| **Maximum number of tasks** | 16 | | | | |
| **Watchdog** | Yes | | | | |
| **Status and diagnosis** | LEDs, web page and internal CPU memory | | | | |
| **Real-time clock (RTC)** | Yes<br>1 ms resolution, max. variation of 3 seconds per day,<br>retention time of 14 days. | | | | |
| **Isolation** | | | | | |
| **Protective earth ⏚ to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) | | | | |
| **Logic/USB to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) | | | | |
| **Power supply to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) | | | | |
| **NET1 to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) | | | | |
| **NET2 to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) | | | | |
| **CAN to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) | | | | |

| | |
|---|---|
| **RS-485 to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) |
| **Analog I/O to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) |
| **Digital inputs to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) |
| **Digital inputs group I0x to I1x** | 1000 Vdc / 1 minute (700 Vac / 1 minute) |
| **Digital outputs to all** | 1000 Vdc / 1 minute (700 Vac / 1 minute) |
| **Maximum power dissipation** | 10 W |
| **Maximum wire size** | 0.5 $mm^2$ (20 AWG) with ferrule<br>1.5 $mm^2$ (16 AWG) without ferrule |
| **Minimum wire temperature rating** | 75 °C |
| **Wire material** | Copper only |
| **IP level** | IP 20 |
| **Conformal coating** | Yes |
| **Operating temperature** | -20 to 60 °C |
| **Storage temperature** | -40 to 70 °C |
| **Operating and storage relative humidity** | 10% to 95%, non-condensing |
| **Vibration resistance (IEC 60068-2-6, sinus)** | 3.5 mm from 5 to 8.4 Hz<br>1 G from 8.4 to 500 Hz<br>10 sweeps each axis, 1 octave per minute |
| **Shock resistance (IEC 60068-2-27, half-sine)** | 15 G for 11 ms, 6 shocks in each of 3 axis |
| **Product dimensions (W x H x D)** | 112.2 x 122.8 x 123.0 mm<br>(102.8 x 122.8 x 110.8 mm without I/O termination and without cable cover) |
| **Package dimensions (W x H x D)** | 128 x 138 x 138 mm |
| **Weight** | 513.5 g |
| **Weight with package** | 675.1 g |

Table 3: General Features

**Notes:**

**I/O Channels:** Maximum number of individual input and output channels used by the application, listed in *I/O Mapping* tab of each device editor window. A channel may contain multiple I/O points (especially for digital I/O), so the count of channels is less than the number of I/O points. The number of I/O Channels used by the application is informed by Mastertool on *Messages* window within the category *Software Metrics*.

**Communication Tags:** Maximum number of variables explicitly designated for data exchange between the controller and external systems using OPC UA, OPC DA and CODESYS Protocol - ARTI, configured on *Communication Manager* or *Symbol Configuration*. The number of *Communication Tags* used by the application is informed by Mastertool on *Messages* window within the category *Software Metrics*.

**Webvisu Tags:** Maximum number of application variables used in web visualization dashboard. The number of *Webvisu Tags* used is informed by Mastertool on *Messages* window within the category *Software Metrics*.

**Mapped points:** Maximum number of communication points mapped specifically on protocols MODBUS, IEC 104, DNP3 and IEC 61850 (if supported). Each mapping can contain one or more mapped points, depending on the data size. This varies depending on whether simple variables or ARRAY-type variables are used. Each simple variable, as well as each index of an ARRAY, is counted as a mapped point, even if it occupies more than one address in the driver. For example, a simple DWORD-type variable mapped in the MODBUS protocol will be counted as a single point, even though it occupies two consecutive addresses/registers in the driver.

**Isolation:** The term *Logic* refers to the processing circuit that is electrically isolated from the interfaces, with the exception of the USB interface.

**Conformal coating:** Conformal coating protects the electronic components inside the product from moisture, dust and other harsh elements to electronic circuits.

**Number of I/O modules:** The maximum number of I/O modules supported by the CPU depends on the power consumed on the bus. Use Mastertool to configure the number of modules and scale the power consumption on the bus.

### 2.2.2. Standards and Certifications

| Standards and Certifications | |
|---|---|
| **IEC** | 61131-2: Industrial-process measurement and control - Programmable controllers - Part 2: Equipment requirements and tests<br><br>61131-3: Programmable controllers - Part 3: Programming languages |
| | Certification process in progress |
| | Certification process in progress |
| | Certification process in progress |
| | Ordinary and Hazardous Locations (C1D2): cULus LISTED<br>Certification process in progress |

Table 4: Standards and Certifications

### 2.2.3. Memory

| | XF300-B | XF300 | XF315 | XF325 | XF325-W |
|---|---|---|---|---|---|
| **Addressable input variables memory (%I)** | 64 KB | | | | |
| **Addressable output variables memory (%Q)** | 64 KB | | | | |
| **Direct representation variable memory (%M)** | 32 KB | | | | |
| **Symbolic variable memory** | 12 MB | | | | 32 MB |
| **Program memory** | 32 MB | | | | |
| **Total memory**<br>**Program memory +**<br>**Source code memory (backup) +**<br>**Webvisu files memory** | 256 MB | | | | |
| **Retain/persistent memory** | 128 KB | | | | |
| **User files memory**<br>**Memory of the CPU +**<br>**Docker Memory** | 2.5GB | | | | |

Table 5: Memory

**Notes:**

**Addressable input variables memory (%I):** Area where the addressable input variables are stored. Addressable variables means that the variables can be accessed directly using the desired address. For instance: %IB0, %IW100. Addressable input variables can be used for mapping digital or analog input points. As reference, 8 digital inputs can be represented per byte and one analog input point can be represented per two bytes.

**Total addressable output variables memory (%Q):** Area where the addressable output variables are stored. Addressable variables means that the variables can be accessed directly using the desired address. For instance: %QB0, %QW100. Addressable output variables can be used for mapping digital or analog output points. As reference, 8 digital outputs can be represented per byte and one analog output point can be represented per two bytes. The addressable output variables can be configured as retain, persistent or redundant variables, but the total size is not modified due to configuration.

**Addressable variables memory (%M):** Area where the addressable marker variables are stored. Addressable variables means that the variables can be accessed directly using the desired address. For instance: %MB0, %MW100.

**Symbolic variables memory:** Area where the symbolic variables are allocated. Symbolic variables are IEC variables created in POUs and GVLs during application development, which are not addressed directly in memory. Symbolic variables can be defined as retentive or persistent, in which case the memory areas of retentive symbolic variables or memory of persistent symbolic variables respectively will be used. The PLC system allocates variables in this area, so the space available for the allocation of variables created by the user is lower than that reported in the table. The occupation of the system variables depends on the characteristics of the project (number of modules, drivers, etc...), so it is recommended to observe the space available in the compilation messages of the Mastertool tool.

**Persistent or Retain symbolic variables memory:** Area where are allocated the retentive symbolic variables. The retentive data keep its respective values even after a CPU's cycle of power down and power up. The persistent data keep its respective values even after the download of a new application in the CPU.

**ATTENTION**

The declaration and use of symbolic persistent variables should be performed exclusively through the *Persistent Vars* object, which may be included in the project through the tree view in *Application -> Add Object -> Persistent Variables*. It should not be used to *VAR PERSISTENT* expression in the declaration of field variables of POUs.

The table below shows the behavior of retentive and persistent variables for different situations in which "-" means the value is lost and "X" means the value is kept.

| Command | Symbolic Variable | Retain variable | Persistent variable |
|---|---|---|---|
| **Power cycle** | - | X | X |
| **Reset warm** | - | X | X |
| **Reset cold** | - | - | X |
| **Reset Origin** | - | - | - |
| **Download** | - | - | X |
| **Online change** | X | X | X |
| **Clean All** | - | - | X |
| **Runtime Reset** | - | - | - |

Table 6: Variables behavior after the event

In the case of the Clean All command, if the application has been modified in such a way that persistent variables have been removed, inserted at the beginning of the list, or have had their type changed, the value of these variables will be lost (a warning is issued by the Mastertool software when performing the download). Therefore, it is recommended that changes to the persistent variables GVL only involve the inclusion of new variables at the end of the list.

**Program memory:** Memory area that corresponds to the maximum size allowed for the user application.

**Source code memory (backup):** Memory area used as a project backup, meaning that if the user wishes to import their project, the Mastertool software will retrieve the necessary information from this area. It is important to emphasize that the user must be careful not to forget to update the project saved as a backup every time they send the application, preventing information from being lost.

**User files memory:** This memory area is intended for storing files, such as: doc, pdf, images, among others; essentially, it allows data recording as if it were a memory card. Further information can be found in the User Files Memory section.

### 2.2.4. Protocols

The following table describes the limits of types and quantities (number of instances) of each protocol. Please note that there are other operational limits related to communication capabilities (maximum number of *I/O Channels*, *Communication Tags* and *Mapped Points*), which are informed on General Features table.

| | Interface | XF300-B | XF300 | XF315 | XF325 | XF325-W |
|---|---|---|---|---|---|---|
| **EtherCAT Master<br>PROFINET Controller<br>EtherNet/IP Scanner** | NET 1 / NET 2 | ✗ | Up to 1 instance | | | |
| **EtherNet/IP Adapter** | NET 1 / NET 2 | Up to 1 instance | Up to 2 instances | | | |
| **PROFINET Device** | - | ✗ | ✗ | | | |
| **CANopen Master<br>SAE J1939** | CAN | Up to 1 instance | Up to 1 instance | | | |
| **CAN low level** | CAN | ✓ | ✓ | | | |
| **CANopen Slave** | - | ✗ | ✗ | | | |
| **Open Protocol** | COM 1 / USB | ✓ | ✓ | | | |
| **Symbolic MODBUS RTU Master<br>Symbolic MODBUS RTU Slave** | COM 1 | Up to 1 instance | Up to 1 instance | | | |
| **Symbolic MODBUS TCP Client<br>Symbolic MODBUS TCP Server<br>Symbolic MODBUS RTU over TCP Client<br>Symbolic MODBUS RTU over TCP Server** | NET 1 / NET 2 | Up to 8 instances | Up to 8 instances | | | |
| **IEC 60870-5-104 Client** | - | ✗ | ✗ | | | |
| **IEC 60870-5-104 Server** | - | ✗ | ✗ | | | |
| **DNP3 Client** | - | ✗ | ✗ | | | |
| **DNP3 Server** | - | ✗ | ✗ | | | |
| **OPC DA Server** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **OPC UA Server** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **SNMP Agent** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **MQTT Client** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **Sparkplug B** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **SNTP Client (clock synchronism)** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **OpenVPN Client** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **OpenVPN Server** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **FTP Server** | NET 1 / NET 2 / USB | ✓ | ✓ | | | |
| **RSTP** | NET 1 / NET 2 | ✓ | ✓ | | | |
| **MRP** | NET 1 / NET 2 | ✓ | ✓ | | | |

Table 7: Protocols

**Notes:**

**USB:** The use of a Serial Converter, WiFi Adapter or 3G/4G Modem is necessary.

### 2.2.5. RS-485 Serial Interface

| | RS-485 |
|---|---|
| **Connector** | 4-pin terminal block (D+, D-, GND and shield) |
| **Physical interface** | RS-485 bus |
| **RS-485 max. transceivers** | 32 |
| **Termination** | Yes (through mechanical switch) |
| **Baud rate** | 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps |
| **Isolation**<br>**Logic to serial port**<br>**Serial port to protection earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 8: RS-485 Serial Interface Features

### 2.2.6. CAN Bus Interface

| | CAN |
|---|---|
| **Connector** | 4-pin terminal block (H, L, GND and shield) |
| **Physical interface** | CAN bus |
| **Supported standards** | CAN 2.0A 2.0B (11-bit and 29-bit identifiers) |
| **Max. number of nodes** | 64 |
| **Termination** | Yes (through mechanical switch) |
| **Baud rate** | 10, 20, 50, 100, 125, 250, 500, 800, 1000 kbit/s |
| **Isolation**<br>**Logic to CAN**<br>**CAN to protection earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 9: CAN Interface Features

### 2.2.7. USB Interface

| | USB |
|---|---|
| **Connector** | USB Type-A (female) |
| **Physical interface** | USB V2.0 |
| **Baud rate** | 1.5 Mbps (Low-Speed), 12 Mbps (Full-Speed) and 480 Mbps (High-Speed) |
| **Maximum current** | 500 mA |
| **Supported devices** | Mass storage<br>USB RS-232 Serial Converter<br>USB 3G/4G Modem<br>USB WiFi Adapter |
| **Isolation**<br>**Logic to USB**<br>**USB to protection earth** ⏚ | <br>Not isolated<br>1000 Vac / 1 minute |

Table 10: USB Interface Features

**Notes:**

**USB RS-232 Serial Converter:** See the list of supported devices on respective section USB to RS-232 Converter.

> **Attention:**
>
> The CPU supports the use of only one USB device at a time. Devices such as USB hubs are not supported.

### 2.2.8. Ethernet

| | Ethernet |
|---|---|
| **Interfaces** | NET1 and NET2 |
| **Connector** | Shielded female RJ45 |
| **Auto crossover** | Yes |
| **Maximum cable length** | 100 m |
| **Cable type** | UTP or ScTP, category 5 |
| **Baud rate** | 10/100 Mbps |
| **Physical layer** | 10/100 BASE-TX |
| **Data link layer** | LLC |
| **Network layer** | IP |
| **Transport layer** | TCP (Transmission Control Protocol)<br>UDP (User Datagram Protocol) |
| **Operation mode** | Single / NIC Teaming / Switch |
| **Diagnostics** | LED (Link/Activity) |
| **Isolation** | |
| **Ethernet interface to logic** | 1000 Vac / 1 minute |
| **Ethernet interface to ethernet interface** | 1000 Vac / 1 minute |
| **Ethernet interface to protection earth** ⏚ | 1000 Vac / 1 minute |

Table 11: Ethernet Interface Features

### 2.2.9. Memory Card Interface

The memory card can be used for different data to be stored such as user logs, project documentation and source files.

| | Memory Card |
|---|---|
| **Maximum Capacity** | 32 Gbytes |
| **Minimum Capacity** | 2 Gbytes |
| **Type** | MicroSD |
| **File System** | FAT32 |
| **Remove card safely** | Yes (via button and web page) |

Table 12: Memory Card Interface Features

**Notes:**

**Maximum Capacity:** The memory card capacity must be less than or equal to this limit for correct operation on Nexto CPU, otherwise the Nexto CPU may not detect the memory card or even present problems during data transfer.

**Minimum Capacity:** The memory card capacity must be greater than or equal to this limit for correct operation on Nexto CPU, otherwise the Nexto CPU may not detect the memory card or even present problems during data transfer.

**File system:** It is recommended to format the memory card using the PLC Web Page.

### 2.2.10. Power Supply

| | **Power supply** |
|---|---|
| **Nominal input voltage** | 24 Vdc |
| **Input voltage** | 18 to 30 Vdc |
| **Maximum input current** | 1.4 A at 18 Vdc |
| **Maximum in-rush current** | 15 A for 1 ms |
| **Maximum interruption time** | 10 ms at 18 Vdc |
| **Number of I/O modules supported by integrated power supply** | 10 |
| **Maximum Current on XF Bus** | 2.2 A |
| **Input protections** | Polarity reversal<br>Overvoltage<br>Short-circuit (fuse)<br>Overload and short-circuit on the I/O module bus |
| **Isolation**<br>**Input to logic**<br>**Input to protective earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 13: Power Supply Features

### 2.2.11. Digital Inputs

| | **Digital inputs** |
|---|---|
| **Number of inputs** | 7 |
| **Type of inputs** | Opto-isolated, sink/source |
| **Connector identification** | I00 to I06 |
| **Input voltage** | 24 Vdc<br>15 to 30 Vdc for logic level 1<br>0 to 5 Vdc for logic level 0 |
| **Input impedance** | 4.95 kΩ |
| **Max. input current** | 6.2 mA at 30 Vdc |
| **Input status indication** | Yes |
| **Update time** | 2 ms |
| **Input filter** | Disabled or configurable by software from 2 ms to 255 ms |
| **Isolation**<br>**Input to logic**<br>**Input to protective earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 14: Digital Inputs Features

**Note:**

**Input Filter:** The filter sampling is performed in the MainTask, so it is recommended to use values that are multiples of the task interval.

### 2.2.12. Fast Inputs

| | Fast digital inputs |
|---|---|
| **Number of fast inputs** | 9 |
| **Type of fast inputs** | Opto-isolated sink/source |
| **Maximum number of fast counters** | 3 (Increment/Decrement, A-pulse, B-direction, and Z-zero) |
| **Maximum number of encoders** | 3 (Quadrature A, B, and Z) |
| **Max. number of external interrupts** | 9 (Rise, Fall or Both) |
| **Connector identification** | I07 to I15 |
| **Input voltage** | 24 Vdc<br>15 to 30 Vdc for logic level 1<br>0 to 5 Vdc for logic level 0 |
| **Input impedance** | 3.9 kΩ |
| **Max. input current** | 7.3 mA at 30 Vdc |
| **Configuration mode** | **Modes for 1 input:**<br>Standard digital input<br>External interrupt<br>**Modes for 3 inputs:**<br>Counter 0: Quadrature or Up/Down (I07, I08, and I09)<br>Counter 1: Quadrature or Up/Down (I10, I11, and I12)<br>Counter 2: Quadrature or Up/Down (I13, I14, and I15) |
| **Direction control for counting** | Only through hardware |
| **Edge detection for counting input** | Rising edge, active at logic level 1 – default (support alternative settings) |
| **Data format** | 32-bit signed integers |
| **Operation limit** | From -2,147,483,648 to 2,147,483,647 |
| **Max. input frequency** | 250 kHz |
| **Min. pulse width @24Vdc** | 2 $\mu$s |
| **Isolation**<br>**Input to logic**<br>**Input to protective earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 15: Fast Digital Inputs Features

### 2.2.13. Digital Outputs

| | Digital outputs |
|---|---|
| **Number of outputs** | 8 |
| **Type of outputs** | Transistor switch, opto-isolated, source |
| **Connector identification** | Q00 to Q07 |
| **Max. current** | 1.5 A per output (single - unprotected)<br>6 A total |
| **Leakage current** | 35 $\mu$A |
| **Output resistance** | 100 m$\Omega$ |
| **External power supply** | 19.2 to 30 Vdc |
| **Switching time** | 3 $\mu$s - off-to-on transition<br>50 $\mu$s - on-to-off transition (400 $\Omega$ load at 24Vdc) |
| **Max. switching frequency** | 250 Hz |
| **Configurable parameters** | Yes |
| **Output status indication** | Yes (LED) |
| **Output protections** | Yes (against voltage surges) |
| **Isolation**<br>**Output to logic**<br>**Output to protective earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 16: Digital Outputs Features

**Notes:**
**Switching time:** The time required to turn off an output depends on the applied load.

**CAUTION**

The outputs are not protect for overcurrent or short circuit events.

### 2.2.14. Fast Outputs

| | Fast digital outputs |
|---|---|
| **Number of outputs** | 8 |
| **Output type** | Transistor switch, opto-isolated, source |
| **Connector identification** | Q08 to Q15 |
| **Max. number of PTO outputs** | 3 |
| **Max. number of VFO/PWM outputs** | 8 (if not using PTO)<br>7 (when using 1 PTO)<br>6 (when using 2 PTOs)<br>5 (when using 3 PTOs) |
| **Maximum current** | 3 to 500 Hz: 1.5A per output (6.0 A total)<br>500 Hz to 250 kHz: 0.5 A per output |
| **Pulse generation minimum frequency** | 3 Hz from 60 mA |
| **Pulse generation maximum frequency** | 250 kHz from 60 mA |

| | Fast digital outputs | |
|---|---|---|
| **Minimum pulse width at 24 Vdc** | Minimum load | Positive pulse |
| | 400 Ω | 240 ns |
| **Switching Time** | 0.2 $\mu$s - off-to-on transition<br>0.8 $\mu$s - on-to-off transition (400 Ω load at 24Vdc) | |
| **Output status indication** | Yes (LED) | |
| **Protections** | Voltage surge (TVS) | |
| **Operating voltage** | 19.2 to 30 Vdc | |
| **Output impedance** | 200 mΩ | |
| **Output modes** | Standard output<br>VFO/PWM<br>PTO (only Q08, Q09, and Q14) | |
| **Software managed functions** | PTO | VFO/PWM |
| | Writing the number of pulses to be generated;<br>Writing the number of acceleration and deceleration pulses;<br>Start/end outputs operation;<br>Fast outputs diagnostics;<br>Fast outputs monitoring state. | Writing the frequency value to be generated (3 Hz to 250 kHz, +/-1%);<br>Writing of outputs duty cycle (1% to 100%);<br>Start/end of outputs operations;<br>Fast outputs diagnostics. |
| **Isolation**<br>**Output to logic**<br>**Output to protective earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute | |

Table 17: Fast Digital Outputs Features

### 2.2.15. Analog Inputs

| | Analog inputs |
|---|---|
| **Number of inputs** | 10 (with 6 current and 4 voltage) |
| **Type of input** | Dedicated as voltage or current, single-ended, configured individually |
| **Data format** | 16 bits in two's complement, left-justified |
| **Converter resolution** | 12 bits monotonicity guaranteed, no missing codes |
| **Conversion time** | 400 $\mu$s (all channels enabled) |
| **Channel status indication** | Yes (LED) |
| **Protections** | Yes, protection against surge voltage and polarity inversion |
| **Isolation**<br>**Channel to logic**<br>**Channel to protective earth** ⏚ | <br>1000 Vac / 1 minute<br>1000 Vac / 1 minute |

Table 18: Analog Inputs Features

| | Voltage input | | |
|---|---|---|---|
| **Input range** | Range | Engineering Scale | Resolution |
| | 0 to 10 Vdc | 0 to 30,000 | 2.52 mV |
| **Precision** | ±0.3 % of full scale @ 25 °C<br>± 0.010 % of full scale / °C | | |
| **Over scale** | 3 % of full scale | | |
| **Maximum input voltage** | 14 Vdc | | |
| **Input impedance** | 21 kΩ | | |
| **Configurable parameters** | Filters | | |
| **Low pass filter time constant** | 100 ms, 1 s, 10 s or disabled | | |

Table 19: Voltage Analog Inputs Features

| | Current input | | |
|---|---|---|---|
| **Input ranges** | Range | Engineering Scale | Resolution |
| | 0 to 20 mA | 0 to 30,000 | 5.03 $\mu$A |
| | 4 to 20 mA | 0 to 30,000 | 5.03 $\mu$A |
| **Precision** | ±0.3 % of full scale at 25 °C<br>± 0.015 % of full scale / °C | | |
| **Over scale** | 3 % of full scale | | |
| **Maximum input current** | 30 mA | | |
| **Input impedance** | 119 Ω | | |
| **Configurable parameters** | Filters, open loop value | | |
| **Low pass filter time constant** | 100 ms, 1 s, 10 s or disabled | | |

Table 20: Current Input Mode Features

### 2.2.16. Analog Outputs

| | Analog outputs |
|---|---|
| **Number of outputs** | 2 |
| **Output type** | Voltage or current output, individually configured |
| **Data format** | 16 bits in two's complement, left-justified |
| **Converter resolution** | 12 bits monotonicity guaranteed, no missing codes |
| **Update time** | 400 $\mu$s (all outputs enabled) |
| **Channel status indication** | Yes (LED) |
| **Protections** | Yes, protection against surge voltage and polarity inversion |
| **Isolation** | |
| **Channel to logic** | 1000 Vac / 1 minute |
| **Channel to protective earth** ⏚ | 1000 Vac / 1 minute |

Table 21: Analog Outputs Features

| | Voltage output mode | | |
|---|---|---|---|
| **Output ranges** | Range | Engineering Scale | Resolution |
| | 0 to 10 V | 0 to 30,000 | 2.52 mV |

| | Voltage output mode |
|---|---|
| **Precision** | ±0.3 % of full scale at 25 °C<br>± 0.010 % of full scale / °C |
| **Stabilization time** | 4 ms |
| **Overscale** | 3% of full scale |
| **Load impedance** | > 1 kΩ |
| **Configurable parameters** | Signal type per output (voltage or current) |

Table 22: Voltage Output Mode Features

| | Current output mode | | |
|---|---|---|---|
| **Output ranges** | Range | Engineering Scale | Resolution |
| | 0 to 20 mA | 0 to 30,000 | 5.03 $\mu$A |
| | 4 to 20 mA | 0 to 30,000 | 5.03 $\mu$A |
| **Precision** | ±0.3 % of full scale at 25 °C<br>± 0.015 % of full scale / °C | | |
| **Stabilization time** | 4 ms | | |
| **Overscale** | 3% of full scale | | |
| **Load impedance** | < 600 Ω | | |
| **Configurable parameters** | Signal type per output (voltage or current) | | |

Table 23: Current Output Mode Features

**Notes:**

**Output ranges:** When configured as 4 to 20 mA, the output can be set to values below 4 mA by assigning negative values to the output variable (-7,500 for 0 mA).

## 2.3. Compatibility with Other Products

To develop an application for Nexto Series CPUs, it is necessary to check the Mastertool version. The following table shows the minimum version required (where the controllers were introduced) and the respective firmware version at that time:

| Controller model | Mastertool X | Firmware version |
|---|---|---|
| **XF300-B, XF300, XF315, XF315 e XF325-W** | 4.0.0 | 1.14.88.2 |

Table 24: Product compatibility

Additionally, along the development roadmap of the prograaming tool, some features may be included (like special Function Blocks, etc...), which can introduce a requirement of minimum firmware version. During the download of the application, Mastertool checks the firmware version installed on the controller and, if it does not match the minimum requirement, will show a message requesting to update. The latest firmware version can be downloaded from Altus website, and it is fully compatible with previous applications.

## 2.4. Performance

The performance of Nexto XF controller relies on:

- Application Interval Time
- User Application Time
- Operational System Time
- Number of integrated I/O channels enabled

### 2.4.1. Interval Time

The application and I/O update are executed on a cyclic (periodic) task called MainTask. The interval time of this task can be configured from 1 to 100 ms. The time spent for these operations is called Cycle Time, and should always be smaller than the interval, because the free time is used for communication and other low priority tasks of the controller.

### 2.4.2. Application Times

The execution time of the application (cycle time) depends on the following variables:

- Integrated inputs read time
- Task execution time
- Integrated outputs write time

### 2.4.3. Time for Instructions Execution

The table below presents the necessary execution time for different instructions.

| Instruction | Language | Variable Type | Time ($\mu$s) |
|---|---|---|---|
| **1000 Contacts** | LD | BOOL | 2,1 |
| **1000 Divisions** | LD, ST | INT | 9,2 |
| | | REAL | 17,0 |
| **1000 Multiplications** | LD, ST | INT | 6,4 |
| | | REAL | 8,2 |
| **1000 Sums** | LD, ST | INT | 4,4 |
| | | REAL | 8,2 |

Table 25: Instruction Times

### 2.4.4. Initialization Times

The initialization time of Nexto XF controllers is approximately 30 s.

## 2.5. Physical Dimensions

Dimensions in mm.

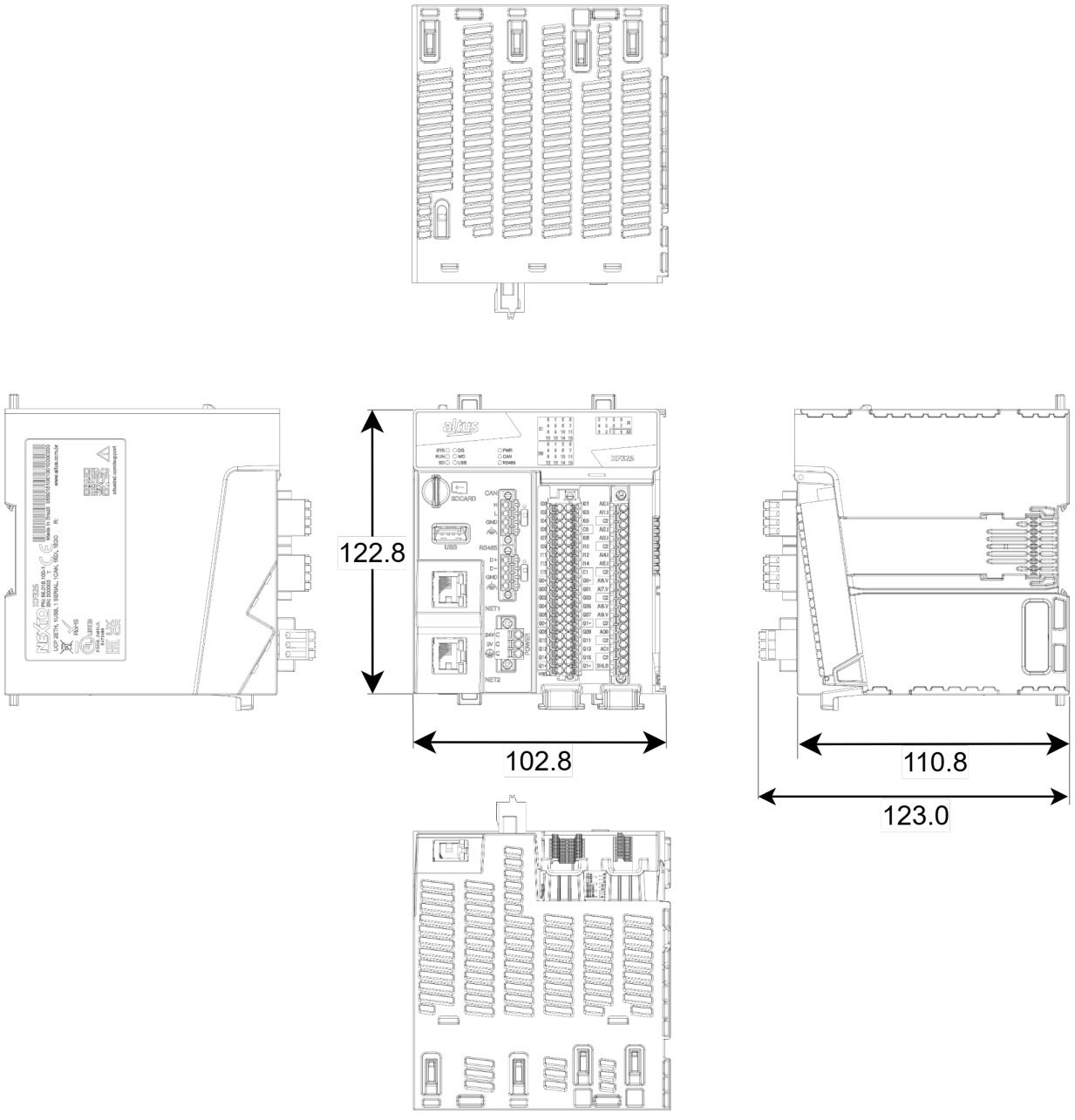



Figure 3: XF3xx Physical Dimensions

## 2.6. Purchase Data

### 2.6.1. Integrant Items

The product packaging includes the following items:

- PLC module with embedded I/O XF3xx
- 40-pin connector
- 20-pin connector (except for XF300 and XF300-B)
- 3-pin connector
- 2 x 4-pin connectors
- Nexto XF bus terminator
- CPU embedded I/O cable cover

### 2.6.2. Product Code

The following code should be used to purchase the product:

| Code | Description |
|---|---|
| **XF300-B** | High-speed compact PLC with 16 digital inputs, 16 transistor digital outputs, 2 Ethernet ports, 1 RS-485 port, 1 CAN port, USB Host, and microSD card. Support for basic protocols only. |
| **XF300** | High-speed compact PLC with 16 digital inputs, 16 transistor digital outputs, 2 Ethernet ports, 1 RS-485 port, 1 CAN port, USB Host, and microSD card. |
| **XF315** | High-speed compact CLP with 16 digital inputs, 16 transistor digital outputs, 6 current analog inputs, 4 voltage analog inputs, 2 Ethernet ports, 1 RS-485 port, 1 CAN port, USB Host, and microSD card. |
| **XF325** | High-speed compact PLC with 16 digital inputs, 16 transistor digital outputs, 6 current analog inputs, 4 voltage analog inputs, 2 analog outputs (V/I), 2 Ethernet ports, 1 RS-485 port, 1 CAN port, USB Host, and microSD card. |
| **XF325-W** | High-speed compact PLC with 16 digital inputs, 16 transistor digital outputs, 6 current analog inputs, 4 voltage analog inputs, 2 analog outputs (V/I), 2 Ethernet ports, 1 RS-485 port, 1 CAN port, USB Host, microSD card and user web page support. |

Table 26: Nexto XF Controller Models

## 2.7. Related Products

The following products must be purchased separately when necessary:

| Code | Description |
|---|---|
| **MT9000** | Mastertool X |
| **AL-2600** | RS-485 network branch and terminator |
| **AL-2306** | RS-485 cable for MODBUS or CAN network |
| **NX9101** | 32 GB microSD memory card with miniSD and SD adapters |
| **FBS-USB-232M9** | Universal USB-Serial converter cable / 2m |
| **XP900** | TP-Link nano Wireless 150 Mbps USB Adapter TL-WN725N (only available in Brazil) |
| **AMJG0808** | Simple cable RJ45-RJ45 2 m |
| **XF101** | 24Vdc 16 DI module |
| **XF201** | 24Vdc 16 DO transistor module |
| **XF600** | 6 AI voltage/current module 12bit |
| **XF610** | 8 AI thermocouple module |
| **XF620** | 8 AI RTD module |
| **XF700** | 4 AO voltage/current module 12bit |
| **XF900** | Nexto XF bus terminator |
| **XF901** | CPU embedded I/O cable cover |
| **XF902** | I/O module cable cover |

Table 27: Related Products

**Notes:**

**AL-2600:** This module is used for branch and termination of RS-485 networks. For each network node, an AL-2600 is required. The AL-2600 units at the ends of the network must be configured with termination, except when there is a device with active external termination, the rest must be configured without termination.

**AL-2306:** Two shielded twisted-pair cable without connectors, used for networks based on RS-485 or CAN.

**FBS-USB-232M9:** Cable for use as a USB-Serial converter on the USB interface of XF controllers.

**AMJG0808:** Cable for programming the CPUs.

# 3. Installation

This chapter presents the necessary proceedings for the physical installation, as well as the care that should be taken with other installation within the panel where the controller is being installed.

**CAUTION**

If the equipment is used in a manner not specified by in this manual, the protection provided by the equipment may be impaired.

**ATTENTION**

For marine applications, additionally to the standard instructions described on this chapter, the following installation requirements shall be met:

- The product shall be installed in a metallic cabinet.

## 3.1. Mechanical Installation

Nexto XF controllers were designed to be installed in a standard DIN rail. Additionally, the user shall provide a suitable enclosure that meets the system protection and safety requirements. The next sections shows the procedures for installing and removing the controller.

### 3.1.1. Installing the controller

To install the controller on the DIN rail, first move the four locks on open position as indicated on the figure below:

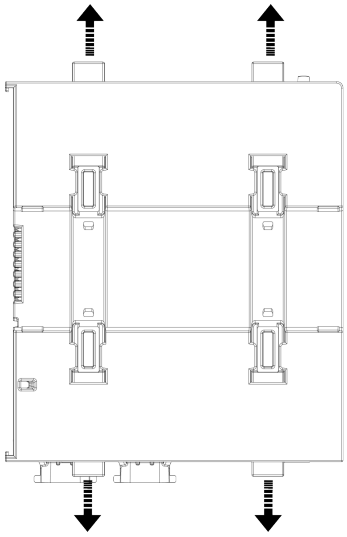


Figure 4: Moving the four locks to open position

Next, place the controller on the DIN rail fitting and move the four locks to closed position to lock the controller on the DIN rail, as shown on the figure below:

**CAUTION**

Always install the product by mounting it perpendicularly onto the DIN rail. Do not insert it and slide from the DIN rail end, as this may bend or damage the rear grounding spring.

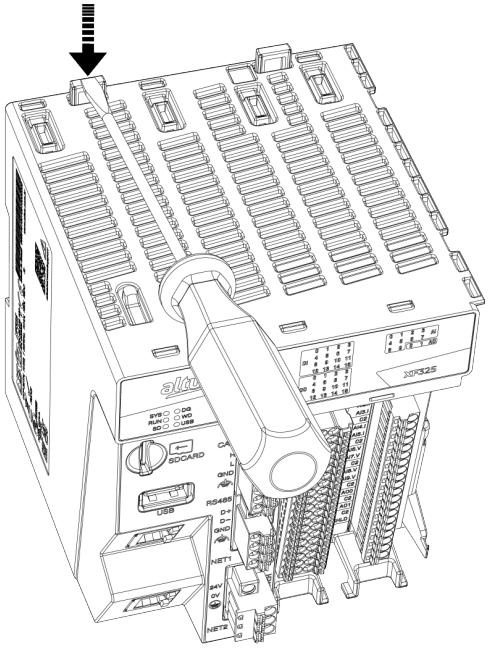


Figure 5: Locking the controller on the DIN rail

### 3.1.2. Removing the controller

To remove the controller from the DIN rail, just move the four locks to the open position as shown on the figure below:

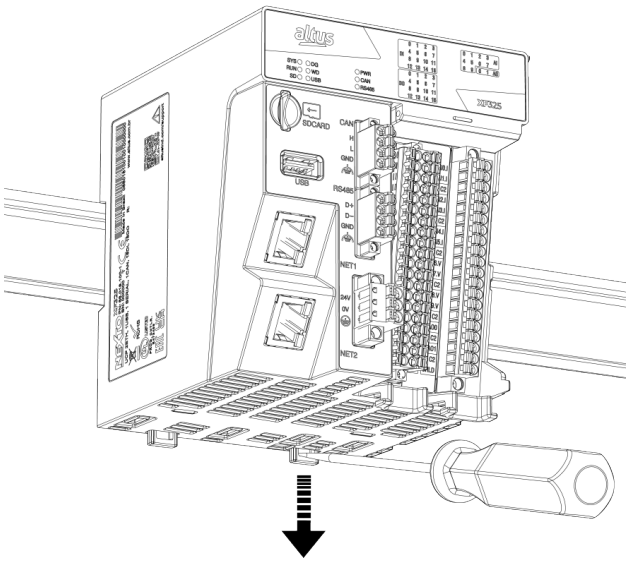


Figure 6: Unlocking the controller from the DIN rail

### 3.1.3. Installing and removing the connector-plug for embedded I/O

To install the connector-plug onto the sockets, align them with the socket, ensuring the correct orientation, and press it firmly until fully seated. Then tighten the screw flanges on both sides of the connector-plug to secure it in place and prevent accidental disconnection.

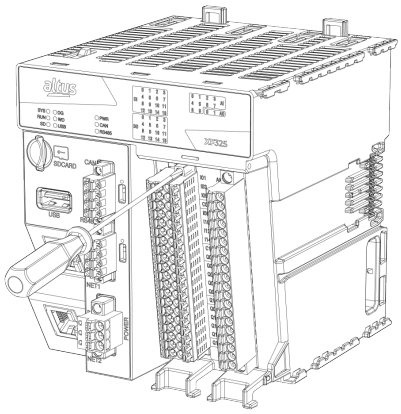


Figure 7: Installing I/O connector-plug

To remove the connector-plug from the socket, loosen the screw flanges on both sides then pull the connector-plug straight out from the socket. To remove the 2x20 terminal block, you must place two thin screwdrivers on the sides of the connector-plug and carefully pry it off.

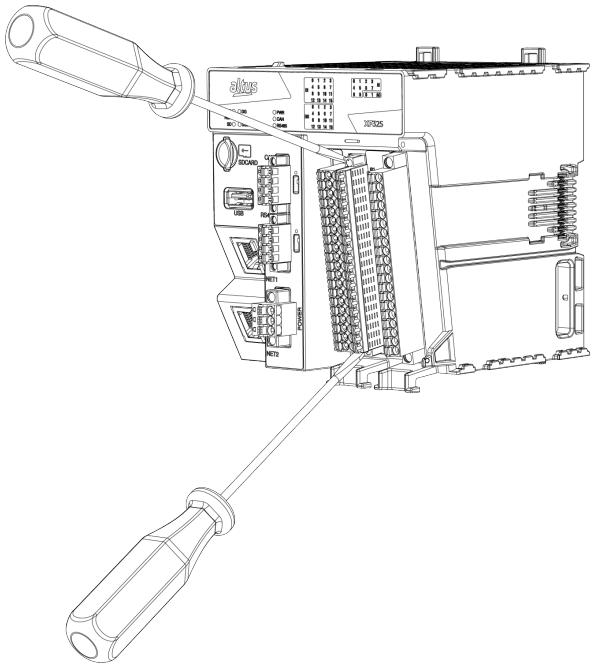


Figure 8: Removing the 2x20 Connector-Plug

To connect the wires to the connector-plug, insert directly into the corresponding terminal opening until it clicks into place. No tools are required. For stranded wires, a ferrule is recommended prior to insertion. To release a wire, press the release button on the corresponding terminal and pull the wire out.

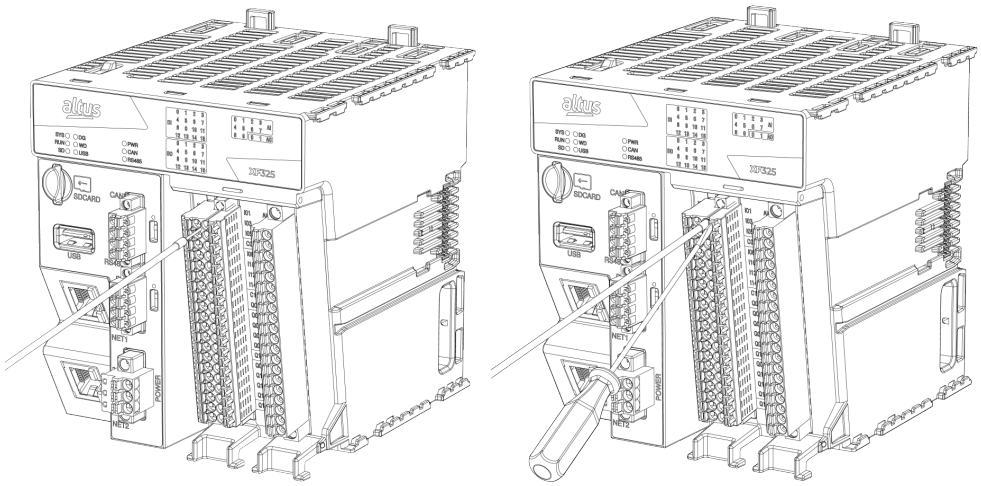


Figure 9: Installing wires over connector-plug

### 3.1.4. Installing and removing the I/O Termination

To install the I/O termination, place the termination at the insertion point and push horizontally towards the DIN rail until the movement stops. To remove the I/O termination, simply reverse the movement, sliding the termination horizontally to the insertion point until the tabs on the plastic part are unobstructed.

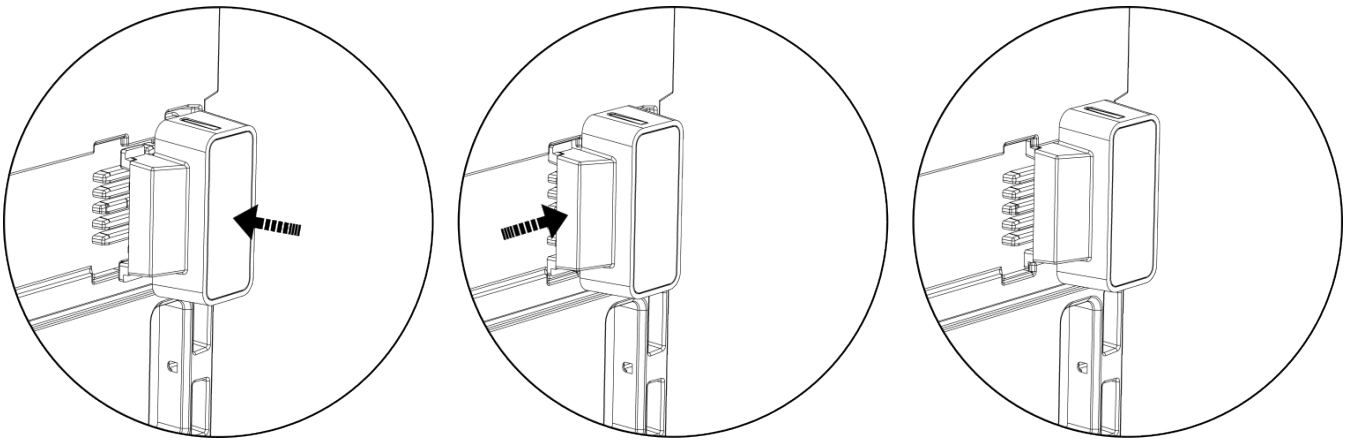


Figure 10: Installing I/O Termination

### 3.1.5. Installing and removing the I/O Cover

To install the integrated I/O cover, place the cover in a horizontal position and fit it into the left cavity, as shown in the figure below. Then, flex the cover tab to fit the pin into the right cavity.

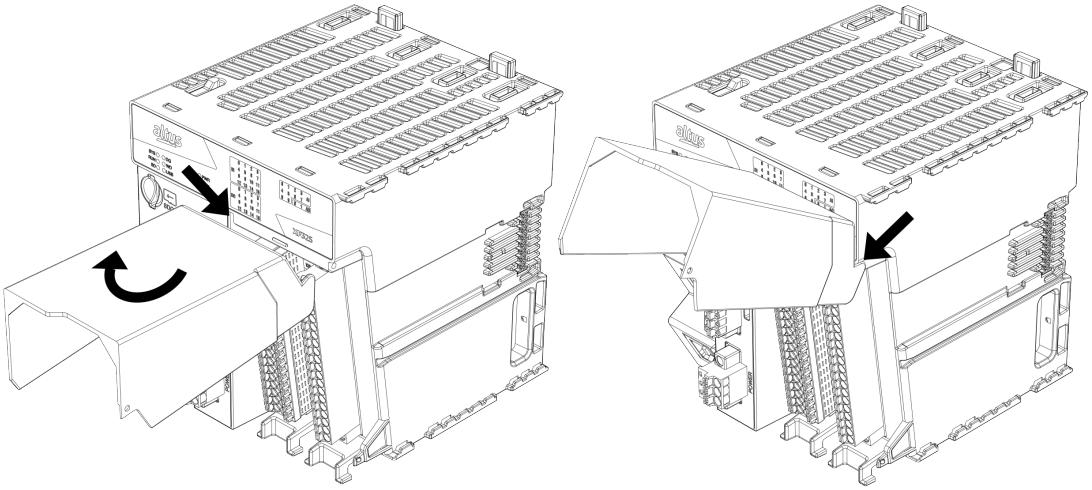


Figure 11: Installing Embedded I/O Cover

To remove the integrated I/O cover, place the cover in a horizontal position and flex the cover tab to remove the right pin from the cavity. Then, remove the left pin from the left cavity, as shown in the figure below.

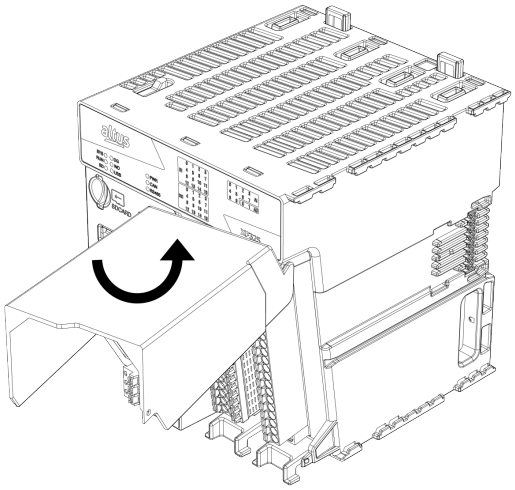


Figure 12: Removing Embedded I/O Cover

## 3.2. Spacing between CPU and other equipment in the panel

Proper installation of the PLC requires adequate clearance on all sides. This clearance serves two purposes: it allows proper device handling and ensures sufficient airflow for convective cooling, keeping the device within its operating temperature range. The minimum required clearances are presented in the following:

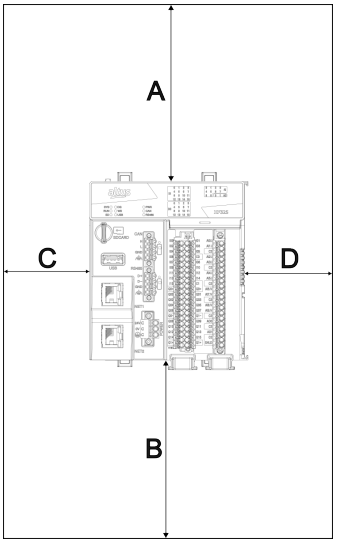



Figure 13: Free space around the PLC

| Dimension A | Dimension B | Dimension C | Dimension D |
|---|---|---|---|
| 10 cm | 10 cm | 4 cm | 4 cm |

Table 28: Free space dimension around the PLC

## 3.3. Cable Duct Dimensioning

When dimensioning the cable duct, besides the area occupied by the wires, the internal heating caused by the heat dissipated from the wires must be observed, as it may lead to a reduction in the usable area of the duct.

Use the following rule: duct area >= sum of wire areas / 0.4

Wire cross-sectional area:

$$A = \left(3,14 \times radius^2\right) \quad (1)$$

The wire area considered must be the total cross-sectional area, including the insulation.

## 3.4. Horizontal/Vertical Assembly

Nexto Series only supports horizontal mounting. Vertical installation in the rack is not allowed.

## 3.5. Thermal Design

The equipment is designed to work in a maximum determined room temperature. Therefore, this must be the maximum internal temperature inside the panel. The following issues must be observed in the panel design:

- Dimension panel with enough internal volume to allow a good air flow
- Predict forced ventilation or air exchangers with the room, if necessary, to avoid temperature levels beyond the specified limit. In critic cases is recommended cooling equipment use, in order to keep the temperature levels within operation limits
- Distribute equally heat sources within the panel
- Consider the high current conduction cables heat dissipation to avoid chute overheating

Following, the user will find a method to calculate the panel internal temperature is shown, regarding its dissipation and power.

### 3.5.1. Heat dissipation in an electrical panel

Each electric panel dissipates, through its surface, a defined heat amount for a specific difference between internal and external temperature. To calculate the heat dissipation in situations which the temperature difference, internal and external, reaches up to 50 °C, the following quantities must be considered:

- Panel effective dissipation surface; calculated according DIN-VED 0660 standard chapter 500, as indicated by the installation type
- The dissipation constant for the painted steel plate in W/m$^2$ °C
- The panel ventilation conditions (installation place)
- Panel occupancy degree (internal air flow impedance)

From the quantities listed previously, only the panel effective dissipation surface can be calculated precisely.

Panel effective dissipation surface A (m$^2$) calculation:

The calculation of the "A" surface is made according the DIN-VDE standard, following the panel installation type.

| Installation type according DIN-VDE 0660/500 standard | Formula for A calculation (m$^2$) |
|---|---|
| Panel free on all sides | A = 1.8 * H * (L + P) + 1.4 * L * P |
| Panel with the back surface obstructed | A = 1.4 * L * (H + P) + 1.8 * P * H |
| Panel with one side surface obstructed | A = 1.4 * L * (H + L) + 1.8 * L * H |
| Panel with one side surface and the back surface obstructed | A = 1.4 * H * (L + P) + 1.4 * L * P |
| Panel with both side surfaces obstructed | A = 1.8 * L * H + 1.4 * L * P + P * H |
| Panel with both side surfaces and the back surface obstructed | A = 1.4 * L * (H + P) + P * H |
| Panel with both side surfaces, the back surface and the front surface obstructed | A = 1.4 * L * H + 0.7 * L * P + P * H |

Table 29: Effective Dissipation Surface Calculation

L = Width (m), H = Height (m), P = Depth (m)

In built panels application with painted steel plate, for a null air flow surrounding it, the heat dissipation constant can be considered 5.5 W/m$^2$ °C.

The power dissipated by a panel can then be calculated through the equation Qs = k * A * (internal temperature – external temperature), or obtained from figure below.

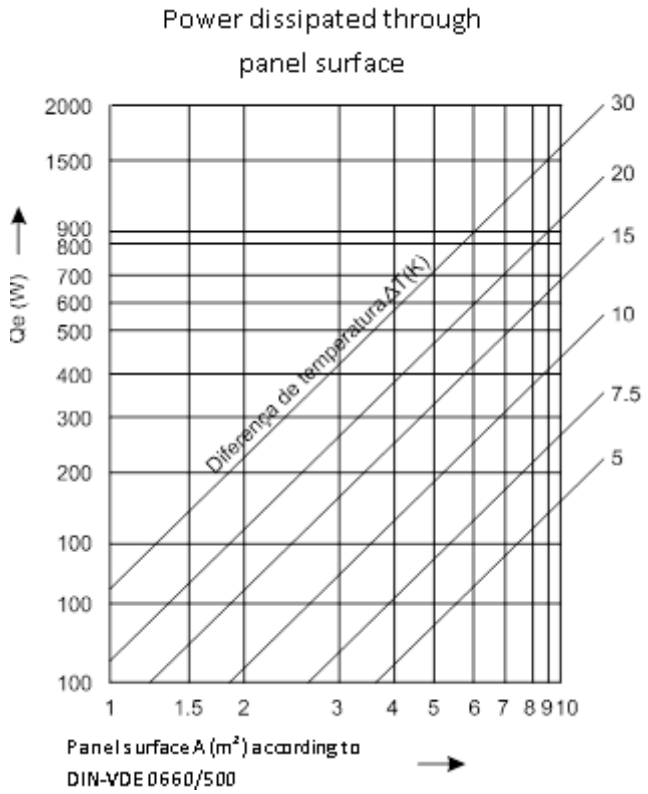



Figure 14: Dissipated Power x Surface x Temperature difference

However, this value may be triple if forced ventilation is applied in the panel interior.

The air flow inside a panel is obstructed by the equipment installed, generating concentrated heating points. In this case, fans may be installed to increase the air flow within the panel.

The forced air circulation through fans in the panel interior brings an improvement in the natural convection and tendency to keep the temperature degree equal throughout the panel. Without it, there's a tendency to form a concentration of heat in the upper part of the panel due to natural convection.

Examples:

For a panel free on all sides, with an effective area of 3.96 $m^2$, installed power of 350 W and room temperature of 30 °C, calculate the internal average temperature.

Qs = k * A * (Ti – Te)

350 = 5.5 * 3.96 * (Ti – 30)

Ti = 46 °C

For the same panel, calculate the internal temperature for an installed power of 1000W.

Qs = k * A * (Ti – Te)

1000 = 5.5 * 3.96 * (Ti – 30)

Ti = 76 °C

In this case, the temperature has exceeded the equipment operation limit (60 °C), and a solution must be provided to exchange the exceeding heat. The installed power limit for the internal temperature limit of 60 °C is given by:

Qs = k * A * (Ti – Te)

Qs = 5.5 * 3.96 * (60 – 30)

Qs = 653W, this being the limit, the exceeding 347W (1000W – 653W) must be dealt with by an air conditioned device, for instance.

**ATTENTION**

In the previous calculation, it must be observed that the internal temperature is always the average and if there's no forced ventilation inside the panel, the temperature of the upper part will be higher than the base, and concentrated heating points may appear. The due security coefficient must be taken for each case.

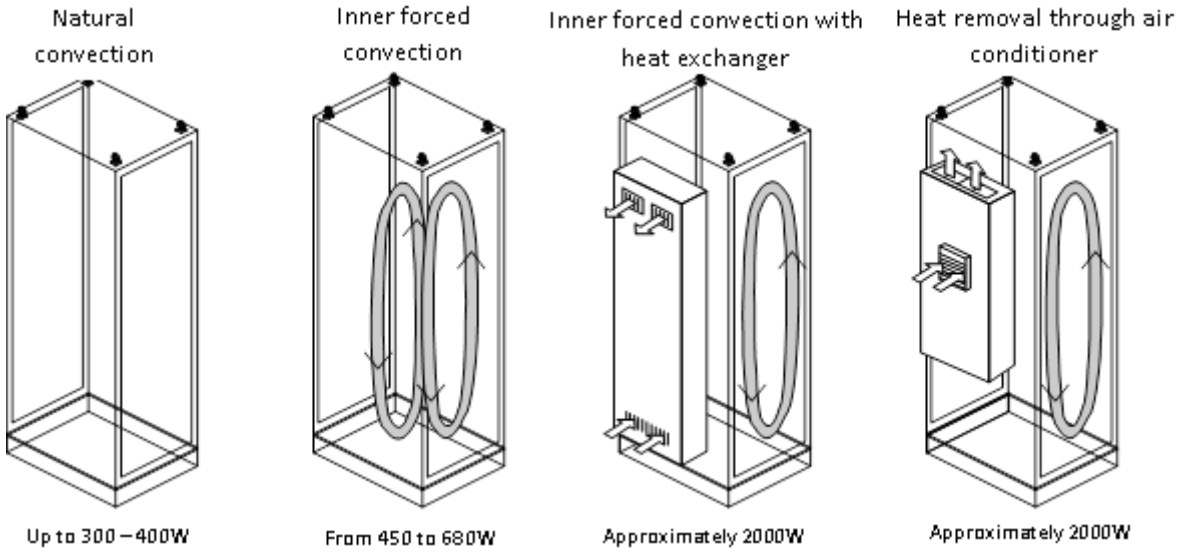



Figure 15: Example of Heat Flow – Closed Installation

A much higher heat dissipation, comparing with the one obtained previously, can be achieved if the room heat exchange is allowed. The ventilation is often used introducing ventilation gaps on the side surfaces, door or on the back of the panel. This will evidently reduce the panel protection degree (IP).

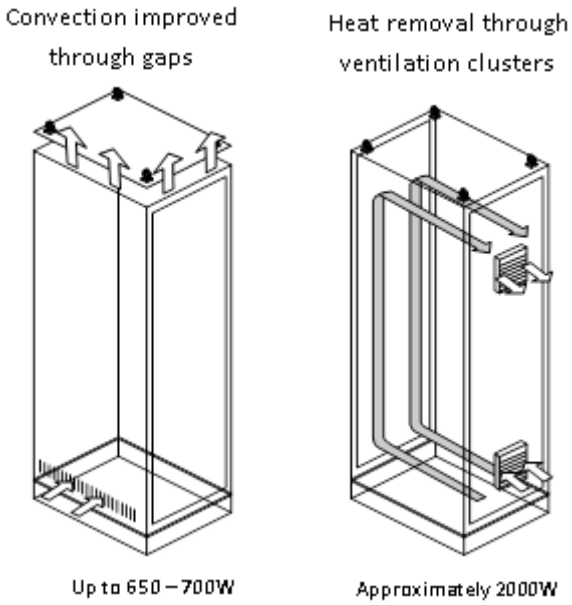



Figure 16: Example of Heat Flow – Open Installation

## 3.6. Electrical Installation

**Danger**

When performing any installation in an electrical panel, ensure that the power supply is TURNED OFF.

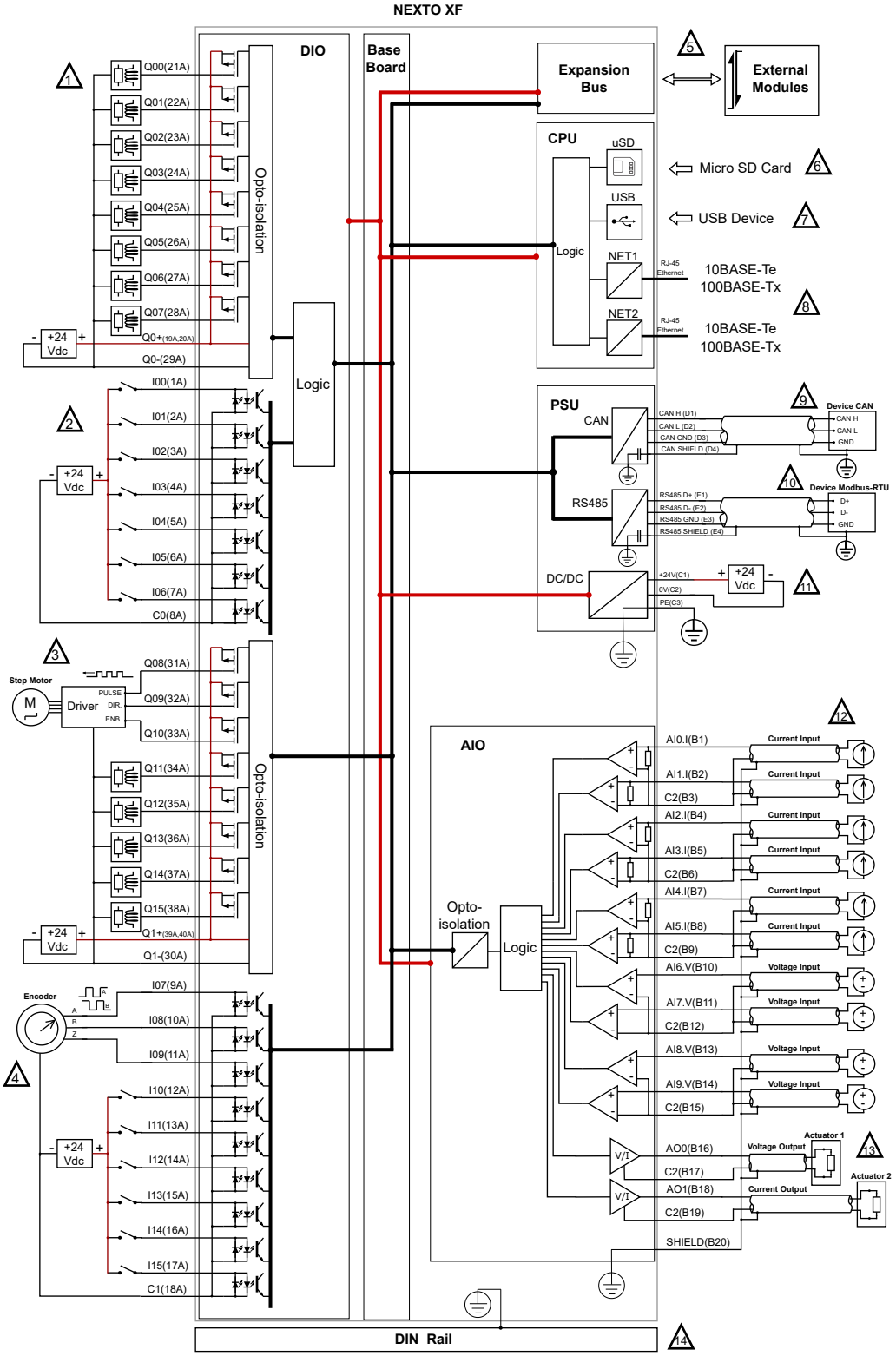



Figure 17: XF3xx electrical wiring diagram

**Diagram notes:**

1 Typical connection of the isolated digital outputs Q00 to Q07 (source type). Independent external 24Vdc power supply connected to terminals Q0+ and Q0-.

2 Typical connection of the isolated digital inputs I00 to I06 (sink type). Independent external 24Vdc power supply connected to the common terminal C0 and switch contacts.

3 Typical connection of the isolated fast digital outputs Q08 to Q15 for stepper motor control and discrete load activation. Independent external 24Vdc power supply connected to terminals Q1+ and Q1-.

4 Typical connection of the isolated fast digital inputs I07 to I15, with a 3-signal encoder (A, B, and Z) and discrete switches. Independent external 24Vdc power supply connected to the common terminal C1 and switch contacts.

5 I/O Expansion Modules bus connection to external modules or termination cover.

6 Micro SD card slot.

7 USB device port. Refer to the technical specifications table for the USB port to obtain the list of supported devices.

8 Use the Ethernet cables specified in the Related Products section. Connector shielding connected directly to internal ground.

9 Typical CAN serial interface connection. Cable shielding internally coupled to ground via capacitor; the cable shielding must be connected to ground externally at only one end.

10 Typical RS-485 serial interface connection. Cable shielding internally coupled to ground via capacitor; the cable shielding must be connected to ground externally at only one end.

11 Connection of the system power supply, nominally 24 VDC. The system grounding point if the DIN rail is not grounded.

12 Typical connection for analog voltage and current inputs. The cable shield grounding occurs at a single point on the connector. Cable shield connection should occur at only one end. This note does not apply to XF300 and XF300-B.

13 Typical connection for configurable voltage and current analog outputs. The cable shield grounding occurs at a single point on the connector. Cable shield connection should occur at only one end. This note does not apply to XF315, XF300 and XF300-B.

14 The grounding is performed through direct contact of the product with a standard DIN rail, which must be metallic and connected through a proper external grounding, see note 11.

Protective earth terminal.

## 3.7. Ethernet Network Connection

The ETH communication interface, identified as NET 1 and NET 2 in Mastertool, allows the connection with an Ethernet network and programming with this tool.

The Ethernet network connection uses twisted pair cables (10/100Base-TX) and the speed detection is automatically made by the Nexto XF controller. This cable must have one of its endings connected to the interface that is likely to be used and another one to the HUB, switch, microcomputer or other Ethernet network point.

### 3.7.1. IP Address

The NET 1 Ethernet interface is used for Ethernet communication and for CPU configuration which comes with the following default parameters configuration:

| | NET 1 |
|---|---|
| **IP Address** | 192.168.15.1 |
| **Subnetwork Mask** | 255.255.255.0 |
| **Gateway Address** | 192.168.15.253 |

Table 30: Default Parameters Configuration for Ethernet NET 1 Interface

First, the NET 1 interface must be connected to a PC network with the same subnet mask to communicate with Mastertool, where the network parameters can be modified. For further information regarding configuration and parameters modifications, see Ethernet Interface chapter.

| | NET 2 |
|---|---|
| **IP Address** | 192.168.16.1 |
| **Subnetwork Mask** | 255.255.255.0 |
| **Gateway Address** | 0.0.0.0 |

Table 31: Default Parameters Configuration for Ethernet NET 2 Interface

### 3.7.2. Gratuitous ARP

The NETx Ethernet interface promptly sends ARP packets type in broadcast informing its IP and MAC address for all devices connected to the network. These packets are sent during a new application download by Mastertool and in the CPU startup when the application goes into Run mode.

Five ARP commands are triggered within a 200 ms initial interval, doubling the interval every new triggered command, totalizing 3 s. Example: first trigger occurs at time 0, the second one at 200 ms and the third one at 600 ms and so on until the fifth trigger at time 3 s.

### 3.7.3. Network Cable Installation

Nexto Series CPUs Ethernet ports, identified on the panel by NET, have standard pinout which are the same used in PCs. The connector type, cable type, physical level, among other details regarding the CPU and the Ethernet network device are defined in the Ethernet Interface.

The table below present the RJ-45 Nexto CPU female connector, with the identification and description of the valid pinout for 10BASE-TE and 100BASE-TX physical levels.

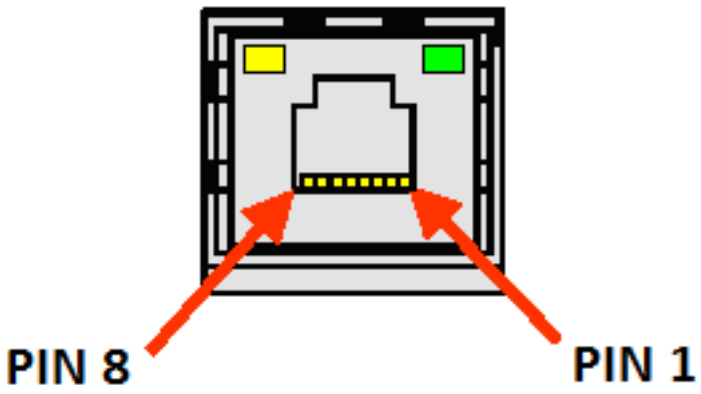



Figure 18: RJ45 Female Connector

| Pin | Signal | Description |
|---|---|---|
| 1 | TXD + | Data transmission, positive |
| 2 | TXD - | Data transmission, negative |
| 3 | RXD + | Data reception, positive |
| 4 | NU | Not used |
| 5 | NU | Not used |
| 6 | RXD - | Data reception, negative |
| 7 | NU | Not used |
| 8 | NU | Not used |

Table 32: RJ45 Female Connector Pinout - 10BASE-TE and 100BASE-TX

The interface can be connected in a communication network through a hub or switch, or straight from the communication equipment. In this last case, due to Nexto CPUs Auto Crossover feature, there is no need for a cross-over network cable, the one used to connect two PCs point to point via Ethernet port.

It is important to stress that it is understood by network cable a pair of RJ45 male connectors connected by a UTP or ScTP cable, category 5 whether straight connecting or cross-over. It is used to communicate two devices through the Ethernet port.

These cables normally have a connection lock which guarantees a perfect connection between the interface female connector and the cable male connector. At the installation moment, the male connector must be inserted in the module female connector until a click is heard, assuring the lock action. To disconnect the cable from the module, the lock lever must be used to unlock one from the other.

## 3.8. Serial RS-485 and CAN Network Connection

Both RS-485 and CAN interface use two communication signals and a ground. The recommended cable is AL-2306, using one of the two pairs and the shield. If the controller is placed at one of the network ends, the external termination shall be enabled, manually via the DIP switch located on the front panel of the CPU, as illustrated below and in the diagram Electrical Installation. The DIP switch position facing upwards means termination is enabled.

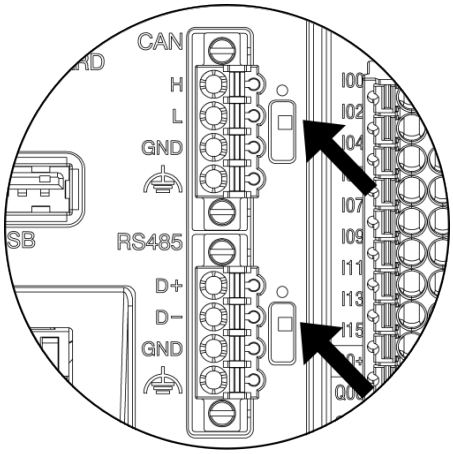



Figure 19: Installing RS-485 and CAN network termination

## 3.9. Memory Card Installation

This section presents how to insert the memory card into the models Nexto Series CPUs. For further information see Memory Card section.

Initially, care must be taken with the correct position the memory card must be inserted. One corner of it is different from the other three and this one must be used as reference for the card correct insertion. Therefore, the memory card must be inserted following the depiction on the CPU frontal part or the way showed on figure below.

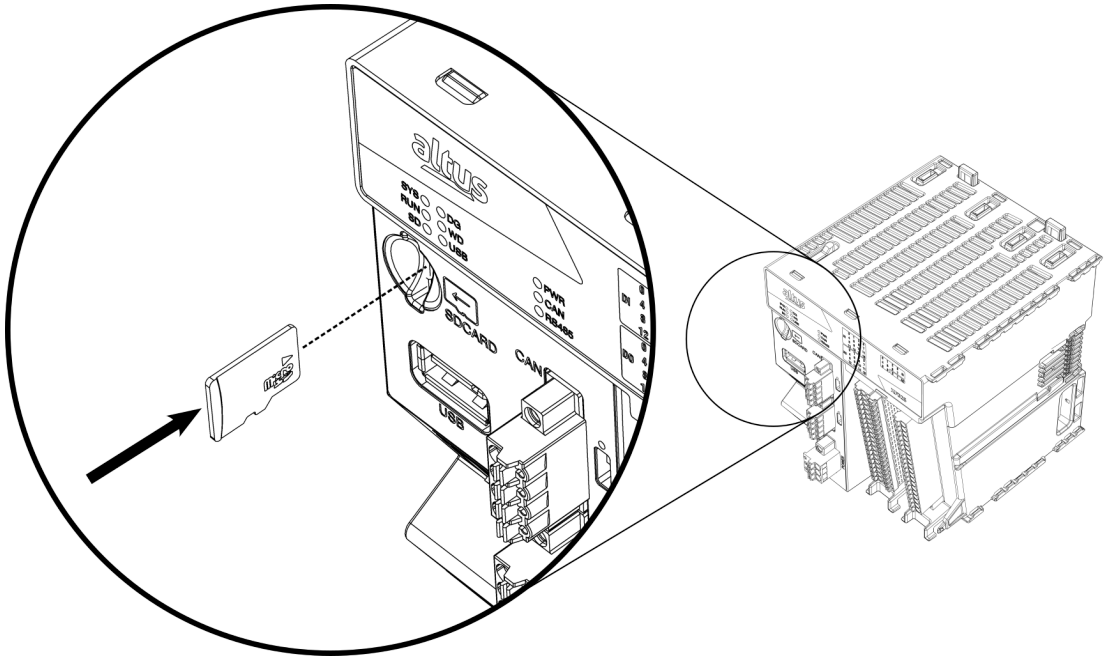



Figure 20: Inserting the Memory Card

When the card is correctly installed, its correspondent LED will turn on. For card secure removing the user must access the webpage and press the button "Unmount", right after, the correspondent LED will turn off and its state will change in status webpage status info. The card is now ready to be taken off. For that, the card must be pressed against the CPU until a click is heard, then release it and withdraw it from the compartment. At this moment the card will be loose.

# 4. Initial Programming

The main goal of this chapter is to help the programming and configuration of Nexto Series CPUs, allowing the user to take the first steps before starting to program the device.

Nexto Series CPU uses the standard IEC 61131-3 for language programming, which are: ST, LD, SFC and FBD, and besides these, an extra language, CFC. These languages can be separated in text and graphic. ST is a text language similar to C. LD, SFC, FBD and CFC are graphic languages. LD uses the relay block representation and it is similar to relay diagrams. SFC uses the sequence diagram representation, allowing an easy way to see the event sequence. FBD and CFC use a group of function blocks, allowing a clear vision of the functions executed by each action.

Mastertool allows the use of all languages in the same project, so the user can apply the best features offered by each language, resulting in more efficient applications development, for easy documentation and future maintenance.

For further information regarding programming see the Mastertool user manual.

## 4.1. Memory Organization and Access

Different from other devices of the Nexto Series (which are based on a big-endian CPU), this CPU model is based on an ARM CPU, which uses the traditional little-endian memory organization (the same found on x86 and Intel processors). On this type of memory organization, the least significant byte is stored first and will always be the smallest address (e.g. %QB0 will always be less significant than %QB1, as shown on the table below, where, for CPUNEXTO string, the letter O is byte 0 and the letter C is the byte 7).

Besides this, the memory access must be done carefully as the variables with higher number of bits (WORD, DWORD, LONG), use as index the most significant byte, in other words, the %QD4 will always have as most significant byte the %QB4. Therefore it will not be necessary to make calculus to discover which DWORD correspond to defined bytes. The table below, shows little and big endian organization.

| **MSB ← Little-endian → LSB** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| BYTE | %QB7 | %QB6 | %QB5 | %QB4 | %QB3 | %QB2 | %QB1 | %QB0 |
| | C | P | U | N | E | X | T | O |
| WORD | %QW6 | | %QW4 | | %QW2 | | %QW0 | |
| | CP | | UN | | EX | | TO | |
| DWORD | %QD4 | | | | %QD0 | | | |
| | CPUN | | | | EXTO | | | |
| LWORD | %QL0 | | | | | | | |
| | CPUNEXTO | | | | | | | |
| **MSB ← Big-endian → LSB** | | | | | | | | |
| BYTE | %QB0 | %QB1 | %QB2 | %QB3 | %QB4 | %QB5 | %QB6 | %QB7 |
| | C | P | U | N | E | X | T | O |
| WORD | %QW0 | | %QW2 | | %QW4 | | %QW6 | |
| | CP | | UN | | EX | | TO | |
| DWORD | %QD0 | | | | %QD4 | | | |
| | CPUN | | | | EXTO | | | |
| LWORD | %QL0 | | | | | | | |
| | CPUNEXTO | | | | | | | |

Table 33: Memory Organization and Access Example

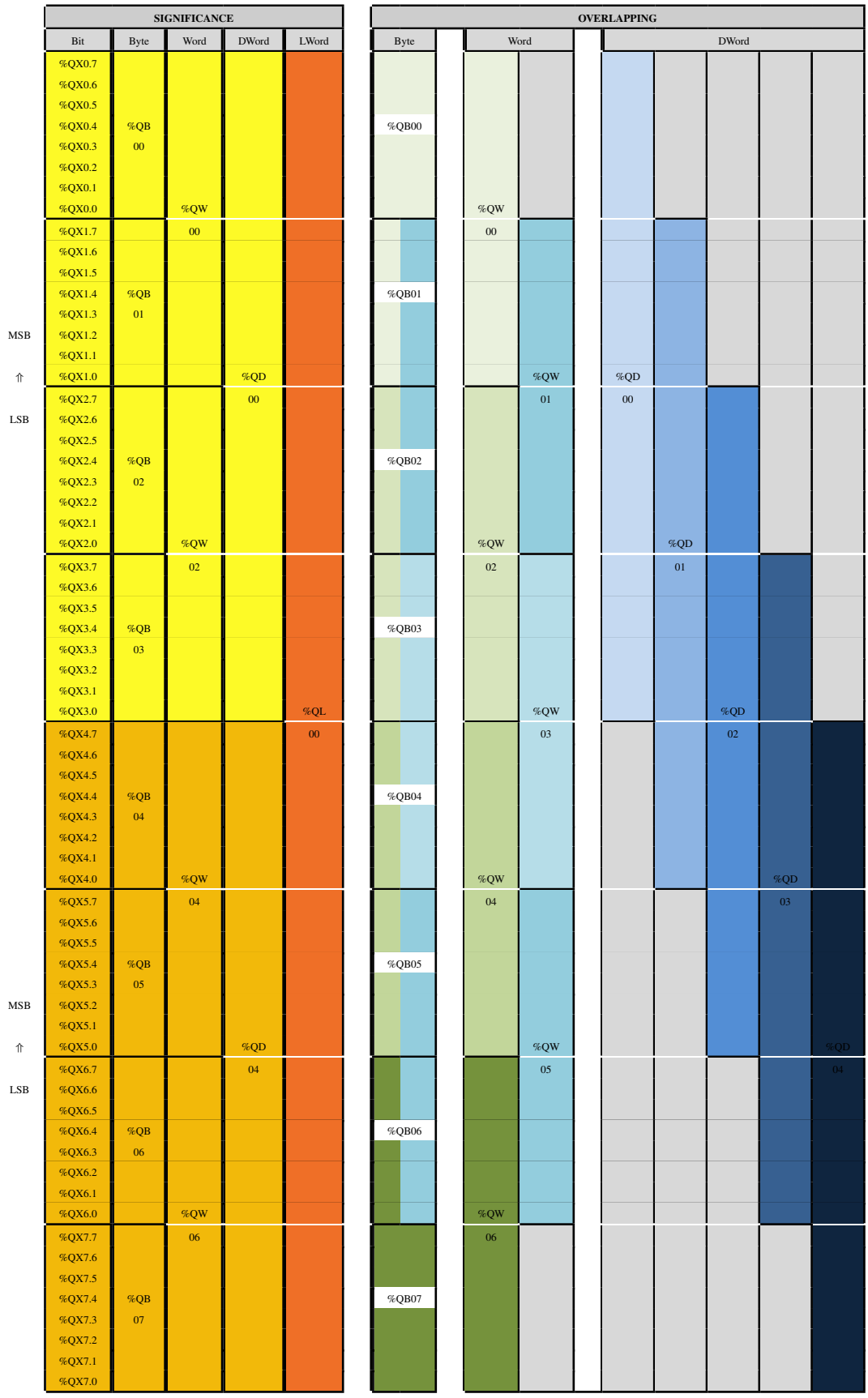



Table 34: Memory Organization and Access

The table above shows the organization and memory access, illustrating the significance of bytes and the disposition of other variable types, including overlapping.

## 4.2. Project Profiles

A project profile consists in an application template combined with a group of verification rules which guides the development of the application, reducing the programming complexity. For Nexto XF controllers, there is only one project profile available: Fast.

The Fast profile is selected on the project creation wizard. Each project profile defines a template for the tasks and programs, which are pre-created according to the selected Project Profile. Also, during the project compilation (generate code), the compiler verify all the rules defined by the selected profile.

The following section details the characteristics of this profile. It is important to note that Mastertool allows the profile change from an existent project (see project update section in the Mastertool user manual), but it's up to the developer to make any necessary adjustments so that the project becomes compatible with the rules of the new selected profile.

**ATTENTION**

Through the description of the Project profiles some tasks types are mentioned, which are described in the section 'Task Configuration', of the Mastertool user manual.

### 4.2.1. Fast Profile

In the Fast Profile, by default, the application has a user task of the Cyclic type called MainTask. This task is responsible for implementing a single Program type POU called MainPrg. This program can call other programming units of the Program, Function or Function Block types, but any user code will run exclusively by MainTask task.

This profile is characterized by allowing shorter intervals in the MainTask, allowing faster execution of user code. This profile may further include an interruption task, with a higher priority than the MainTask, and hence, can interrupt its execution at any time.

| Task | POU | Priority | Type | Interval | Event |
|---|---|---|---|---|---|
| **MainTask** | MainPrg | 13 | Cyclic | 20 ms | - |

Table 35: Fast Profile Task

### 4.2.2. Maximum Number of Tasks

The user can create several task types, while the established numbers for each one and the total value are not surpassed. Table on General Features shows the maximum number of tasks. Mastertool shows detailed tasks type limits in Task Configuration "Properties" tab.


Figure 21: Maximum number of tasks

The several tasks that user can create and associated tasks to each enabled functionality are shown in "Task Configuration" in the tree view. Under the tab "Monitor" they can be count compared with their limits.

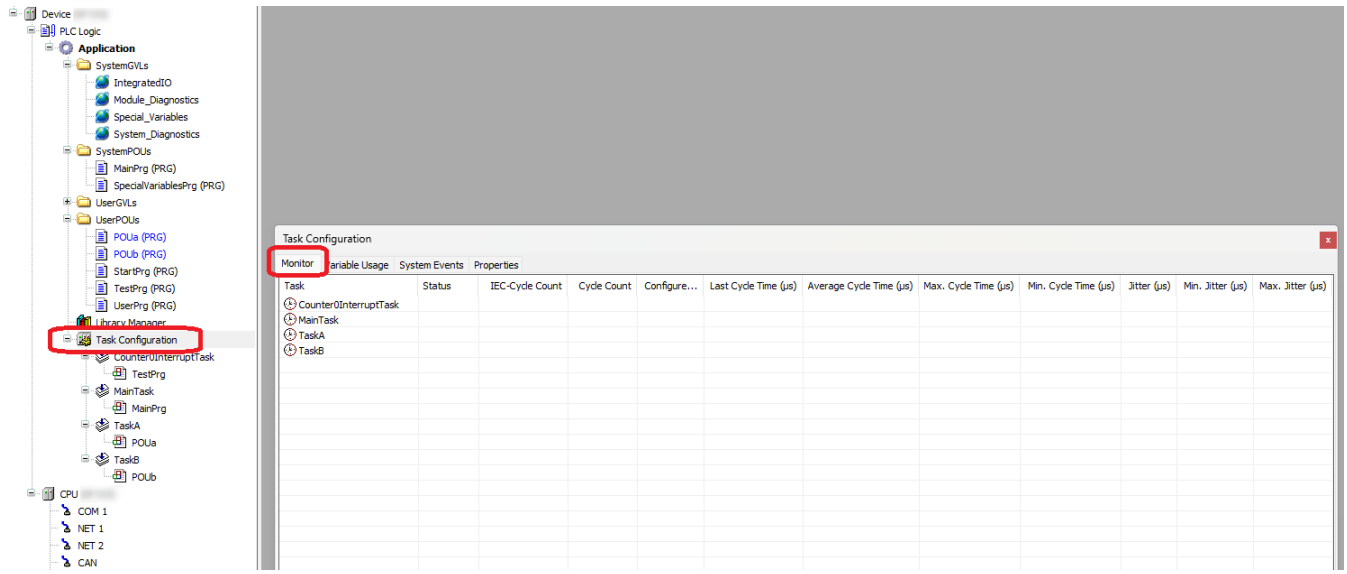



Figure 22: Current number of tasks

## 4.3. CPU Configuration

The Nexto CPU configuration is located in the device tree, as shown on figure below, and can be accessed by a double-click on the corresponding object.Further information can be found in the Controller's CPU.

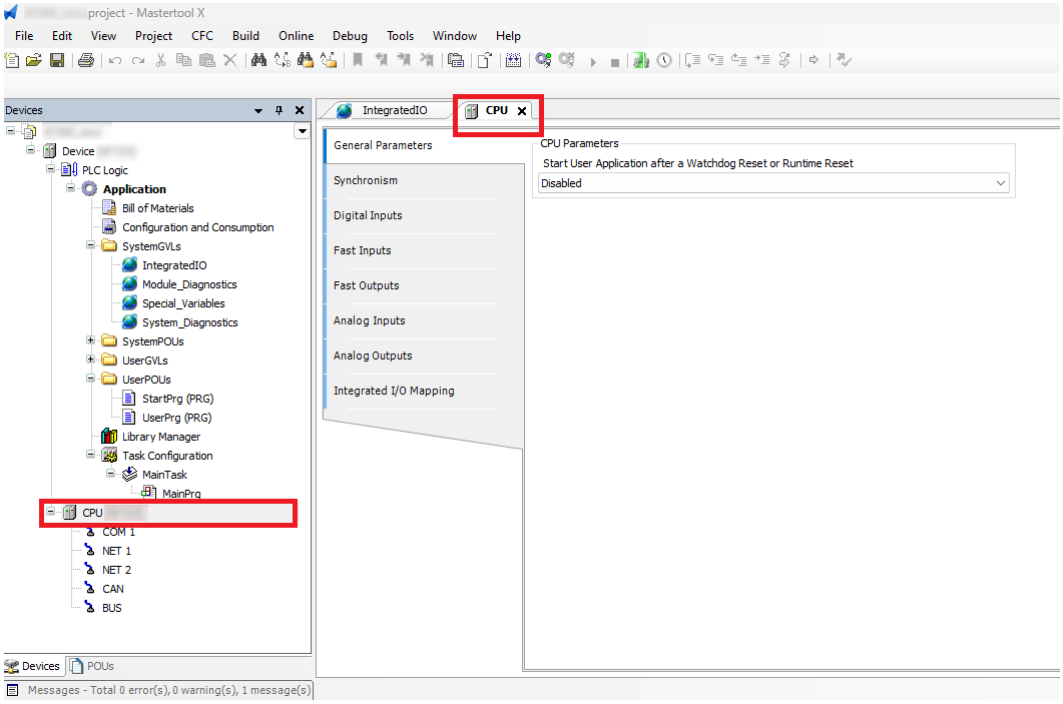



Figure 23: CPU Configuration

Besides that, by double-clicking on CPU's NET 1 icon, it's possible to configure the Ethernet interface that will be used for communication between the controller and Mastertool.

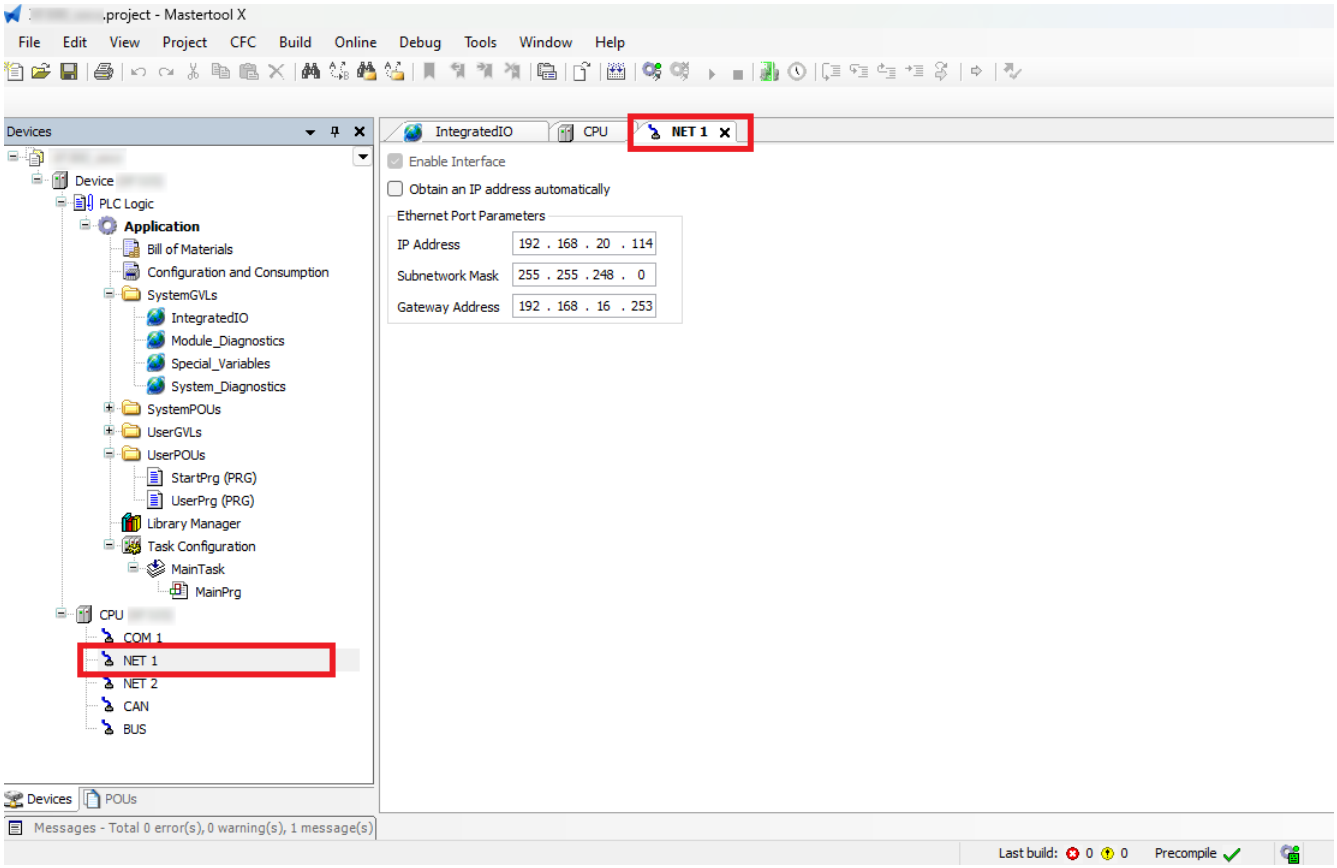



Figure 24: Configuring the Communication Port

The configuration defined on this tab will be applied to the device only when sending the application to the device (download), which is described further on sections Finding the Device and Login.

Additionally, the device tree also offers the configuration of the integrated I/O available on Nexto XF controllers, as shown on the figure below. In this tab it is possible to configure digital inputs filters, the mode of each analog input, among other parameters.

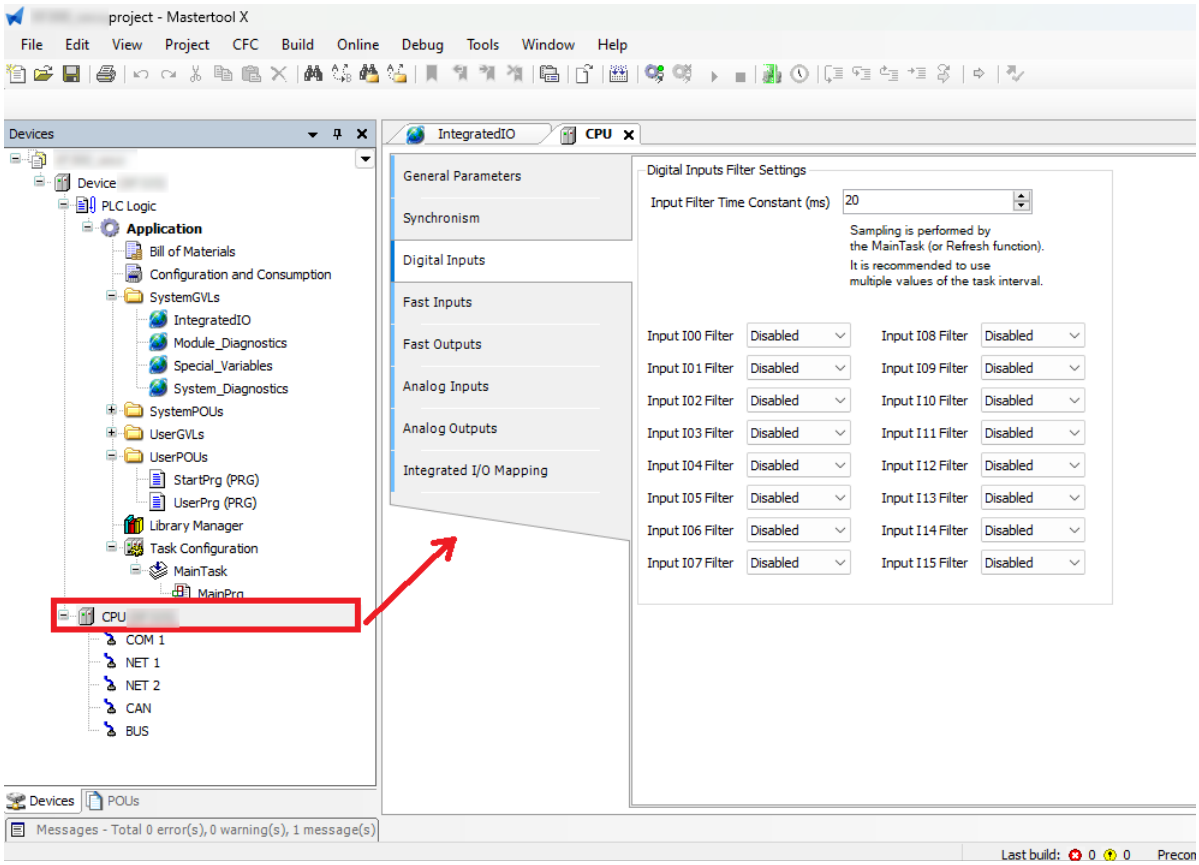



Figure 25: Configuring the Integrated I/O

## 4.4. Libraries

There are several Mastertool resources which are available through libraries. Therefore, these libraries must be inserted in the project so its utilization becomes possible. The insertion procedure and more information about available libraries must be found in the programming manual.

## 4.5. Inserting a Protocol Instance

The Nexto Series CPUs, as described in the Protocols section, offers several communication protocols. Except for the OPC DA and OPC UA communication, which have a different configuration procedure, the insertion of a protocol can be done by simply right-clicking on the desired communication interface, selecting to add the device and finally performing the configuration as shown in the Communication Protocols section. Below is presented an examples.

### 4.5.1. MODBUS RTU

The first step for configuring the MODBUS RTU, in slave mode, is to include the instance in the desired COM (COM 1 in this case) by clicking with the right button on the *COM* and select *Add Device...*, as shown on figure below:

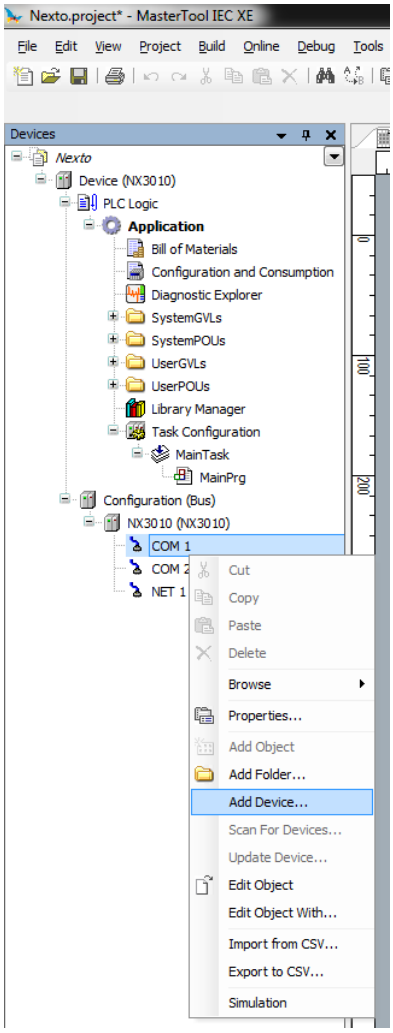



Figure 26: Adding an Instance

After that, the available protocols for the user will appear on the screen. Define the protocol configuration mode selecting *MODBUS Symbol RTU Slave*, for *symbolic mapping* setting or *MODBUS RTU Slave*, for *direct addressing (%Q)* and click on *Add Device*, as depicted on figure below.

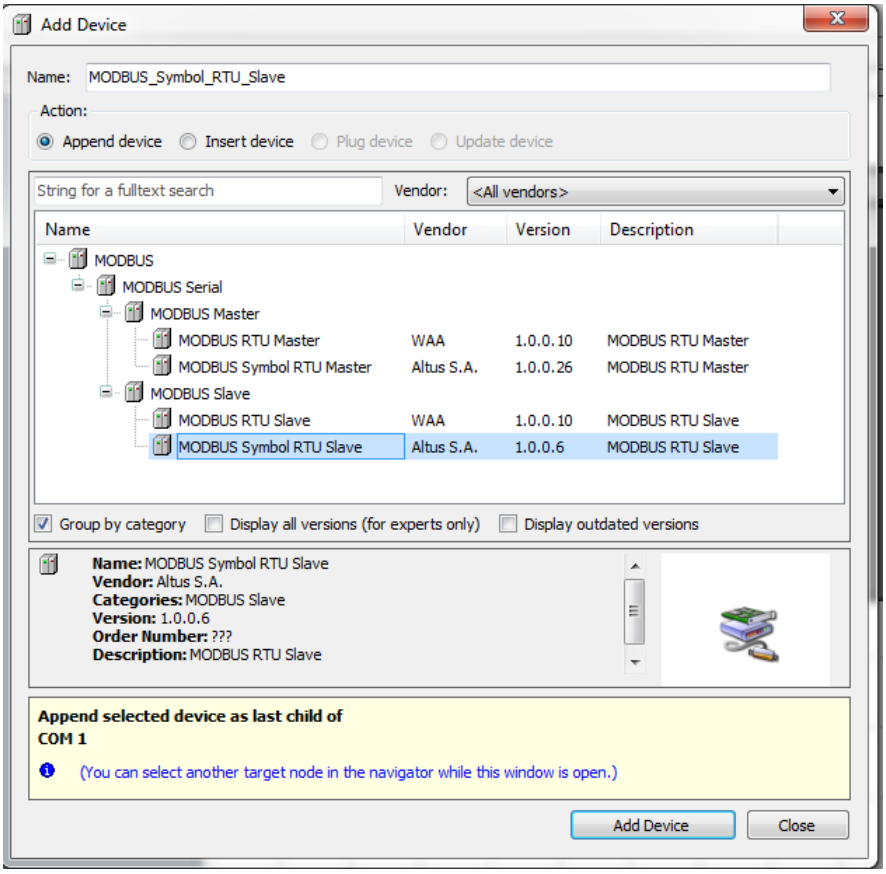



Figure 27: Selecting the Protocol

### 4.5.2. MODBUS Ethernet

The first step to configure the MODBUS Ethernet (Server in this example) is to include the instance in the desired NET. Click on the NET with the mouse right button and select Add Device..., as shown on Figure 28:

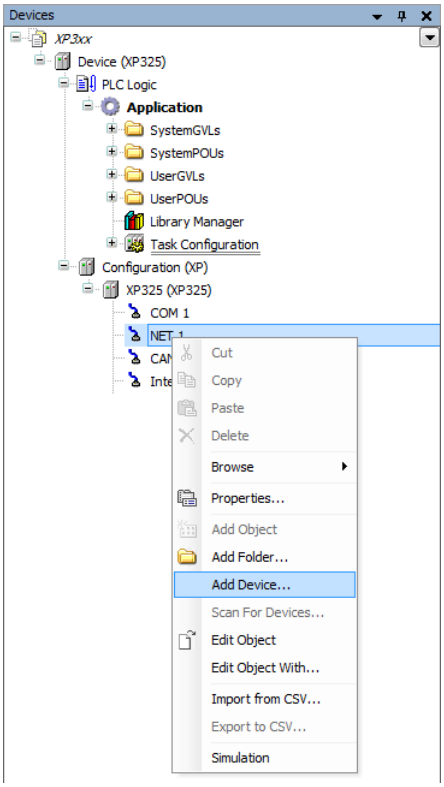



Figure 28: Adding the Instance

After that, the list of protocols will appear on the screen. Simply select MODBUS Symbol Server as described on the figure below:

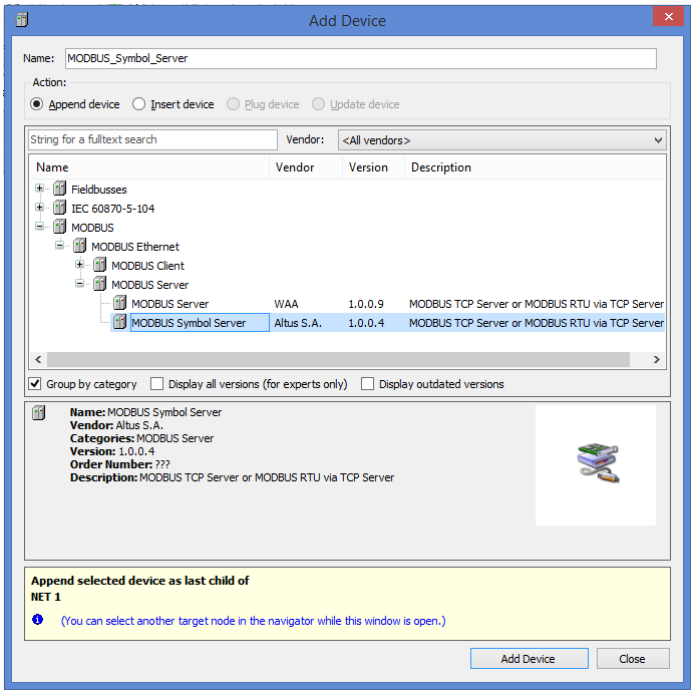



Figure 29: Selecting the Protocol

## 4.6. Finding the Device

To establish the communication between the CPU and Mastertool, first it's necessary to find and select the desired device. The configuration of this communication is located on the object *Device* on device tree, on *Communication Settings* tab. On this tab, after selecting the *Gateway* and clicking on button *Scan network*, Mastertool performs a search for devices and shows the CPUs found on the network of the Ethernet interface of the station where the tool is running.

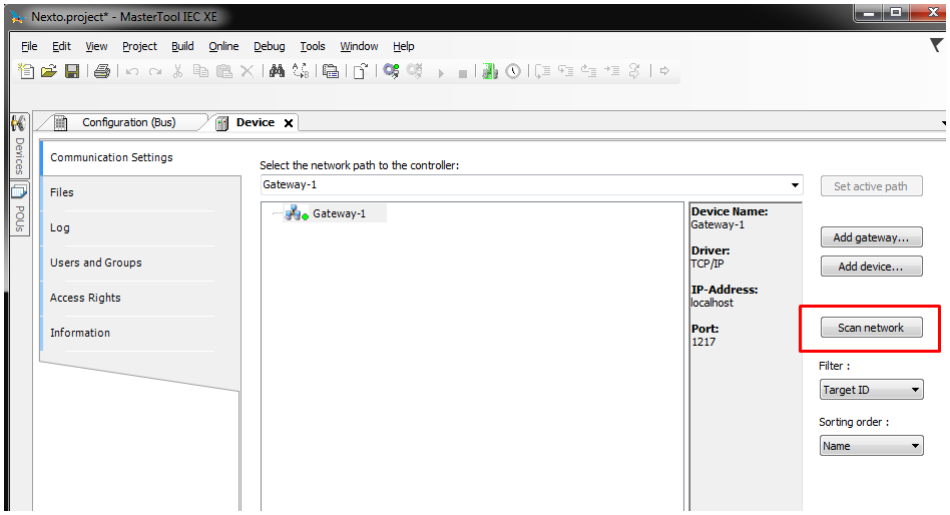



Figure 30: Finding the CPU

If there is no gateway previously configured, it can be included by the button *Add gateway*, using the default IP address *localhost* to use the gateway resident on the station or changing the IP address to use the device internal gateway.

Next, the desired controller must be selected by clicking on *Set active path*. This action selects the controller and informs the configuration software which controller shall be used to communicate and send the project.

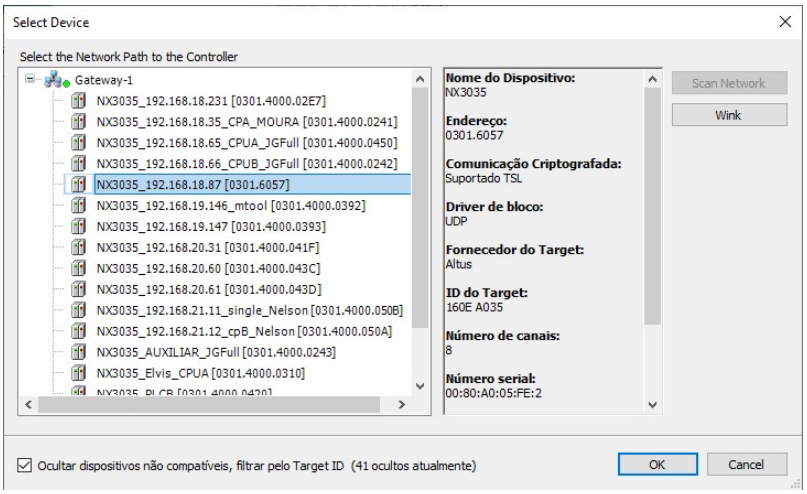



Figure 31: Selecting the CPU

Additionally, the user can change the default name of the device that is displayed. For that, you must click the right mouse button on the desired device and select *Change Device Name*. After a name change, the device will not return to the default name under any circumstances.

In case the Ethernet configuration of the CPU to be connected is in a different network from the Ethernet interface of the station, Mastertool will not be able to find the device. In this case, it's recommended to use the command *Easy Connection* located on Online menu.

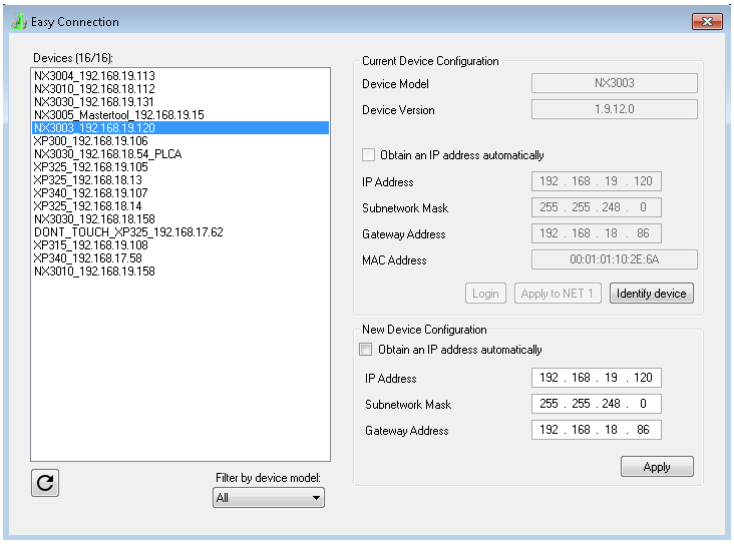



Figure 32: Easy Connection

This command performs a MAC level communication with the NET 1 interface of the device, allowing to permanently change the configuration of the CPU's Ethernet interface, independently of the IP configuration of the station and from the one previously configured on the device. So, with this command, it's possible to change the device configuration to put it on the same network of the Ethernet interface of the station where Mastertool is running, allowing to find and select the device for the communication. The complete description of *Easy Connection* command can be found in Mastertool's user manual.

## 4.7. Login

After compiling the application and fixing errors that might be found, it's time to send the project to the CPU. To do this, simply click on *Login* command located on the *Online* menu of Mastertool, as shown on the following figure. This operation may take a few seconds, depending on the size of the generated file.

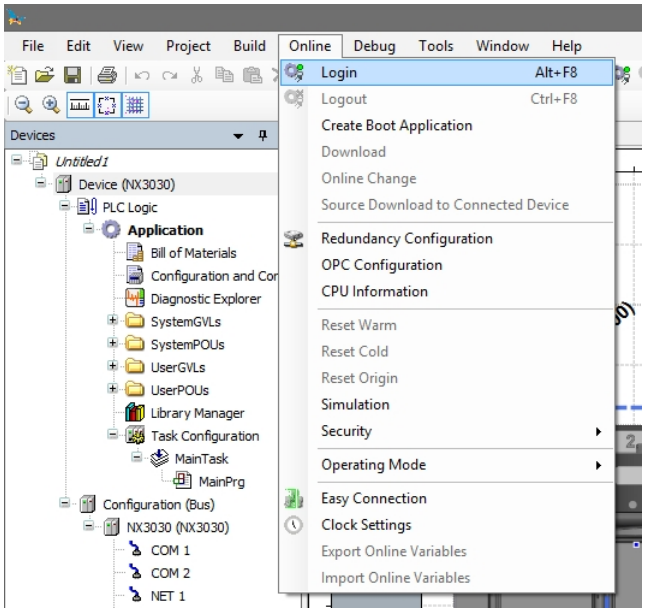



Figure 33: Sending the Project to the CPU

After the command execution, some user interface messages may appear, which are presented due to differences between an old project and the new project been sent, or simply because there was a variation in some variable.

If the Ethernet configuration of the project is different from the device, the communication may be interrupted at the end of download process when the new configuration is applied on the device. So, the following warning message will be presented, asking the user to proceed or not with this operation.

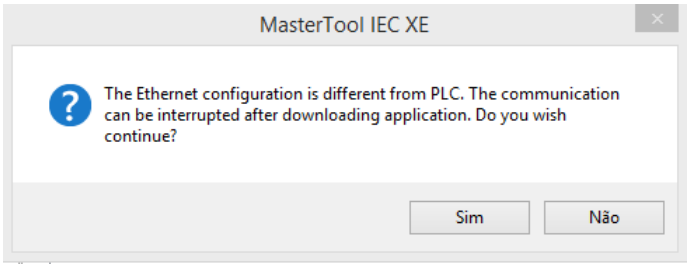



Figure 34: IP Configuration Warning

If there is no application on the CPU, the following message will be presented.

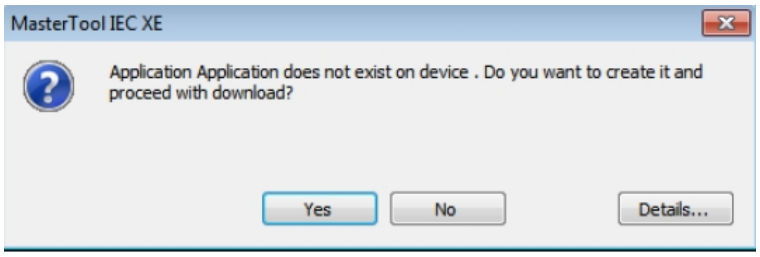



Figure 35: No application on the device

If there is already an application on the CPU, depending on the differences between the projects, the following options will be presented:

- **Login with online change:** execute the login and send the new project without stopping the current CPU application (see Run Mode item), updating the changes when a new cycle is executed.
- **Login with download:** execute the login and send the new project with the CPU stopped (see Stop Mode). When the application is initiated, the update will have been done already.
- **Login without any change:** executes the login without sending the new project.

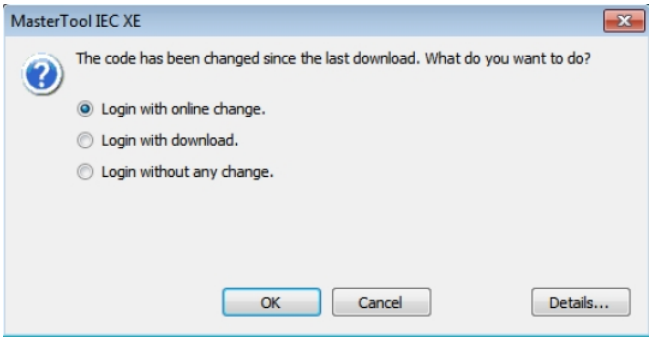



Figure 36: Project Update at the CPU

**ATTENTION**

In the online changes is not permitted to associate symbolic variables mapping from a global variable list (GVL) and use these variables in another global variable list (GVL).

If the new application contains changes on the configuration, the online change will not be possible. In this case, Mastertool requests whether the login must be executed as download (stopping the application) or if the operation must be cancelled.

**PS.:** The button *Details...* shows the changes made in the application.

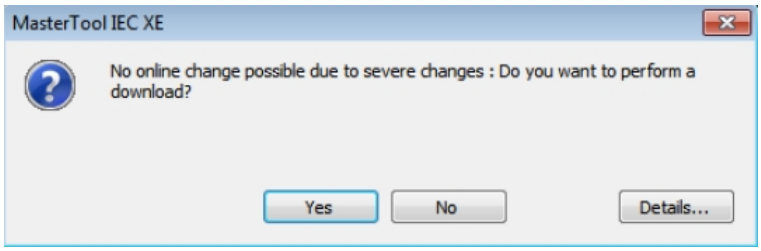



Figure 37: Configuration Changes

Finally, at the end of this process Mastertool offers the option to transfer (download) the source code to the internal memory of the device, as shown on the following figure:

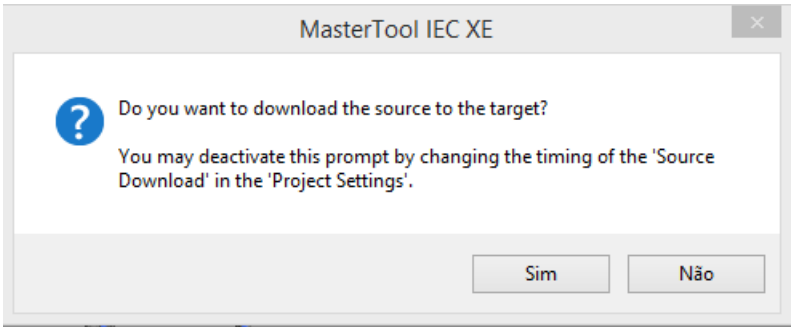



Figure 38: Source code download

Transferring the source code is fundamental to ensure the future restoration of the project and to perform modifications on the application that is loaded into the device.

## 4.8. Run Mode

Right after the project has been sent to the CPU, the application will not be immediately executed (except for the case of an online change). For that to happen, the command Start must be executed. This way, the user can control the execution of the application sent to the CPU, allowing pre-configuring initial values which will be used by the CPU on the first execution cycle.

To execute this command, simply go to the *Debug* menu and select the option *Start*, as shown on figure below.

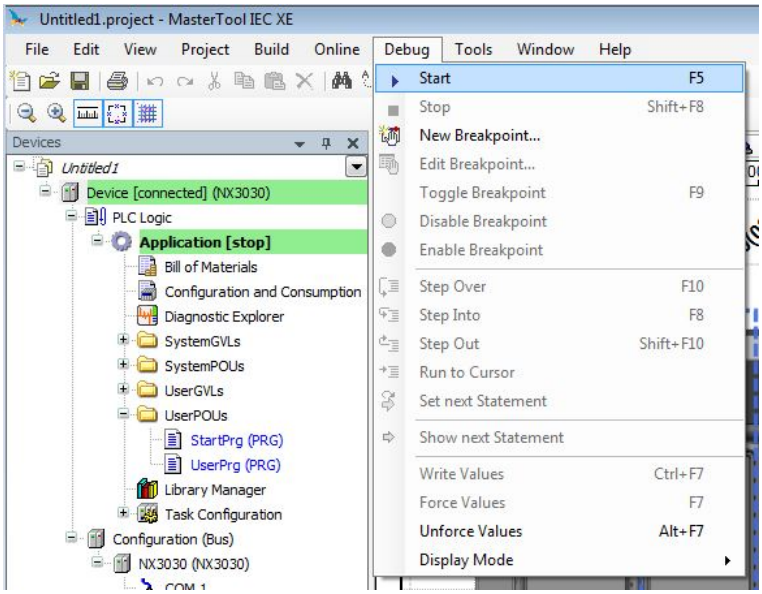



Figure 39: Starting the Application

The figure below shows the application in execution. In case the POU tab is selected, the created variables are listed on a monitoring window, in which the values can be visualized and forced by the user.

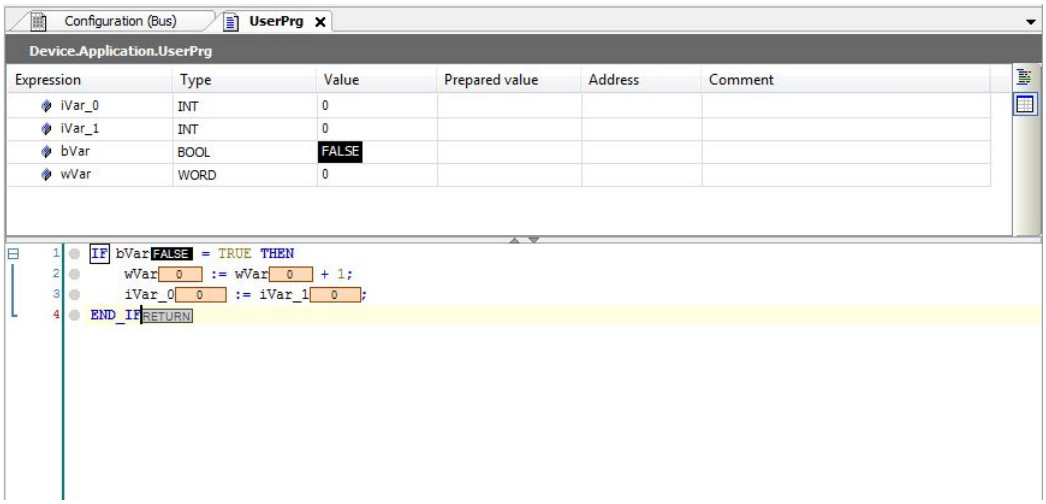



Figure 40: Program running

If the CPU already have a boot application internally stored, it goes automatically to Run Mode when the device is powered on, with no need for an online command through Mastertool.

## 4.9. Stop Mode

To stop the execution of the application, the user must execute the *Stop* command, available at the menu *Debug*, as shown on figure below.

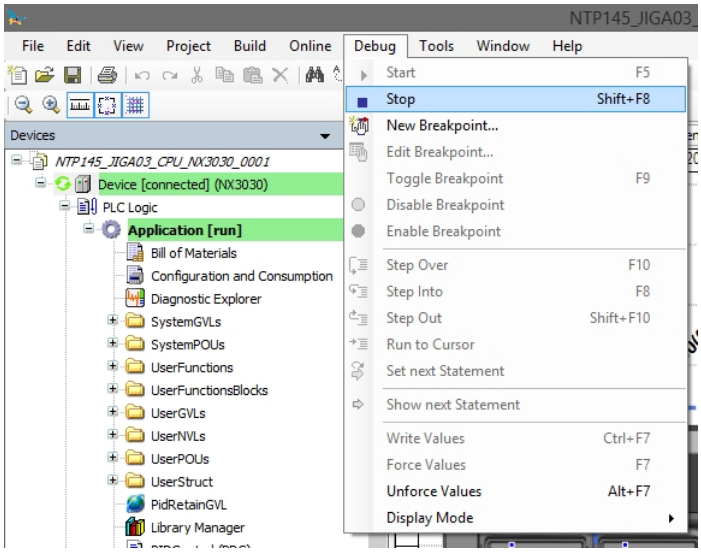



Figure 41: Stopping the Application

In case the CPU is initialized without the stored application, it automatically goes to Stop Mode, as it happens when a software exception occurs.

## 4.10. Writing and Forcing Variables

After Logging into a PLC, the user can write or force values to a variable of the project.

The writing command (*CTRL + F7*) writes a value into a variable and this value could be overwritten by instructions executed in the application.

Moreover, the forced writing command (*F7*) writes a value into a variable without allowing this value to be changed until the forced variables are released.

It is important to highlight that, when used the MODBUS RTU Slave and the MODBUS Ethernet Server, and the *Read-only* option from the configured relations is not selected, the forced writing command (*F7*) must be done over the available variables in the monitoring window as the writing command (*CTRL + F7*) leaves the variables to be overwritten when new readings are done.

**ATTENTION**

The variables forcing can be done in Online mode.
Diagnostic variables cannot be forced, only written, because diagnostics are provided by the CPU and will be overwritten by it.

**ATTENTION**

When a CPU is with forced variables and it is de-energized, the variables will lose the forcing in the next initialization.
The limit of forcing for the Nexto CPUs is 128 variables, regardless of model or configuration of CPU used.

## 4.11. Logout

To finalize the online communication with the CPU, the command *Logout* must be executed, located in the *Online* menu, as shown on figure below.

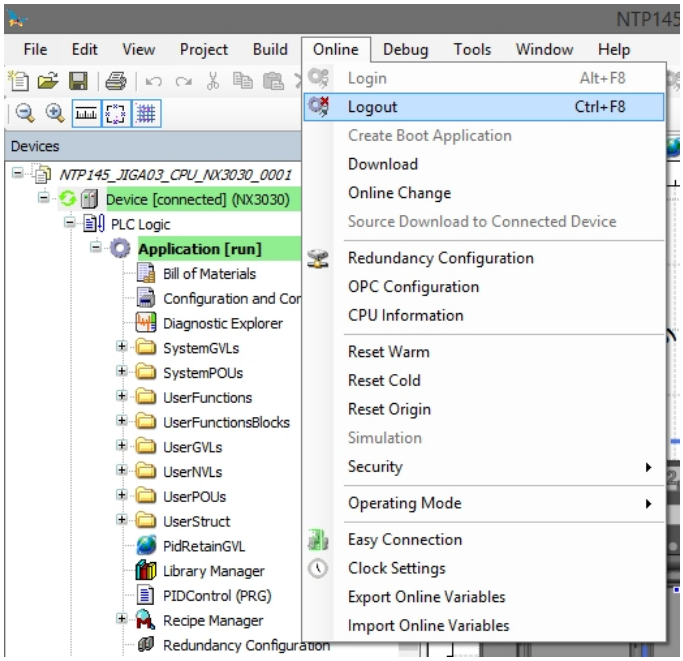



Figure 42: Finalizing the online communication with the CPU

## 4.12. Project Upload

Nexto Series CPUs are capable to store the source code of the application on the internal memory of the device, allowing future retrieval (upload) of the complete project and to modify the application.

To recover a project previously stored on the internal memory of the CPU, the command located on *File* menu must be executed as shown on the following figure.

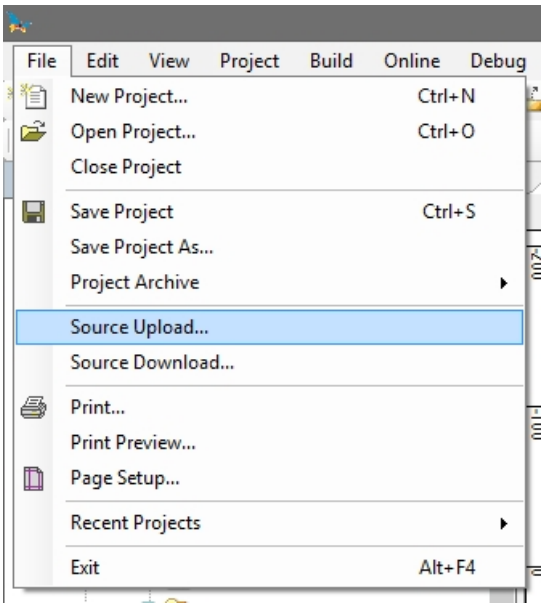



Figure 43: Project Upload Option

Next, just select the desired CPU and click *OK*, as shown on figure below.

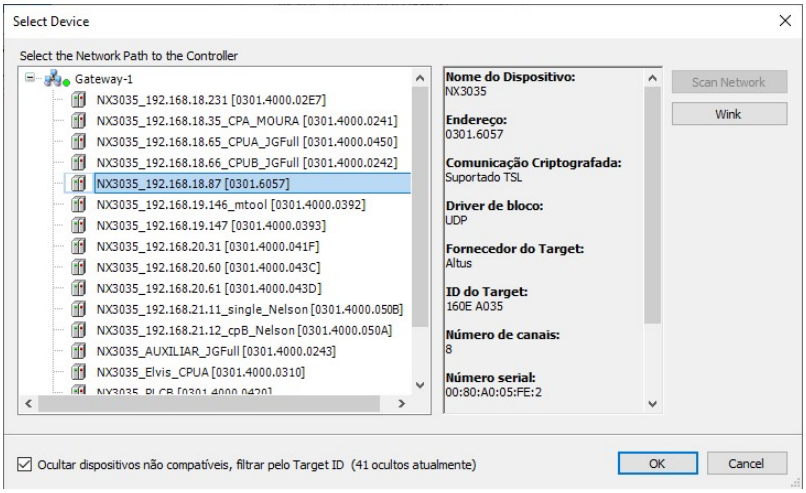



Figure 44: Selecting the CPU

To ensure that the project loaded in the CPU is identical and can be accessed in other workstations, consult the chapter *Projects Download/Login Method without Project Differences* at Mastertool's user manual.

**ATTENTION**

The memory size area to store a project in the Nexto CPUs is defined on section Memory.

**ATTENTION**

The upload recovers the last project stored in the controller as described in the previous paragraphs. In case only the application was downloaded, without transferring its source code, it will not be possible to be recovered by the Upload procedure.

## 4.13. CPU Operating States

### 4.13.1. Run

When a CPU is in *Run* mode, all application tasks are executed.

### 4.13.2. Stop

When a CPU is in *Stop* mode, all application tasks are stopped. The variable values in the tasks are kept with the current value and output points go to the safe state.

When a CPU goes to the *Stop* mode due to the download of an application, the variables in the application tasks will be lost except the persistent variables type.

### 4.13.3. Breakpoint

When a debugging mark is reached in a task, it is interrupted. All the active tasks in the application will not be interrupted, continuing their execution. With this feature, it's possible to go through and investigate the program flow step by step in *Online* mode according to the positions of the interruptions included through the editor.

For further information about the use of breakpoints, please consult Mastertool's user manual.

### 4.13.4. Exception

When a CPU is in *Exception* it indicates that some improper operation occurred in one of the application active tasks. The task which caused the Exception will be suspended and the other tasks will pass for the Stop mode. It is only possible to take off the tasks from this state and set them in execution again after a new CPU start condition. Therefore, only with a *Reset Warm*, *Reset Cold*, *Reset Origin* or a CPU restart puts the application again in Run mode.

#### 4.13.5. Reset Warm

This command puts the CPU in *Stop* mode and initializes all the application tasks variables, except the persistent and retentive type variables. The variables initialized with a specific value will assume exactly this value, the other variables will assume the standard initialization value (zero).

#### 4.13.6. Reset Cold

This command puts the CPU in *Stop* mode and initializes all the application tasks variables, except the persistent type variables. The variables initialized with a specific value will assume exactly this value, the other variables will assume the standard initialization value (zero).

#### 4.13.7. Reset Origin

This command removes all application task variables, including persistent type variables, deletes the application CPU and puts the CPU in Stop mode.

**Notes:**

**Reset:** When a Reset is executed, the breakpoints defined in the application are disabled.

**Command:** To execute the commands *Reset Warm*, *Cold* or *Origin*, it's necessary to have Mastertool in *Online* mode with the CPU.

## 4.14. Programs (POUs) and Global Variable Lists (GVLs)

The project created by Mastertool contains a set of program modules (POUs) and global variables lists that aims to facilitate the programming and utilization of the controller. The following sections describe the main elements that are part of this standard project structure.

#### 4.14.1. MainPrg Program

The MainTask is associated to one unique POU of program type, named MainPrg. The MainPrg program is created automatically and cannot be edited by user.

The MainPrg program code is the following, in ST language:

```
(*Main POU associated with MainTask that calls StartPrg,
 UserPrg/ActivePrg and NonSkippedPrg.
 This POU is blocked to edit.*)

PROGRAM MainPrg
VAR
   isFirstCycle : BOOL := TRUE;
END_VAR

SpecialVariablesPrg();
IF isFirstCycle THEN
   StartPrg();
   isFirstCycle := FALSE;
ELSE
   UserPrg();
END_IF;
```

MainPrg call other two POUs of program type, named *StartPrg* and *UserPrg*. While the *UserPrg* is always called, the *StartPrg* is only called once in the PLC application start.

To the opposite of *MainPrg* program, that must not be modified, the user can change the *StartPrg* and *UserPrg* programs. Initially, when the project is created from the wizard, these two programs are created *empty*, but the user might insert code in them.

### 4.14.2. StartPrg Program

In this POU the user might create logics, loops, start variables, etc. that will be executed only one time in the first PLC's cycle, before execute *UserPrg* POU by the first time. And not being called again during the project execution.

In case the user load a new application, or if the PLC gets powered off, as well as in *Reset Origin*, *Reset Cold* and *Reset Warm* conditions, this POU is going to be executed again.

### 4.14.3. UserPrg Program

In this POU the user must create the main application, responsible by its own process control. This POU is called by the main POU (MainPrg).

The user can also create additional POUs (programs, functions or function blocks), and called them or instance them inside UserPrg POU, to ends of its program instruction. Also it is possible to call functions and instance function blocks defined in libraries.

### 4.14.4. GVL IntegratedIO

The GVL *IntegratedIO* contains the variables correspondent to the fast input and output channels integrated into the controller.

The following picture shows an example of the presentation of this GVL when in *Online* mode.

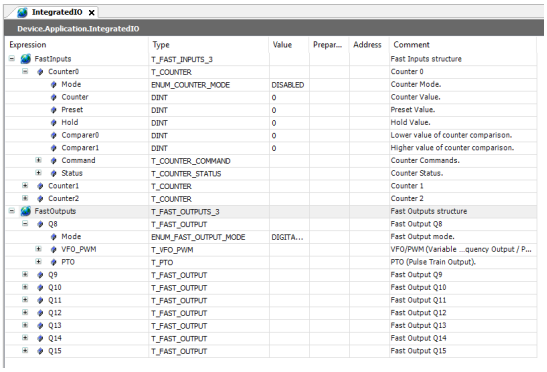



Figure 45: IntegratedIO GVL in Online Mode

### 4.14.5. GVL System_Diagnostics

The *System_Diagnostics* GVL contains the diagnostic variables of the controller's CPU, communication and I/O interfaces. This GVL isn't editable and the variables are declared automatically with type specified by the device to which it belongs when it is added to the project.

The following picture shows an example of the presentation of this GVL when in *Online* mode.

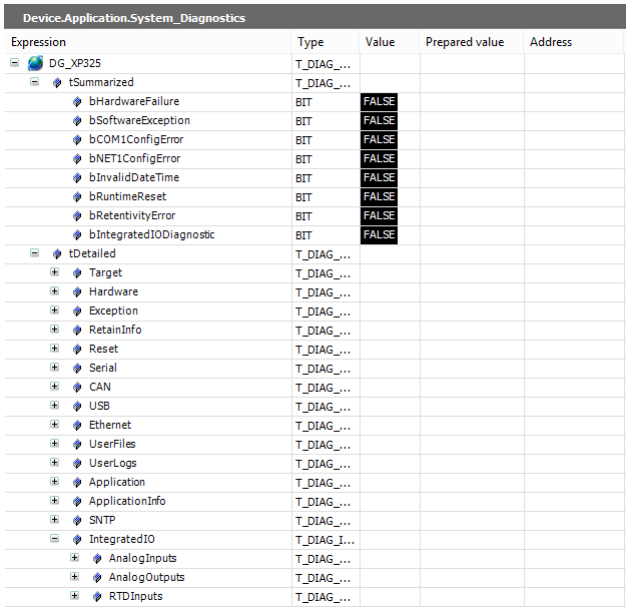



Figure 46: System_Diagnostics GVL in Online Mode

### 4.14.6. GVL Disables

The *Disables* GVL contains the MODBUS Master/Client by symbolic mapping requisition disabling variables. It is not mandatory, but it is recommended to use the automatic generation of these variables, which is done clicking in the button *Generate Disabling Variables* in device requisition tab. These variables are declared as type BOOL and follow the following structure:

Requisition disabling variables declaration:

```
[Device Name]_DISABLE_[Requisition Number] : BOOL;
```

Where:

**Device name:** Name that shows on Tree View to the MODBUS device.

**Requisition Number:** Requisition number that was declared on the MODBUS device requisition table following the sequence from up to down, starting on 0001.

Example:

Device.Application.Disables

```
VAR_GLOBAL
  MODBUS_Device_DISABLE_0001   : BOOL;
  MODBUS_Device_DISABLE_0002   : BOOL;
  MODBUS_Device_DISABLE_0003   : BOOL;
  MODBUS_Device_1_DISABLE_0001   : BOOL;
  MODBUS_Device_1_DISABLE_0002   : BOOL;
END_VAR
```

The automatic generation through button *Generate Disabling Variables* only create variables, and don't remove automatically. This way, in case any relation is removed, its respective disabling variable must be removed manually.

The *Disables* GVL is editable, therefore the requisition disabling variables can be created manually without need of following the model created by the automatic declaration and can be used both ways at same time, but must always be of BOOL type. And it is need to take care to do not delete or change the automatic declared variables, cause them can being used for some MODBUS device. If the variable be deleted or changed then an error is going to be generated while the project is being

compiled. To correct the automatically declared variable name, it must be followed the model exemplified above according to the device and the requisition to which they belong.

The following picture shows an example of the presentation of this GVL when in *Online* mode. If the variable values are TRUE it means that the requisition to which the variables belong is disabled and the opposite is valid when the variable value is FALSE.

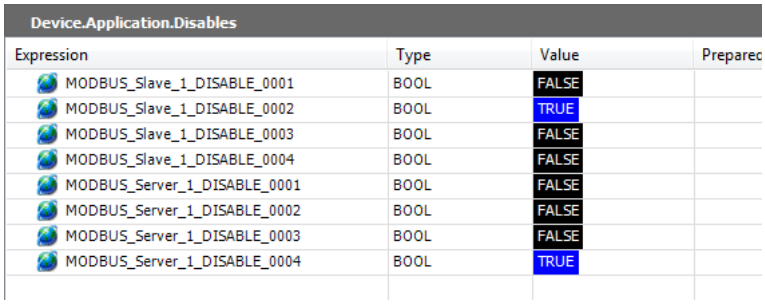



Figure 47: Disable GVL in Online Mode

### 4.14.7. GVL Qualities

The *Qualities* GVL contains the quality variable of the internal variables MODBUS Master/Client of symbolic mapping. It is not mandatory but is recommended to use these variables' automatic generation, what is done clicking on button *Generate Quality Variables* in the device mapping tab. These variables are declared as *LibDataTypes.QUALITY* type and follow the following structure:

Quality mapping variable declaration:

```
[Device Name]_QUALITY_[Mapping Number]: LibDataTypes.QUALITY;
```

Where:

**Device Name:** Name that appear at the Tree View to the device.

**Mapping Number:** Number of the mapping that was declared on the device mapping table, following the up to down sequence, starting with 0001.

**ATTENTION**

It is not possible to associate quality variables to the direct representation MODBUS Master/Client drivers' mappings. Therefore it is recommended the use of symbolic mapping MODBUS drivers.

Example: Device.Application.Qualities

```
VAR_GLOBAL
  MODBUS_Device_QUALITY_0001: LibDataTypes.QUALITY;
  MODBUS_Device_QUALITY_0002: LibDataTypes.QUALITY;
  MODBUS_Device_QUALITY_0003: LibDataTypes.QUALITY;
END_VAR
```

The *Qualities* GVL is editable, therefore the mapping quality variables can be created manually without need to follow the automatic declaration model, and can be used both ways at same time. But must always be of *LibDataTypes.QUALITY* type and take care to don't delete or change a variable automatically declared, because they might being used by some device. If the variable be deleted or changed an error is going to be generated while the project is being compiled. To correct the automatically declared variable name, it must be followed the model exemplified above according to the device and the requisition to which they belong.

The following picture shows an example of the presentation of this GVL when in *Online* mode.

Device.Application.Qualities

| Expression | Type | Value | Address | Comment |
|---|---|---|---|---|
| MODBUS_Slave_1_QUALITY_0001 | LibDataTypes.QUALITY | | | |
| VALIDITY | QUALITY_VALIDITY | VALIDITY_GOOD | | Quality validity |
| FLAGS | QUALITY_FLAGS | | | Quality flags |
| FLAG_OUT_OF_RANGE | BIT | FALSE | | Bit 8 |
| FLAG_INACCURATE | BIT | FALSE | | Bit 9 |
| FLAG_OLD_DATA | BIT | FALSE | | Bit 10 |
| FLAG_FAILURE | BIT | FALSE | | Bit 11 |
| FLAG_OPERATOR_BLOCKED | BIT | FALSE | | Bit 12 |
| FLAG_TEST | BIT | FALSE | | Bit 13 |
| FLAG_RESERVED_0 | BIT | FALSE | | Bit 14 |
| FLAG_RESERVED_1 | BIT | FALSE | | Bit 15 |
| FLAG_RESTART | BIT | FALSE | | Bit 0 |
| FLAG_COMM_FAIL | BIT | FALSE | | Bit 1 |
| FLAG_REMOTE_SUBSTITU... | BIT | FALSE | | Bit 2 |
| FLAG_LOCAL_SUBSTITUTED | BIT | FALSE | | Bit 3 |
| FLAG_FILTER | BIT | FALSE | | Bit 4 |
| FLAG_OVERFLOW | BIT | FALSE | | Bit 5 |
| FLAG_REFERENCE_ERROR | BIT | FALSE | | Bit 6 |
| FLAG_INCONSISTENT | BIT | FALSE | | Bit 7 |
| MODBUS_Slave_1_QUALITY_0002 | LibDataTypes.QUALITY | | | |
| MODBUS_Slave_1_QUALITY_0003 | LibDataTypes.QUALITY | | | |
| MODBUS_Slave_1_QUALITY_0004 | LibDataTypes.QUALITY | | | |
| MODBUS_Server_1_QUALITY_0001 | LibDataTypes.QUALITY | | | |
| MODBUS_Server_1_QUALITY_0002 | LibDataTypes.QUALITY | | | |
| MODBUS_Server_1_QUALITY_0003 | LibDataTypes.QUALITY | | | |
| VALIDITY | QUALITY_VALIDITY | VALIDITY_QUESTIONABLE | | Quality validity |
| FLAGS | QUALITY_FLAGS | | | Quality flags |
| FLAG_OUT_OF_RANGE | BIT | FALSE | | Bit 8 |
| FLAG_INACCURATE | BIT | FALSE | | Bit 9 |
| FLAG_OLD_DATA | BIT | TRUE | | Bit 10 |
| FLAG_FAILURE | BIT | FALSE | | Bit 11 |
| FLAG_OPERATOR_BLOCKED | BIT | FALSE | | Bit 12 |
| FLAG_TEST | BIT | FALSE | | Bit 13 |
| FLAG_RESERVED_0 | BIT | FALSE | | Bit 14 |
| FLAG_RESERVED_1 | BIT | FALSE | | Bit 15 |
| FLAG_RESTART | BIT | FALSE | | Bit 0 |
| FLAG_COMM_FAIL | BIT | TRUE | | Bit 1 |
| FLAG_REMOTE_SUBSTITU... | BIT | FALSE | | Bit 2 |
| FLAG_LOCAL_SUBSTITUTED | BIT | FALSE | | Bit 3 |
| FLAG_FILTER | BIT | FALSE | | Bit 4 |
| FLAG_OVERFLOW | BIT | FALSE | | Bit 5 |
| FLAG_REFERENCE_ERROR | BIT | FALSE | | Bit 6 |
| FLAG_INCONSISTENT | BIT | FALSE | | Bit 7 |
| MODBUS_Server_1_QUALITY_0004 | LibDataTypes.QUALITY | | | |

Figure 48: Qualities GVL in Online Mode

### 4.14.8. GVL ReqDiagnostics

The *ReqDiagnostics* GVL contains the requisition diagnostics variables of symbolic mapping MODBUS Master/Client. It is not mandatory, but recommended the use of these variables' automatic generation, what is done by clicking in the button *Generate Diagnostics Variables* in device requests tab. These variables declaration follow the following structure:

Requisition diagnostic variable declaration:

```
[Device Name]_REQDG_[Requisition Number]: [Variable Type];
```

Where:

**Device Name:** Name that appear at the Tree View to the device.

**Mapping Number:** Number of the mapping that was declared on the device mapping table, following the up to down sequence, starting with 0001.

**Variable Type:** NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS_RTU_MAPPING_1 to MODBUS Master and NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS_ETH_MAPPING_1 to MODBUS Client.

**ATTENTION**

The requisition diagnostics variables of direct mapping MODBUS Master/Client are declared at *System_Diagnostics* GVL.

Example:

Device.Application.ReqDiagnostics

```
VAR_GLOBAL
  MODBUS_Device_REQDG_0001   : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_RTU_MAPPING_1;
  MODBUS_Device_REQDG_0002   : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_RTU_MAPPING_1;
  MODBUS_Device_REQDG_0003   : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_RTU_MAPPING_1;
  MODBUS_Device_1_REQDG_0001  : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_ETH_MAPPING_1;
  MODBUS_Device_1_REQDG_0002  : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_ETH_MAPPING_1;
END_VAR
```

The *ReqDiagnostics* GVL is editable, therefore the requisitions diagnostic variables can be manually created without need to follow the model created by the automatic declaration. Both ways can be used at same time, but the variables must always be of type referring to the device. And take care to don't delete or change a variable automatically declared, because they might being used by some device. If the variable be deleted or changed an error is going to be generated while the project is being compiled. To correct the automatically declared variable name, it must be followed the model exemplified above according to the device and the requisition to which they belong.

The following picture shows an example of the presentation of this GVL when in *Online* mode.

Device.Application.ReqDiagnostics

| Expression | Type | Value |
|---|---|---|
| MODBUS_Slave_1_REQDG_0001 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| byStatus | T_DIAG_MODBUS_RTU_MAPPING_STATUS | |
| eLastErrorCode | MASTER_ERROR_CODE | NO_ERROR |
| eLastExceptionCode | MODBUS_EXCEPTION | NO_EXCEPTION |
| byDiag_3_reserved | BYTE | 0 |
| wCommCounter | WORD | 969 |
| wCommErrorCounter | WORD | 0 |
| MODBUS_Slave_1_REQDG_0002 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| MODBUS_Slave_1_REQDG_0003 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| MODBUS_Slave_1_REQDG_0004 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| MODBUS_Server_1_REQDG_0001 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| MODBUS_Server_1_REQDG_0002 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| MODBUS_Server_1_REQDG_0003 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |
| byStatus | T_DIAG_MODBUS_ETH_MAPPING_STATUS | |
| eLastErrorCode | MASTER_ERROR_CODE | ERR_CONNECTION_TIMEOUT |
| eLastExceptionCode | MODBUS_EXCEPTION | NO_EXCEPTION |
| byDiag_3_reserved | BYTE | 0 |
| wCommCounter | WORD | 116 |
| wCommErrorCounter | WORD | 49 |
| MODBUS_Server_1_REQDG_0004 | NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS... | |

Figure 49: ReqDiagnostics GVL in Online Mode

# 5. Configuration

The Nexto Series CPUs are configured and programmed through Mastertool. The configuration made defines the behavior and utilization modes for peripherals use and the CPUs special features. The programming represents the Application developed by the user.

## 5.1. Device

### 5.1.1. User Management and Access Rights

It provides functions to define users accounts and to configure the access rights to the project and to the CPU. Using Mastertool, it's possible to create and manage users and groups, setting, different access right levels to the project.

Simultaneously, the Nexto CPUs have an user permissions management system that blocks or allows certain actions for each user group in the CPU. For more information, consult Mastertool's user manual, in the User Management and Access Rights section.

### 5.1.2. PLC Settings

On this tab of the generic device editor, you make the basic settings for the configuration of the PLC, for example the handling of inputs and outputs and the bus cycle task.

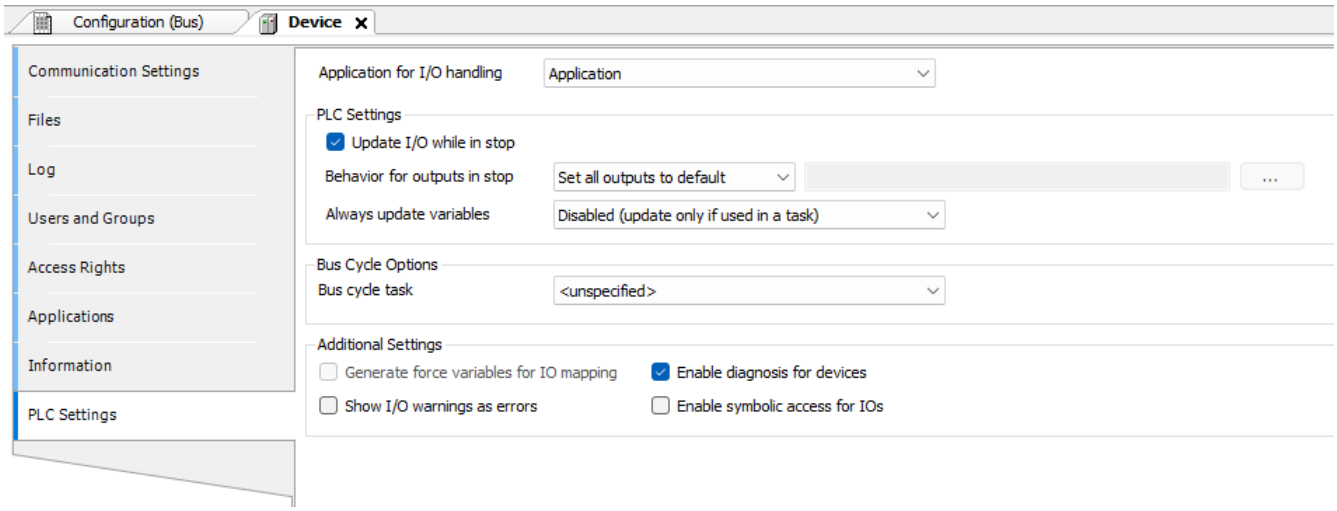



Figure 50: PLC Settings

| Parameter | Description |
|---|---|
| Application for I/O handling | Application that is responsible for the I/O handling. |
| Update I/O while in stop | **TRUE:** The values of the input and output channels are also refreshed when the PLC is in STOP mode. If the watchdog detects a malfunction, the outputs are set to the predefined default values.<br>**FALSE:** The values of the input and output channels in STOP mode are not refreshed. |

| Parameter | Description |
|---|---|
| Behavior for outputs in stop | Handling of the output channels when the controller enters STOP mode:<br>**Retain values:** The current values are retained.<br>**All outputs to default value:** The default values resulting from the I/O mapping are assigned.<br>**Execute program:** The handling of the output values is controlled by a program contained in the project which is executed in STOP mode. Enter the name of the program in the field on the right. |
| Always update variables | Globally defines whether or not the I/O variables are updated in the bus cycle task.<br>This setting is effective for the I/O variables of the slaves and modules only if "deactivated" is defined in their update settings.<br>**Deactivated (update only if used in a task):** The I/O variables are updated only if they are used in a task.<br>**Enabled 1 (use bus cycle task if not used in any task):** The I/O variables in the bus cycle task are updated if they are not used in any other task.<br>**Enabled 2 (always in bus cycle task):** All variables in each cycle of the bus cycle task are updated, regardless of whether they are used and whether they are mapped to an input or output channel. |
| Bus cycle task | Task that controls the bus cycle. By default the task defined by the device description is entered.<br>By default, the bus cycle setting of the superordinate bus device applies (use cycle settings of the superordinate bus). This means that the device tree is searched upwards for the next valid definition of the bus cycle task. |
| Generate force variables for IO mapping | **TRUE:** When compiling the application, two global variables are created for each I/O channel which is mapped to a variable in the I/O Mapping dialog. |
| Enable diagnostics for devices | **TRUE:** The CAA Device Diagnosis library is integrated in the project. An implicit function block is generated for each device. If there is already a function block for the device, then either an extended function block is generated (example: EtherCAT) or another function block instance is added. This then contains a general implementation of the device diagnostics. |
| Show I/O warnings as errors | Warnings concerning the I/O configuration are displayed as errors. |
| Enable symbolic access for IOs | **TRUE:** It allows access to I/O points from the internal symbolic name generated in the device declaration. The symbolic name can be consulted in the *Channel* column on the *Bus I/O Mapping* tab of each device. |

Table 36: PLC Settings

**ATTENTION**

The Nexto (NX), Nexto Jet (NJ), Nexto XF and Xtorm (HX) products do not support the *Enable symbolic access for I/O* parameter.

## 5.2. Controller's CPU

### 5.2.1. General Parameters

The parameters related to the controller's CPU are located at project treeview on item XF3xx just below *Configuration*. Each item must be properly verified for the correct project execution. These parameters are described below:

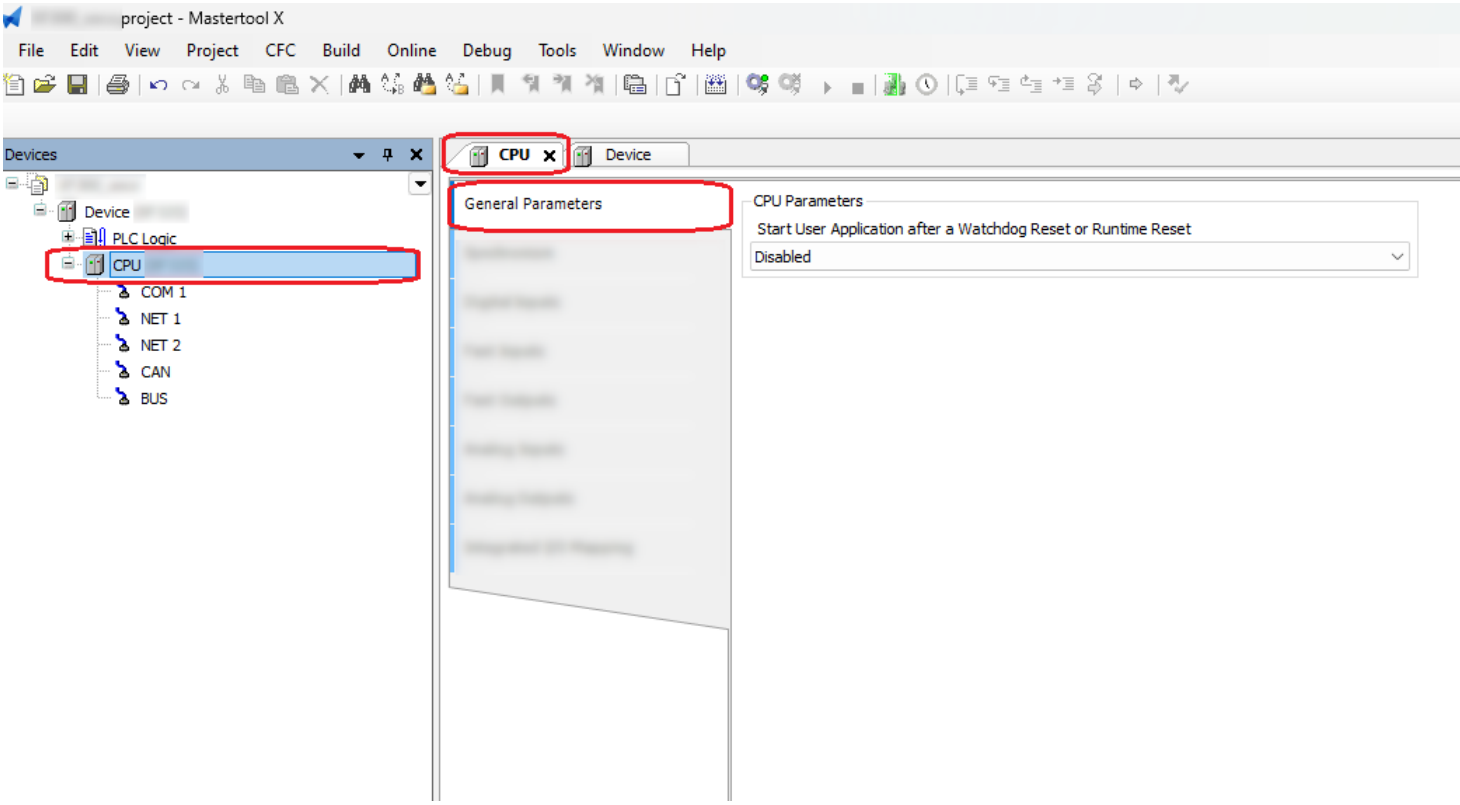



Figure 51: CPU configuration

| Configuration | Description | Default | Options |
|---|---|---|---|
| | **CPU Parameters** | | |
| **Start User Application After a Watchdog Reset** | When enabled starts the user application after the hardware watchdog reset or through the Runtime restart, but keeps the diagnostics indication via LED DG and via variables | Disabled | Enabled<br>Disabled |

Table 37: CPU Configuration

### 5.2.2. Time Synchronization

For the time synchronization, Nexto Series CPUs use the SNTP (*Simple Network Time Protocol*).

To use the time sync protocols, the user must set the following parameters at *Synchronism* tab, accessed through the CPU, in the device tree:

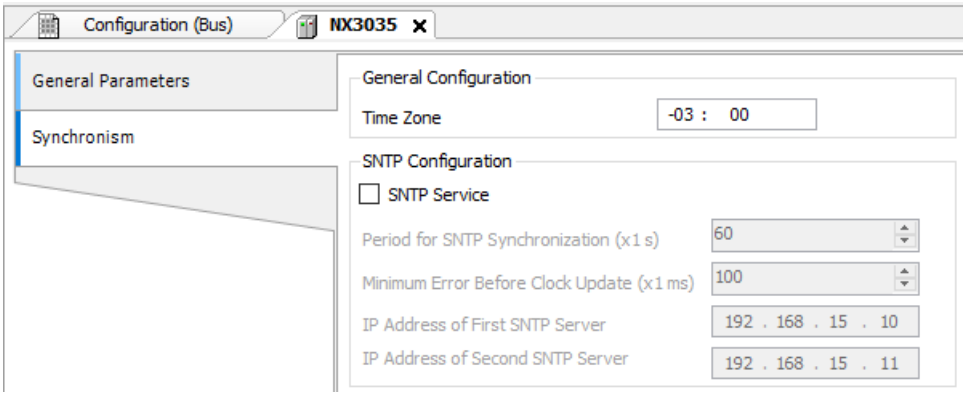



Figure 52: SNTP Configuration

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Time Zone (hh:mm)** | Time zone of the user location. Hours and minutes can be inserted. | -3:00 | -12:59 to +13:59 |
| **SNTP Service** | Enables the SNTP service. | Disabled | Disabled<br>Enabled |
| **Period for SNTP Synchronization (x1 s)** | Time interval of the synchronization requests (seconds). | 60 | 1 to 255 |
| **Minimum Error Before Clock Update (x1 ms)** | Offset value acceptable between the server and client (milliseconds). | 100 | 1 to 65519 |
| **IP Address of First SNTP Server** | IP Address of the primary SNTP server. | 192.168.15.10 | 1.0.0.1 to 223.255.255.254 |
| **IP Address of Second Second SNTP Server** | IP Address of the secondary SNTP server. | 192.168.15.11 | 1.0.0.1 to 223.255.255.254 |

Table 38: SNTP Configurations

**Notes:**

**SNTP Server:** It is possible to define a preferential address and another secondary one in order to access a SNTP server and, therefore, to obtain a synchronism of time. If both fields are empty, the SNTP service will remain disabled.

**Time zone:** The time zone configuration is used to convert the local time into UTC and vice versa. While some sync sources use the local time, others use the UTC time. The UTC time is usually used to stamp events, while the local time is used by an others CPU's features.

It is allowed to enable more than one sync source on the project, however the device doesn't supports the synchronism from more than one sync source during operation. Therefore there are implicitly defined a priority mechanism. The synchronism through SNTP has the higher prioritary. So, when more than one sources is enabled and SNTP server is present, it is going to be responsible for the CPU's clock sync, and any sync command from other source is going to be denied.

#### 5.2.2.1. SNTP

When enabled, the CPU will behave as a SNTP client, which is, it will send requests of time synchronization to a SNTP/NTP server which can be in the local net or in the internet. SNTP client works with a resolution of 1 ms, but with an accuracy of 100 ms. The precision of the time sync through SNTP depends on the protocol configurations (minimum error to clock update) and the features of the Ethernet network where it is, if both client and server are in the same network (local) or in different networks (remote). Typically the precision is in tens of milliseconds order.

The CPU sends the cyclic synchronization requests according to the time set in the *Period for SNTP Synchronization* field. In the first synchronization attempt, just after the service start up, the request is for the first server set in the first server IP address. In case it does not respond, the requests are directed to the second server set in the second server IP address providing

a redundancy of SNTP servers. In case the second server does not respond either, the same process of synchronization attempt is performed again but only after the Period of Synchronization having been passed. In other words, at every synchronization period the CPU tries to connect once in each server, it tries the second server in case the first one does not respond. The waiting time for a response from the SNTP server is defined by default in 5 s and it cannot be modified.

If, after a synchronization, the difference between the current time of the CPU and the one received by the server is higher than the value set in the *Minimum Error Before Clock Update* parameter, the CPU time is updated. SNTP uses the time in the UTC (Universal Time Coordinated) format, so the *Time Zone* parameter needs to be set correctly so the time read by the SNTP will be properly converted to a local time.

The execution process of the SNTP client can be exemplified with the following steps:

1. Attempt of synchronization through the first server. In case the synchronization occurs successfully, the CPU waits the time for a new synchronization (*Period for SNTP Synchronization*) and will synchronize again with this server, using it as a primary server. In case of failure (the server does not respond in less than 5 s) step 2 is performed.
2. Attempt of synchronization through the second server. In case the synchronization occurs successfully, the CPU waits the time for a new synchronization (*Period for SNTP Synchronization*) and will try to synchronize with this server using the primary server. In case of failure (the server does not respond in less than 5 s) the time relative to the Synchronization Period is waited and step 1 is performed again.

As the waiting time for the response of the SNTP server is 5 s, the user must pay attention to lower than 10 s values for the Synchronization Period. In case the primary server does not respond, the time for the synchronization will be the minimum of 5 s (waiting for the primary server response and the synchronization attempt with secondary server). In case neither the primary server nor the secondary one responds, the synchronization time will be 10 s minimum (waiting for the two servers response and the new connection with first server attempt).

Depending on the SNTP server's subnet, the client will use the Ethernet interface that is in the corresponding subnet to make the synchronization requests. If there is no interface configured on the same subnet as the server, the request can be made by any interface that can find a route to the server.

**ATTENTION**

The SNTP Service depends on the user application only for its configuration. Therefore, this service will be executed even when the CPU is in *STOP* or *BREAKPOINT* modes, as long as there is an application in the CPU with the SNTP client enabled and correctly configured.

#### 5.2.2.2. Daylight Saving Time (DST)

The DST configuration must be done indirectly through the function *SetTimeZone*, which changes the time zone applied to the RTC. In the beginning of the DST, it has to be used a function to increase the time zone in one hour. At the end of the DST, it is used to decrease it in one hour.

For further information, see the section RTC Clock.

## 5.3. Serial Interface

### 5.3.1. COM 1

The COM 1 communication interface consists of the D+ and D- terminals, for the half-duplex RS-485 standard, allowing point-to-point or network communication in open protocols MODBUS RTU Slave or MODBUS RTU Master.

When using the MODBUS Master/Slave protocol, some of these parameters (such as *Serial Mode*, *Data Bits*, *RX Threshold* are automatically adjusted by the Mastertool for the correct operation of this protocol.

Below, it follows the parameters that must be configured for the proper operation of the application.

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Serial Type** | Serial channel configuration. | RS-485 | RS-485 |
| **Baud Rate** | Serial communication port speed configuration. | 115200 | 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps |

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Parity** | Serial port parity configuration. | None | Odd<br>Even<br>None |
| **Data Bits** | Sets the data bits quantity in each serial communication character. | 8 | 6, 7 and 8 |
| **Stop Bits** | Sets the serial port stop bits. | 1 | 1 and 2 |
| **Serial Mode** | Sets the serial port operation mode. | Normal Mode | - Extended Mode: Extended operation mode which delivers information regarding the received data frame (see notes below)<br>- Normal Mode: Serial communication normal operation mode |

Table 39: RS-485 Standard Serial Configurations

**Notes:**

**Extended Mode:** This serial communication operation mode provides information regarding the data frame received. The information available is the following:

- One byte for the received data (RX_CHAR : BYTE): Store the six, seven or eight bits from the data received, depending on the serial communication configuration.
- One byte for the signal errors (RX_ERROR : BYTE): It has the format described below:
  - Bit 0: 0 - the character in bits 0 to 7 is valid. 1 - the character in bits 0 to 7 is not valid (or it cannot be valid), due to problems indicated in bits 10 to 15.
  - Bit 1: Not used.
  - Bit 2: Not used.
  - Bit 3: UART interruption error. The serial input remained in logic 0 (space) for a time greater than a character (start bit + data bits + parity bit + stop bits).
  - Bit 4: UART frame error. The logic 0 (space) was read when the first stop bit was expected and it should be logic 1 (mark).
  - Bit 5: UART parity error. The parity bit read is not correct according to the calculated one.
  - Bit 6: UART overrun error. Data was lost during the FIFO UART reading. New characters were received before the later ones were removed. This error will only be indicated in the first character read after the overrun error indication. This means some old data were lost.
  - Bit 7: RX line overrun error. This character was written when the RX line was completed, overwriting the unread characters.
- Two bytes for the timestamp signal (RX_TIMESTAMP : DWORD): Indicates the silence time, within the 0 to 4.294.967.295 interval, using 1 us as base. It saturates in 4.294.967.295 us if the silence time is higher than 4.294.967.295 units. The RX_TIMESTAMP of a character measures the time from a reference which can be any of the three options below:
  - On most of the cases, the end of the later character.
  - Serial port configuration.
  - The end of serial communication using the SERIAL_TX FB, in other words, when the last character is sent on line.

  Besides measuring the silence between characters, the RX_TIMESTAMP is also important as it measures the silence time of the last character on the RX line. The silence measuring is important for the correct protocol implementation, as MODBUS RTU, for example. This protocol specifies an inter-frame greater than 3.5 characters and an inter-byte less than 1.5 characters.

### 5.3.2. Advanced Configurations

The advanced configurations are related to the serial communication control, in other words, when it is necessary the utilization of a more accurate data transmission and reception control.

| Configuration | Description | Default | Options |
|---|---|---|---|
| **UART RX Threshold** | Bytes quantity which must be received for a new UART interruption to be generated. Low values make the TIMESTAMP more precise when the EXTENDED MODE is used and minimizes the overrun errors. However, values too low may cause several interruptions delaying the CPU. | 8 | 1, 4, 8 and 14 |

Table 40: RS-485 Standard Serial Advanced Configurations

## 5.4. Ethernet Interface

The interfaces are composed by a RJ45 communication connector 10/100Base-TX standard. It allows the point to point or network communication in the following open protocols, for example: MODBUS TCP Client, MODBUS RTU via TCP Client, MODBUS TCP Server and MODBUS RTU via TCP Server.

Below are the IP addressing parameters that must be configured for the proper functioning of the application.

### 5.4.1. NET 1

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Obtain an IP address automatically** | Enables the DHCP Client functionality on the device for automatic IP assignment | Unmarked | Marked or Unmarked |
| **IP Address** | IP address of the port on the Ethernet network | 192.168.15.1 | 1.0.0.1 to 223.255.255.254 |
| **Subnetwork Mask** | Subnet mask of the port on the Ethernet network | 255.255.255.0 | 128.0.0.0 to 255.255.255.252 |
| **Gateway Address** | Controller Gateway address of the port on the Ethernet network | 192.168.15.253 | 0.0.0.0 to 223.255.255.254 |

Table 41: NET 1 Configuration

### 5.4.2. NET 2

| Configuration | Description | Default | Options |
|---|---|---|---|
| **IP Address** | IP address of the port on the Ethernet network | 192.168.16.1 | 1.0.0.1 to 223.255.255.254 |
| **Subnetwork Mask** | Subnet mask of the port on the Ethernet network | 255.255.255.0 | 128.0.0.0 to 255.255.255.252 |

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Gateway Address** | Gateway address of the port on the Ethernet network | 192.168.16.253 | 0.0.0.0 to 223.255.255.254 |

Table 42: NET 2 Configuration

### 5.4.3. Configuration of Internal Ethernet Interfaces

#### 5.4.3.1. Single Mode

In this mode, the interface operates as an independent Ethernet port, with no relation to the subsequent interface.

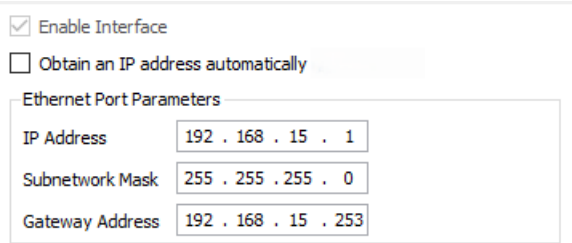



Figure 53: Advanced Configuration of Local Ethernet Interfaces - Single Mode

### 5.4.4. Reserved TCP/UDP Ports

The following TCP/UDP ports of the Ethernet interfaces, both integrated and remote, are used by CPU services (depending on availability according to table Protocols) and, therefore, are reserved and must not be used by the user.

| Service | TCP | UDP |
|---|---|---|
| System Web Page | 80 | - |
| SNTP | - | 123 |
| SNMP | - | 161 |
| MODBUS TCP | 502* | - |
| Mastertool | 1217* | 1740:1743 |
| SQL Server | 1433 | - |
| MQTT | 1883* / 8883* | - |
| EtherNet/IP | 44818 | 2222 |
| IEC 60870-5-104 | 2404* | - |
| IEC 61850 | 102* | - |
| DNP3 | 20000* / 20005* | - |
| OPC UA | 4840 | - |
| WEBVISU | 8080 | - |
| CODESYS ARTI | 11740 | - |
| PROFINET | - | 34964 |
| Portainer Docker | 9000 | - |
| SysLog | - | 514 |
| LibHART | 1234 | - |

Table 43: Reserved TCP/UDP ports

* Default port, but user changeable.

### 5.4.5. Network Routing

The CPU operating system has a network routing mechanism that allows the communication between different networks:

1. Client protocols, when configured to communicate with a Server device located in a different network, will resolve the routing by the following priority:
   - VPN (if this feature is enabled and the network of NET interface of Client protocol is configured as a *Private Network* on the VPN Server settings)
   - USB Devices (Modem, Ethernet or Wifi)
   - Another integrated NET interface configured to the same network as the destination address (in this case, the communication will route internally)

   If none of these conditions occurs, the operating system will forward the communication to the Default Gateway configured on the NET interfaces, starting from the one configured on NET1 and then going through the next ones if the communication fails.

2. Server protocols will accept connections only on the interface where it is instaciated, except for MODBUS, where a parameter called "Allow Connections from Any Interface" gives the ability to receive incoming connections from other interfaces like a USB Modem or VPN.

## 5.5. Controller Area Network Interface

### 5.5.1. CAN

The CAN interface allows point to point or network communication with other devices that have this interface using CANopen application protocol.

The parameters of CAN interface which must be configured for the proper functioning of the application are described below:

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Network** | CAN interface ID number | 0 | 0 (fixed) |
| **Baudrate** | CAN Bus baudrate (kbit/s). The other devices must to use the same baudrate. | 250 | 10, 20, 50, 100, 125, 250, 500, 800, 1000 |

Table 44: CAN Configuration

The parameters related to CANopen protocol are described on CANopen Manager section.

## 5.6. Integrated I/O

Nexto XF controllers have integrated I/O points, which allows it to interface with external devices like sensors, actuators, step motors, encoders, etc...

There are two objects on project treeview related to Integrated I/O, as shown on the figure below:

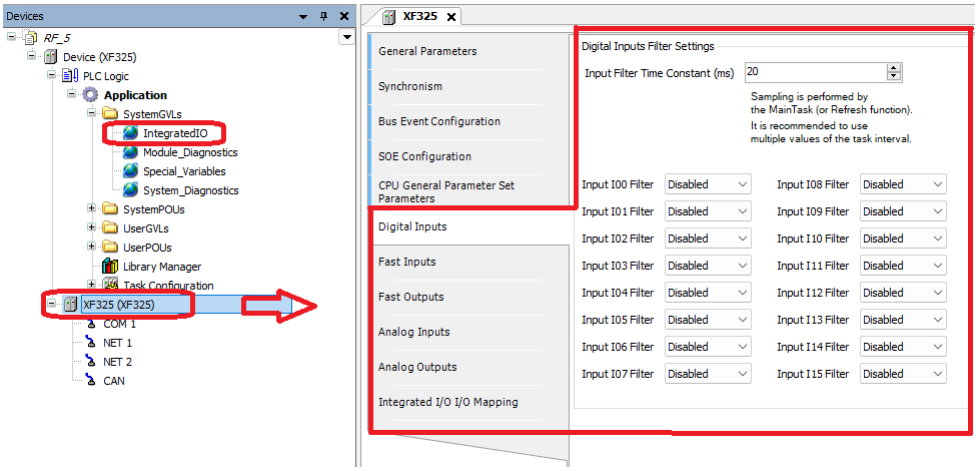



Figure 54: Integrated I/O objects on project treeview

One of these objects is the GVL *IntegratedIO*, which is created automatically by MasterTool and contains a list of global symbolic variables that are directly mapped to the onboard fast inputs and fast outputs.

### 5.6.1. Digital Inputs

The parameters related to the *Digital Inputs* are located on the screen below (example from XF325), for both standard and fast I/O (when configured as standard digital inputs):

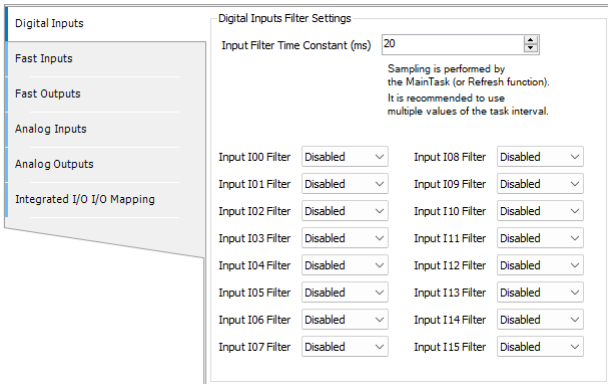



Figure 55: Digital Inputs Parameters

The table below shows the possible configuration values:

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Filter time** | Minimum time that an input must remain in a given state to confirm the state change | 20 ms | 2 to 255 ms |
| **Filter** | Enable/Disable filter for each input | Disabled | Enabled or Disabled |

Table 45: Digital Inputs Parameters

**Note:**

**Input Filter Time Constant:** The filter sampling is performed on MainTask, then it's recommended to use multiple values of the task interval.

**ATTENTION**

The standard digital inputs from I00 to I06 have an update response time of 2ms. Thus, the user may observe one Main Task cycle delay in these inputs.

### 5.6.2. Fast Inputs

The fast inputs are special input signals that can be used for special high-speed functions. These special physical inputs can be assigned to two types of logical elements: high-speed counters and external interruption. Each of these logical elements consumes a certain amount of fast inputs signals. For the high-speed counters unit, it depends on the selected mode (Up/Down, Quadrature, etc...), while each external interruption uses one fast input signal. The number of physical fast inputs, as well as the maximum number of high-speed counter and external interruption logical elements assignable for these inputs is described on Technical Description section.

The following table shows how each high-speed counter unit is assigned to the fast inputs signals:

| High-Speed Counter | Counter Mode | Fast Inputs | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | I07 | I08 | I09 | I10 | I11 | I12 | I13 | I14 | I15 |
| **Counter 0** | Up/Down (A count, B direction) with zero | A | B | Z | - | - | - | - | - | - |
| | Quadrature 2X | A | B | - | - | - | - | - | - | - |
| | Quadrature 2X with zero | A | B | Z | - | - | - | - | - | - |
| | Quadrature 4X | A | B | - | - | - | - | - | - | - |
| | Quadrature 4X with zero | A | B | Z | - | - | - | - | - | - |
| **Counter 1** | Up/Down (A count, B direction) with zero | - | - | - | A | B | Z | - | - | - |
| | Quadrature 2X | - | - | - | A | B | - | - | - | - |
| | Quadrature 2X with zero | - | - | - | A | B | Z | - | - | - |
| | Quadrature 4X | - | - | - | A | B | - | - | - | - |
| | Quadrature 4X with zero | - | - | - | A | B | Z | - | - | - |
| **Counter 2** | Up/Down (A count, B direction) with zero | - | - | - | - | - | - | A | B | Z |
| | Quadrature 2X | - | - | - | - | - | - | A | B | - |
| | Quadrature 2X with zero | - | - | - | - | - | - | A | B | Z |
| | Quadrature 4X | - | - | - | - | - | - | A | B | - |
| | Quadrature 4X with zero | - | - | - | - | - | - | A | B | Z |

Table 46: High-Speed Counters and Fast Inputs allocation

For each configuration described above, the remaining fast input signals (not used by the high-speed counter units) can be used as external interruption, respecting the maximum number of this kind of logical element specified on Technical Description section.

The configuration of high-speed counters and interruptions is located on the following screen:

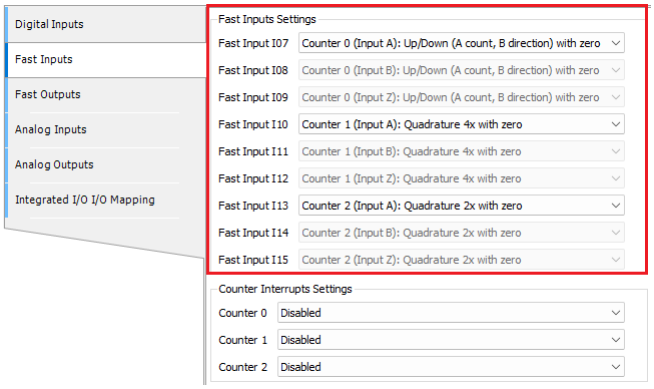



Figure 56: Fast Inputs settings

When selecting the function of a fast input, the following inputs are automatically assigned (locked for edition) according to the mode of the high-speed counter unit.

The table below shows the possible configuration values for each fast input:

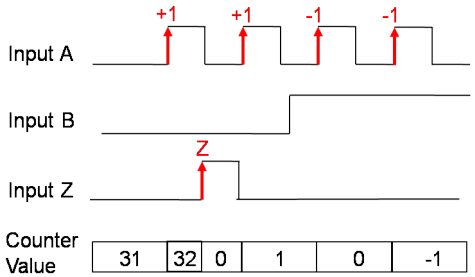
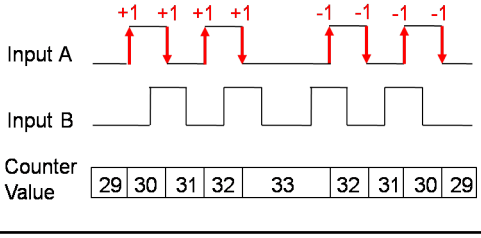
| Configuration | Description | Default | Options |
|---|---|---|---|
| **Fast Input I07** | Fast Input I07 configuration | Digital Input | Digital Input<br>Counter 0 (Input A): Up/Down (A count, B direction) with zero<br>Counter 0 (Input A): Quadrature 2X<br>Counter 0 (Input A): Quadrature 2X with zero<br>Counter 0 (Input A): Quadrature 4X<br>Counter 0 (Input A): Quadrature 4X with zero<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge) |
| **Fast Input I08** | Fast Input I08 configuration | Digital Input | Digital Input<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge)<br><br>Obs: This field will be set automatically when Fast Input I07 is configured as Up/Down or Quadrature Counter. |
| **Fast Input I09** | Fast Input I09 configuration | Digital Input | Digital Input<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge)<br><br>Obs: This field will be set automatically when Fast Input I07 is configured as Up/Down or Quadrature Counter with zero. |
| **Fast Input I10** | Fast Input I10 configuration | Digital Input | Digital Input<br>Counter 1 (Input A): Up/Down (A count, B direction) with zero<br>Counter 1 (Input A): Quadrature 2X<br>Counter 1 (Input A): Quadrature 2X with zero<br>Counter 1 (Input A): Quadrature 4X<br>Counter 1 (Input A): Quadrature 4X with zero<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge) |
| **Fast Input I11** | Fast Input I11 configuration | Digital Input | Digital Input<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge)<br><br>Obs: This field will be set automatically when Fast Input I10 is configured as Up/Down or Quadrature Counter. |
| **Fast Input I12** | Fast Input I12 configuration | Digital Input | Digital Input<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge)<br><br>Obs: This field will be set automatically when Fast Input I10 is configured as Up/Down or Quadrature Counter with zero. |
| | | | Digital Input |

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Fast Input I13** | Fast Input I13 configuration | Digital Input | Counter 2 (Input A): Up/Down (A count, B direction) with zero<br>Counter 2 (Input A): Quadrature 2X<br>Counter 2 (Input A): Quadrature 2X with zero<br>Counter 2 (Input A): Quadrature 4X<br>Counter 2 (Input A): Quadrature 4X with zero<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge) |
| **Fast Input I14** | Fast Input I14 configuration | Digital Input | Digital Input<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge)<br><br>Obs: This field will be set automatically when Fast Input I13 is configured as Up/Down or Quadrature Counter. |
| **Fast Input I15** | Fast Input I15 configuration | Digital Input | Digital Input<br>Interruption (Rising Edge and Falling Edge)<br>Interruption (Falling Edge)<br>Interruption (Rising Edge)<br><br>Obs: This field will be set automatically when Fast Input I13 is configured as Up/Down or Quadrature Counter with zero. |

Table 47: Fast Inputs Parameters

#### 5.6.2.1. High-Speed Counters

The high-speed counter units have multiple operating modes. The following table describes the details of each of these modes:

| Counter Mode | Counting waveforms |
|---|---|
| **Up/Down (A count, B direction) with zero** |  |
| **Quadrature 2X** |  |

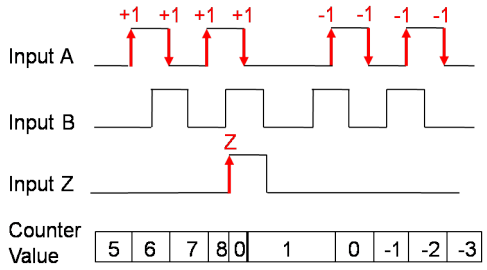
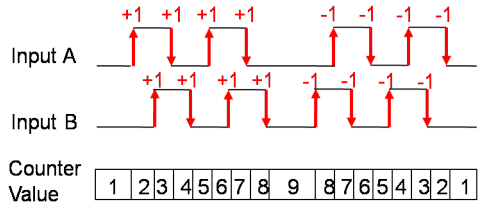
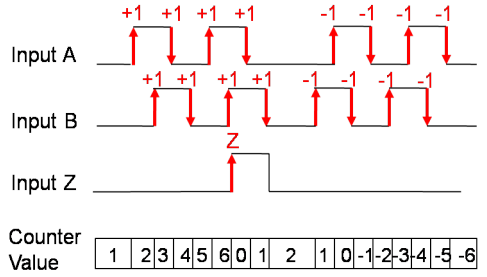
| Counter Mode | Counting waveforms |
|---|---|
| **Quadrature 2X with zero** |  |
| **Quadrature 4X** |  |
| **Quadrature 4X with zero** |  |

Table 48: High-speed counter modes

The overall behavior is the same for all counters: when counting UP and the maximum positive value is reached, the next value will be the minimum negative value. The same thing happens for the oposite direction, so when counting DOWN and the minimum negative value is reached, the next value will be the maximum positive value.

The user program can access the high-speed counters through the *FastInputs* symbolic structure, which is automatically created on *IntegratedIo* GVL. For each high-speed counter unit, there are 3 main areas as shown on the following figure:

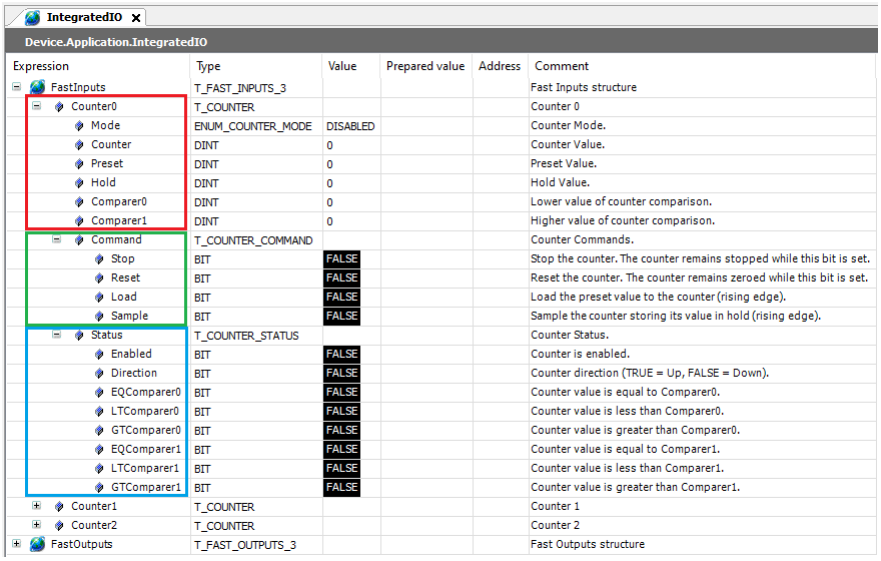



Figure 57: Counter structure

The table below describes the counter variables structure:

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Mode** | Configured counter mode (read only) | ENUM_COUNTER _MODE | DISABLED<br>UP_DOWN_A_COUNT_B _DIR_WITH_ZERO<br>QUADRATURE_2X<br>QUADRATURE_2X _WITH_ZERO<br>QUADRATURE_4X<br>QUADRATURE_4X _WITH_ZERO |
| **Counter** | Counter value | DINT | -2147483648 to 2147483647 |
| **Preset** | Preset value | DINT | -2147483648 to 2147483647 |
| **Hold** | Hold value | DINT | -2147483648 to 2147483647 |
| **Comparer0** | Lower value of counter comparison | DINT | -2147483648 to 2147483647 |
| **Comparer1** | Higher value of counter comparison | DINT | -2147483648 to 2147483647 |
| **Command** | Counter commands structure | T_COUNTER _COMMAND | - |
| **Status** | Counter status structure | T_COUNTER _STATUS | - |

Table 49: Counter structure variables

The command and status are structures of bits that allow the user program to control the counter operation. The following table describes the counter command structure.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Stop** | Stop the counter. The counter remains stopped while this bit is set | BIT | FALSE or TRUE |
| **Reset** | Reset the counter. The counter remains zeroed while this bit is set | BIT | FALSE or TRUE |

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Load** | Load the preset value to the counter value. This operation is performed on rising edge of this bit | BIT | FALSE or TRUE |
| **Sample** | Sample the counter storing its value in hold. This operation is performed on rising edge of this bit | BIT | FALSE or TRUE |

Table 50: Counter command structure

The following table describes the counter status structure.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Enabled** | Counter is enabled | BIT | FALSE or TRUE |
| **Direction** | Counter direction (TRUE = Up, FALSE = Down) | BIT | FALSE or TRUE |
| **EQComparer0** | Counter value is equal to Comparer0 | BIT | FALSE or TRUE |
| **LTComparer0** | Counter value is less than Comparer0 | BIT | FALSE or TRUE |
| **GTComparer0** | Counter value is greater than Comparer0 | BIT | FALSE or TRUE |
| **EQComparer1** | Counter value is equal to Comparer1 | BIT | FALSE or TRUE |
| **LTComparer1** | Counter value is less than Comparer1 | BIT | FALSE or TRUE |
| **GTComparer1** | Counter value is greater than Comparer1 | BIT | FALSE or TRUE |

Table 51: Counter status structure

Additionally to the *IntegratedIo* global variables, there is a function block from *LibIntegratedIo* library which allows to instantiate high-speed counter in POUs written in graphical languages (e.g Ladder Logic Diagram). This function block is, actually, a wrapper to the structured variables described before. The figure below shows the function block instantiated in a Ladder program.

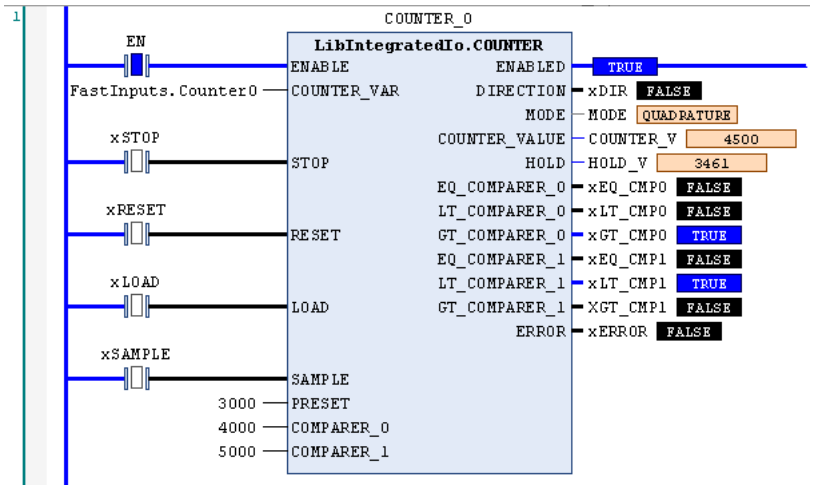



Figure 58: LibIntegratedIo.COUNTER function block

The table below describes the inputs and outputs variables of the function block.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **ENABLE** | Enable the function block execution | BOOL | FALSE or TRUE |
| **COUNTER_VAR** | Counter variable | REFERENCE TO T_COUNTER | FastInputs.Counter0 |
| **STOP** | Stop the counter | BOOL | FALSE or TRUE |
| **RESET** | Reset the counter | BOOL | FALSE or TRUE |
| **LOAD** | Load the preset value to the counter value | BOOL | FALSE or TRUE |
| **SAMPLE** | Sample counter storing its value in hold | BOOL | FALSE or TRUE |
| **PRESET** | Preset value | DINT | -2147483648 to 2147483647 |
| **COMPARER_0** | Lower value of counter comparison | DINT | -2147483648 to 2147483647 |
| **COMPARER_1** | Higher value of counter comparison | DINT | -2147483648 to 2147483647 |
| **ENABLED** | Counter is enabled | BOOL | FALSE or TRUE |
| **DIRECTION** | Counter direction (TRUE = Up, FALSE = Down) | BOOL | FALSE or TRUE |
| **Mode** | Counter mode | ENUM_ COUNTER_MODE | DISABLED UP_DOWN_A_- COUNT_ B_DIR_- WITH_ZERO QUADRATURE_2X QUADRATURE_2X_- WITH_ZERO QUADRATURE_4X QUADRATURE_4X_- WITH_ZERO |
| **COUNTER_VALUE** | Counter value | DINT | -2147483648 to 2147483647 |
| **HOLD** | Hold value | DINT | -2147483648 to 2147483647 |
| **EQ_COMPARER_0** | Counter value is equal to Comparer0 | BOOL | FALSE or TRUE |
| **LT_COMPARER_0** | Counter value is less than Comparer0 | BOOL | FALSE or TRUE |
| **GT_COMPARER_0** | Counter value is greater than Comparer0 | BOOL | FALSE or TRUE |
| **EQ_COMPARER_1** | Counter value is equal to Comparer1 | BOOL | FALSE or TRUE |
| **LT_COMPARER_1** | Counter value is less than Comparer1 | BOOL | FALSE or TRUE |
| **GT_COMPARER_1** | Counter value is greater than Comparer1 | BOOL | FALSE or TRUE |
| **ERROR** | Error occurred in function block execution. Can be caused by invalid COUNTER_VAR or counter disabled. | BOOL | FALSE or TRUE |

Table 52: LibIntegratedIo.COUNTER function block description

#### 5.6.2.1.1. Counter Interrupts

The high-speed counter units have the ability to generate interrupts by comparison, i.e., when the counter reach a certain comparison value, an specific task will run and interrupt the main program execution. Each high-speed counter unit have two comparison values, called *Comparer0* and *Comparer1*, which are present on the corresponding global symbolic data structure or FunctionBlock as described on previous sections. The configuration of counter interrupt for each high-speed counter unit is located on the following screen:

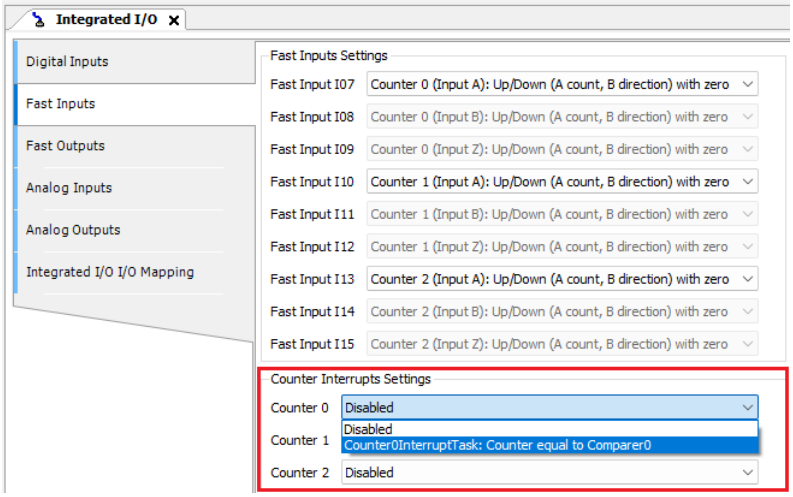



Figure 59: Counter interrupt settings

The table below shows the possible configuration values for the counter interrupt:

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Counter 0** | Counter0 comparator interrupt configuration | Disabled | Disabled<br>Counter0InterruptTask: Counter equal to Comparer0<br><br>Obs: This configuration is available when the Counter0 is associated to some Fast Input. |
| **Counter 1** | Counter1 comparator interrupt configuration | Disabled | Disabled<br>Counter1InterruptTask: Counter equal to Comparer0<br><br>Obs: This configuration is available when the Counter1 is associated to some Fast Input. |
| **Counter 2** | Counter2 comparator interrupt configuration | Disabled | Disabled<br>Counter2InterruptTask: Counter equal to Comparer0<br><br>Obs: This configuration is available when the Counter2 is associated to some Fast Input. |

| Configuration | Description | Default | Options |
|---|---|---|---|

Table 53: Counter interrupt parameters

The counter interrupt will generate an specific event. This event must trigger the execution of external event task, which must call an specific POU. For example, the comparison event generated for Counter 0 is called *COUNTER0_EVT*. So, an external event task called *Counter0InterruptTask* must be configured to be triggered by this event, and must call a POU called *Counter0InterruptPrg* which will contain the user program to be executed.

The figure below shows this configuration scenario in Mastertool.

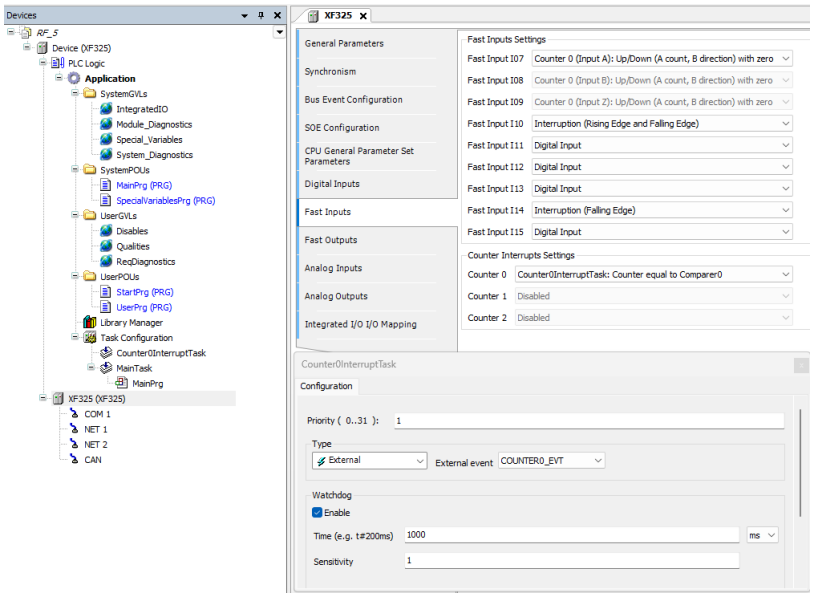



Figure 60: Counter Interrupt Settings

### 5.6.2.2. External Interruption

The fast inputs can be set in three different interuption modes: *Interruption (Rising Edge)* mode, *Interruption (Falling Edge)* mode, or *Interruption (Rising Edge and Falling Edge)* mode. Which means that when the chosen edge (0V <–> 24V transition) is performed on the input, a specific task will run and interrupt the main program execution.

Each external interruption will generate an specific event. This event must trigger the execution of external event task, which must call an specific POU. For example, the external interruption event generated for fast input I12 is called *FIN12_EVT*. So, an external event task called *FastInputI12InterruptTask* must be configured to be triggered by this event, and must call a POU called *FastInputI12InterruptPrg*, which will contain the user program to be executed.

The figure below shows this configuration scenario in MasterTool.

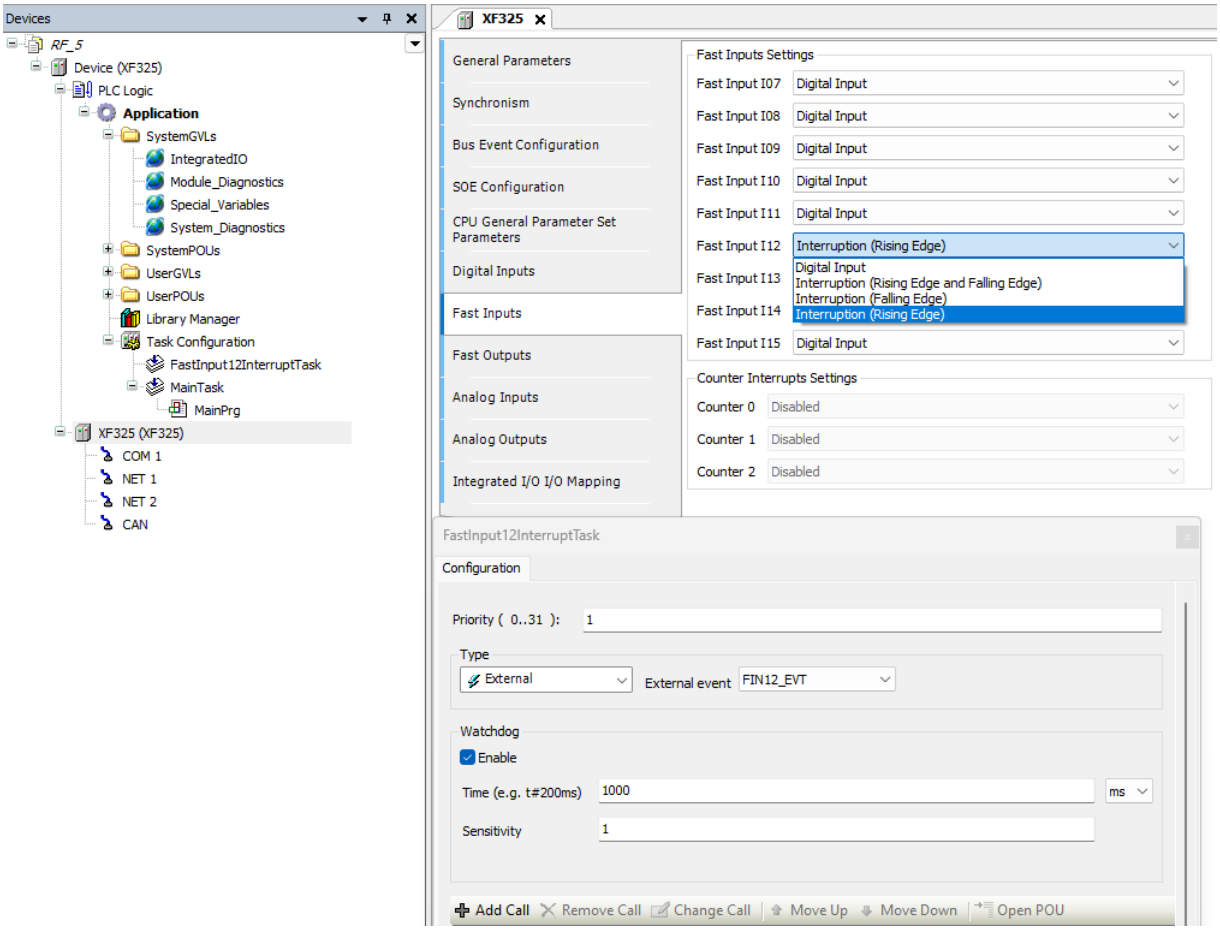



Figure 61: Fast Inputs Interruption Settings

**ATTENTION**

The external interruption input have a 10ms time window filter to protect the controller against spurious transitions on the input signal. This window starts right after the occurance of the interruption and, during this time, any other external interruption event will be discarded.

**ATTENTION**

The external interruption does not supports reentrancy. If another interruption occurs (after the filter time) and its program execution is still not finished, this interruption will be discarded.

### 5.6.3. Fast Outputs

The fast outputs are special output signals that can be used for pulse generator outputs. These special physical outputs can be assigned to two types of logical elements: VFO/PWM (variable frequency/pulse width) and PTO (pulse train output). Each of these logical elements consumes one fast output signal each one. The number of physical fast outputs, as well as the maximum number of the VFO/PWM and PTO logical elements assignable to these outputs is described on Technical Description section.

The configuration of fast outputs is located on the following screen:

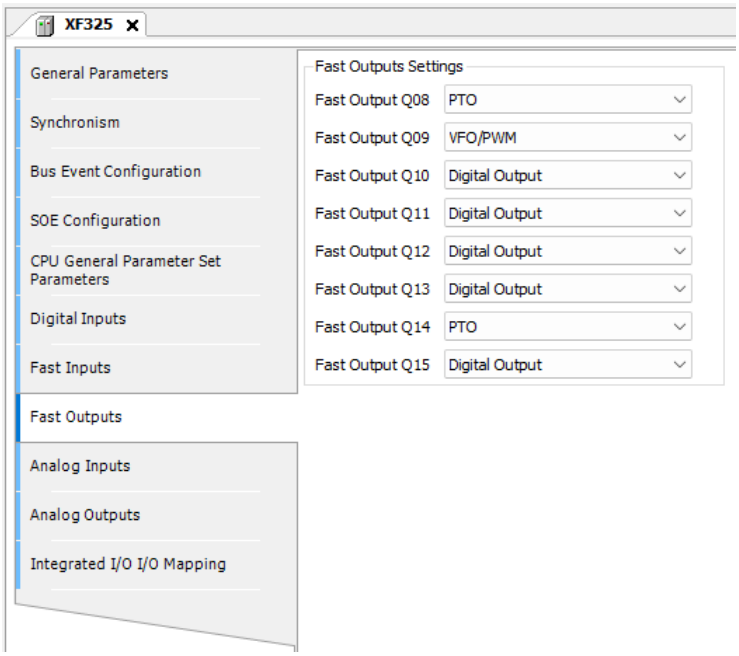



Figure 62: Fast Outputs Parameters

The table below shows the possible configuration values:

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Fast Output Q08** | Fast Output Q08 configuration. | Digital Output | Digital Output<br>VFO/PWM<br>PTO |
| **Fast Output Q09** | Fast Output Q09 configuration. | Digital Output | Digital Output<br>VFO/PWM<br>PTO |
| **Fast Output Q10** | Fast Output Q10 configuration. | Digital Output | Digital Output<br>VFO/PWM |
| **Fast Output Q11** | Fast Output Q11 configuration. | Digital Output | Digital Output<br>VFO/PWM |
| **Fast Output Q12** | Fast Output Q12 configuration. | Digital Output | Digital Output<br>VFO/PWM |
| **Fast Output Q13** | Fast Output Q13 configuration. | Digital Output | Digital Output<br>VFO/PWM |
| **Fast Output Q14** | Fast Output Q14 configuration. | Digital Output | Digital Output<br>VFO/PWM<br>PTO |
| **Fast Output Q15** | Fast Output Q15 configuration. | Digital Output | Digital Output |

| Configuration | Description | Default | Options |
|---|---|---|---|
| | | | VFO/PWM |

Table 54: Fast Outputs Parameters

As shown on the previous table, the fast outputs can be configured as standard digital output. In this case, its digital value can be set using the standard variable mapped in *IntegratedIo*.

When configured as VFO/PWM or PTO, the user program can control the fast outputs through the *FastOutputs* symbolic structure, which is automatically created on IntegratedIo GVL as shown on the following figure:

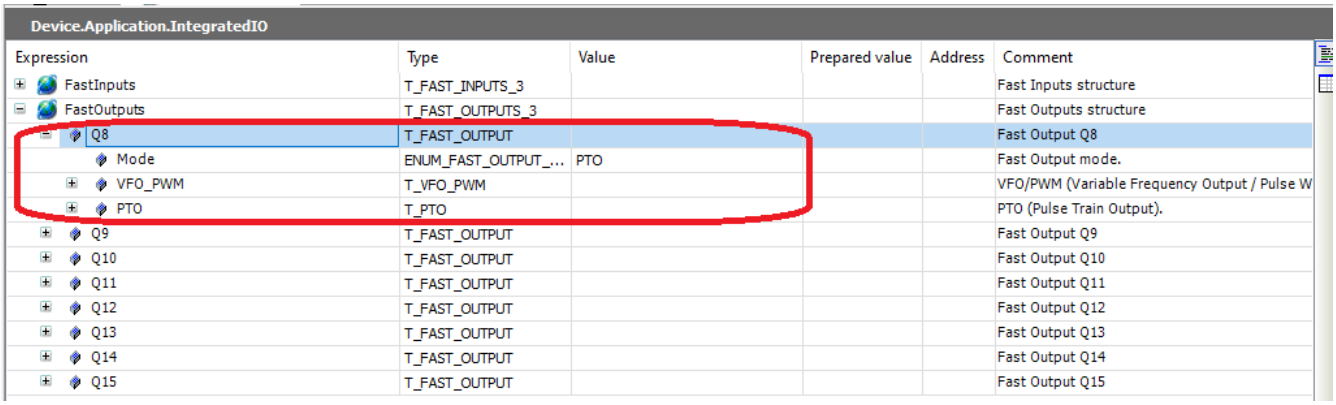



Figure 63: Fast Output structure

The table below describes the fast output variables structure:

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Mode** | Fast output configured mode (read only) | ENUM_FAST_OUTPUT_MODE | DIGITAL_OUTPUT<br>PWM<br>PTO |
| **VFO_PWM** | VFO/PWM structure. It contains a structure to control the fast output when it's configured as VFO/PWM. | T_VFO_PWM | - |
| **PTO** | PTO structure. It contains a structure to control the fast output when it's configured as PTO. | T_PTO | - |

Table 55: Fast Output structure variables

The next subsections give more details about how to use these pulse generator functions, describing these structures for each mode.

#### 5.6.3.1. VFO/PWM

The VFO/PWM (Variable Frequency Output / Pulse Width Modulator) is a pulse generator output mode where the frequency and duty cycle can be controlled by the user program. It's appliable, for example, to control the power transfered to an electric load or to control the angle of a servo motor. The principle of operation of VFO/PWM output is very simple, see the pulsed waveform that is shown in the figure below:

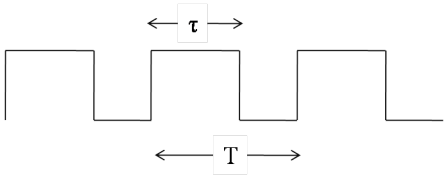



Figure 64: VFO/PWM waveform

The figure shows a pulsed waveform, where T is the period of the pulses and $\tau$ is the pulse width. Those are the pulse parameters which can be changed on VFO/PWM mode. The frequency is defined as the inverse of period, then:

$$f = \frac{1}{T}$$

The duty cycle is the reason between the pulse width and the period, then:

$$D = \frac{\tau}{T}100\%$$

To control the VFO/PWM output, the user program must access the VFO_PWM variable of the fast output structure. The structure of VFO_PWM is shown on the table below:

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Frequency** | Frequency in Hertz | UDINT | 3 to 250000 |
| **DutyCycle** | Duty Cycle in percent | USINT | 0 to 100 |
| **Command** | VFO/PWM commands structure | T_VFO_PWM_COMMAND | - |
| **Status** | VFO/PWM status structure | T_VFO_PWM_STATUS | - |

Table 56: VFO_PWM variable structure

The table below shows the VFO_PWM commands structure.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Enable** | Enable VFO/PWM output | BIT | FALSE or TRUE |

Table 57: VFO/PWM Command structure

The table below shows the VFO_PWM status structure.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **InvalidFrequency** | Frequency value is invalid (out of range) | BIT | FALSE or TRUE |
| **InvalidDutyCycle** | Duty Cycle value is invalid (out of range) | BIT | FALSE or TRUE |

Table 58: VFO/PWM Status structure

Once the Enable command is TRUE, the input parameters will be continuously checked and the status variables will be updated accordingly.

Additionally to the *IntegratedIo* global variables, there is a function block from *LibIntegratedIo* library which allows to instantiate VFO/PWM in POUs written in graphical languages (e.g Ladder Logic Diagram). This function block is, actually, a wrapper to the structured variables described before. The figure below shows the function block instantiated in a Ladder program.

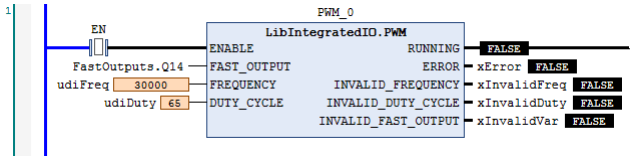



Figure 65: LibIntegratedIo.PWM function block

The table below describes the inputs and outputs variables of the function block.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **ENABLE** | Enable the function block execution. | BOOL | FALSE or TRUE |
| **FAST_OUTPUT** | Fast Output Variable. | REFERENCE TO T_FAST _OUTPUT | FastOutputs.Q8<br>FastOutputs.Q9<br>FastOutputs.Q10<br>FastOutputs.Q11<br>FastOutputs.Q12<br>FastOutputs.Q13<br>FastOutputs.Q14<br>FastOutputs.Q15 |
| **FREQUENCY** | Frequency in Hertz. | UDINT | 3 to 250000 |
| **DUTY_CYCLE** | Duty Cycle in percent. | USINT | 0 to 100 |
| **RUNNING** | VFO/PWM is being performed. | BOOL | FALSE or TRUE |
| **ERROR** | Error occurred in function block execution. The follow variables provide detailed information. | BOOL | FALSE or TRUE |
| **INVALID_FREQUENCY** | Frequency value is invalid (out of range). | BOOL | FALSE or TRUE |
| **INVALID_DUTY_CYCLE** | Duty Cycle value is invalid (out of range). | BOOL | FALSE or TRUE |
| **INVALID_FAST_OUTPUT** | FAST_OUTPUT was not assigned to the block or isn't configured as VFO/PWM. | BOOL | FALSE or TRUE |

Table 59: LibIntegratedIo.PWM function block description

#### 5.6.3.2. PTO

The PTO (Pulse Train Output) is a pulse generator mode. It's used, for example, to control step motors responsible for positioning of mechanisms with considerable inertia. For these cases, the rotation speed must increase slowly (acceleration) when the movement is starting and decrease slowly (deceleration) when the movement is stopping. These acceleration and deceleration are made on pulse train by increasing and decreasing the frequency of the pulses, maintaining the 50% of duty cycle.

There are a set of parameter that must be defined for a pulse train: Start frequency, operation frequency, stop frequency, acceleration profile, total number of pulses, number of pulses in acceleration step, number of pulses in deceleration step. The figure below shows, on Cartesian plane, the relation between the frequency of the pulses and time. The pulse train shown is called trapezoidal profile, because the acceleration and deceleration ramps produce a trapezium shape.

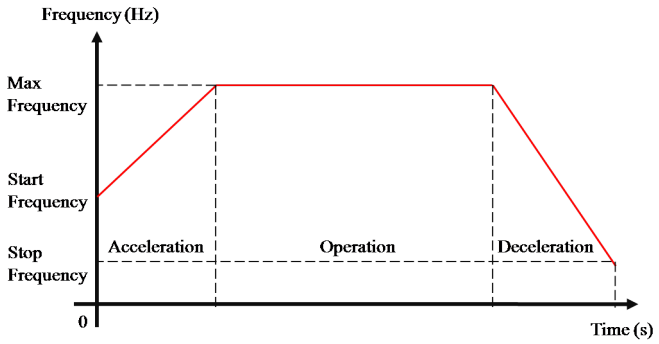



Figure 66: PTO with trapezoidal profile

For some applications it is more recommended to use the “S” profile, which acceleration and deceleration curves are similar to “S” shape. The figure below shows this profile.

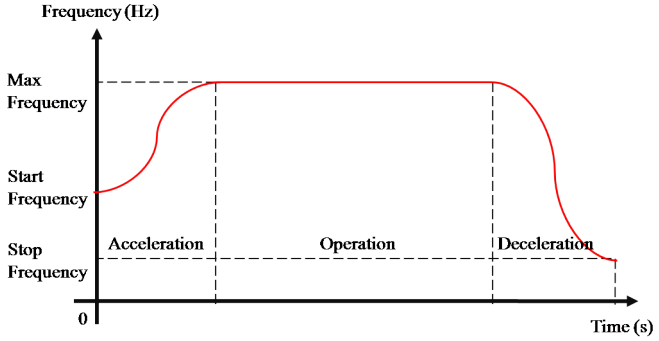



Figure 67: PTO with “S” profile

Besides the PTO parameters, there are status information and commands that the user program can use to control the output. Some important status information are the pulse counter (proportional to a position), the pulse train step (acceleration, operation, deceleration) and, even, if the output is working fine. The commands required to control PTO are to start the pulse train, to stop the pulse train and to stop the pulse train softly (soft stop). The soft stop command is very important, once can be used for emergency situations where the system can’t stop abruptly. The figures below shows how the soft stop command change the pulse train when it is performed. The dashed blue lines represents the PTO if the soft stop command is performed on acceleration and operation steps. The soft stop command on deceleration step has no effect, once the system is already stopping.

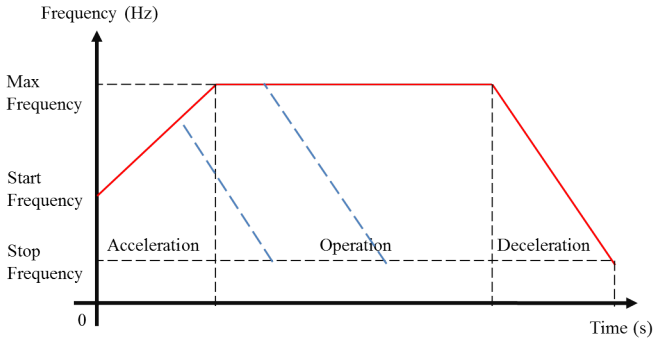



Figure 68: PTO Softstop on trapezoidal profile

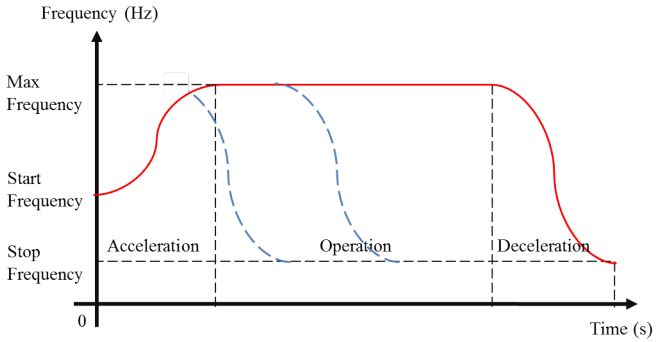



Figure 69: PTO Softstop on "S" profile

To control the PTO, the user program must access the PTO variable of the fast output structure. The structure of PTO is shown on the table below:

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **StartFrequency** | Start frequency in Hertz | UDINT | 3 to 250000 |
| **StopFrequency** | Stop frequency in Hertz | UDINT | 3 to 250000 |
| **MaxFrequency** | Maximum frequency in Hertz | UDINT | 3 to 250000 |
| **AccelerationProfile** | Acceleration profile (FALSE = Trapezoidal profile, TRUE = S profile) | BOOL | FALSE or TRUE |
| **AccelerationPulses** | Pulses in acceleration | UDINT | 0 to (TotalPulses-DecelerationPulses-1) |
| **DecelerationPulses** | Pulses in deceleration | UDINT | 0 to (TotalPulses-AccelerationPulses-1) |
| **TotalPulses** | Total number of pulses | UDINT | 1 to 16777215 |
| **PulsesCounter** | Number of pulses generated for the current pulse train | UDINT | 0 to 4294967295 |
| **Command** | PTO commands structure | T_PTO_COMMAND | - |
| **Status** | PTO status structure | T_PTO_STATUS | - |

Table 60: PTO variable structure

The table below shows the PTO commands structure.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Start** | Start the pulse train when this bit is set (rising edge) | BIT | FALSE or TRUE |
| **Stop** | Stop the pulse train when this bit is set (rising edge) | BIT | FALSE or TRUE |
| **Softstop** | Stop softly the pulse train when this bit is set (rising edge) | BIT | FALSE or TRUE |

Table 61: PTO Command structure

The table below shows the PTO status structure.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **Running** | Pulse train is being performed | BIT | FALSE or TRUE |
| **Acceleration** | Acceleration step (from StartFrequency to MaxFrequency) | BIT | FALSE or TRUE |
| **Deceleration** | Deceleration step (from MaxFrequency to StopFrequency) | BIT | FALSE or TRUE |
| **Operation** | Operation Step (MaxFrequency) | BIT | FALSE or TRUE |
| **Done** | Pulse train has already been performed | BIT | FALSE or TRUE |
| **InvalidFrequency** | Frequency (start, stop or maximum) is invalid | BIT | FALSE or TRUE |
| **InvalidPulses** | Number of pulses (TotalPulses, Acceleration or Deceleration) is invalid | BIT | FALSE or TRUE |

Table 62: PTO Status structure

Once the Start command is TRUE, the input parameters will be continuously checked and the status variables will be updated accordingly.

Additionally to the IntegratedIo global variables, there is a function block from *LibIntegratedIo* library which allows to instantiate PTO in POUs written in graphical languages (e.g Ladder Logic Diagram). This function block is, actually, a wrapper to the structured variables described before. The figure below shows the function block instantiated in a Ladder program.

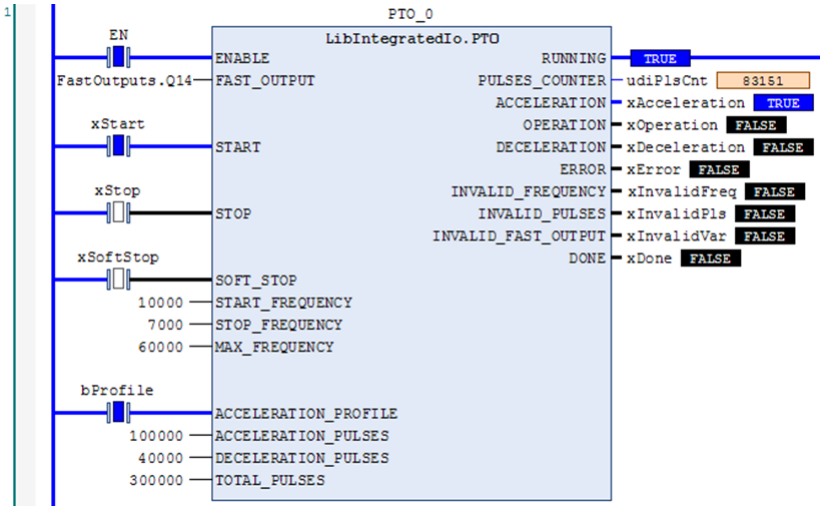



Figure 70: LibIntegratedIo.PTO function block

The table below describes the inputs and outputs variables of the function block.

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **ENABLE** | Enable the function block execution | BOOL | FALSE or TRUE |

| Variable | Description | Type | Allowed Values |
|---|---|---|---|
| **FAST_OUTPUT** | Fast Output Variable | REFERENCE TO T_FAST _OUTPUT | FastOutputs.Q8 FastOutputs.Q9 FastOutputs.Q14 |
| **START** | Start the pulse train when this bit is set (rising edge) | BOOL | FALSE or TRUE |
| **STOP** | Stop the pulse train when this bit is set (rising edge) | BOOL | FALSE or TRUE |
| **SOFT_STOP** | Stop softly the pulse train when this bit is set (rising edge) | BOOL | FALSE or TRUE |
| **START_FREQUENCY** | Start frequency in Hertz | UDINT | 3 to 250000 |
| **STOP_FREQUENCY** | Stop frequency in Hertz | UDINT | 3 to 250000 |
| **MAX_FREQUENCY** | Maximum frequency in Hertz | UDINT | 3 to 250000 |
| **ACCELERATION_PROFILE** | Acceleration profile (FALSE = Trapezoidal profile, TRUE = S profile) | BOOL | FALSE or TRUE |
| **ACCELERATION_PULSES** | Pulses in acceleration | UDINT | 0 to (TotalPulses-DecelerationPulses-1) |
| **DECELERATION_PULSES** | Pulses in deceleration | UDINT | 0 to (TotalPulses-AccelerationPulses-1) |
| **TOTAL_PULSES** | Total number of pulses | UDINT | 1 to 16777215 |
| **RUNNING** | Pulse train is being performed | BOOL | FALSE or TRUE |
| **PULSES_COUNTER** | Number of pulses generated for the current pulse train | UDINT | 0 to 4294967295 |
| **ACCELERATION** | Acceleration step (from StartFrequency to MaxFrequency) | BOOL | FALSE or TRUE |
| **OPERATION** | Operation Step (MaxFrequency) | BOOL | FALSE or TRUE |
| **DECELERATION** | DecelerationStep (from MaxFrequency to StopFrequency) | BOOL | FALSE or TRUE |
| **ERROR** | Error occurred in function block execution. The follow variables detail the error. | BOOL | FALSE or TRUE |
| **INVALID_FREQUENCY** | Frequency (start, stop or maximum) is invalid | BOOL | FALSE or TRUE |
| **INVALID_PULSES** | Number of pulses (acceleration or deceleration) is invalid | BOOL | FALSE or TRUE |
| **INVALID_FAST_OUTPUT** | FAST_OUTPUT was not assigned to the block or isn't configured as PTO. | BOOL | FALSE or TRUE |
| **DONE** | Pulse train has already been performed | BOOL | FALSE or TRUE |

Table 63: LibIntegratedIo.PTO function block description

### 5.6.4. Analog Inputs

The parameters related to the Analog Inputs are shown below:

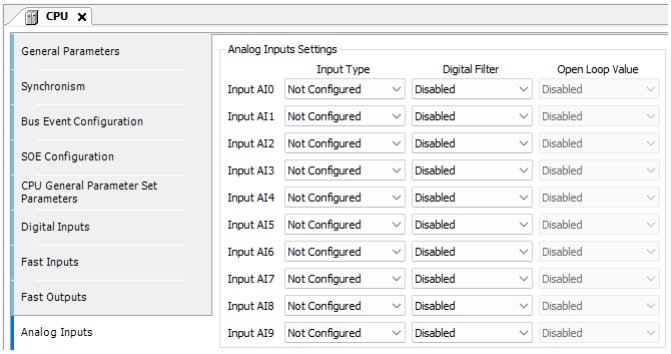



Figure 71: Analog Inputs Parameters

The table below shows the possible configuration values:

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Input Type** | Selects the input type<br>(AI0 to AI5 current only)<br>(AI6 to AI9 voltage only) | Not configured | Not configured<br>Voltage 0 - 10 Vdc (AI6 to AI9)<br>Currrent 0 - 20 mA (AI0 to AI5)<br>Current 4 - 20 mA (AI0 to AI5) |
| **Digital Filter** | Enable/Disable<br>(1st order low pass digital filter for each input) | Disabled | Disabled<br>100 ms<br>1 s<br>10 s |
| **Open Loop Value** | Set value when in open loop condition<br>(Only valid for 4 - 20 mA scale) | Disabled | Disabled<br>0<br>30000 |

Table 64: Analog Inputs Parameters

**Notes:**
**Input Type:** Be sure to use the proper pin on the terminal block correspondent to the selected type (voltage or current).
**Open Loop Value:** Determines the behavior of the input variable when set to 4 - 20 mA scale and current less than 3 mA.

### 5.6.5. Analog Outputs

The parameters related to the Analog Outputs are shown below:

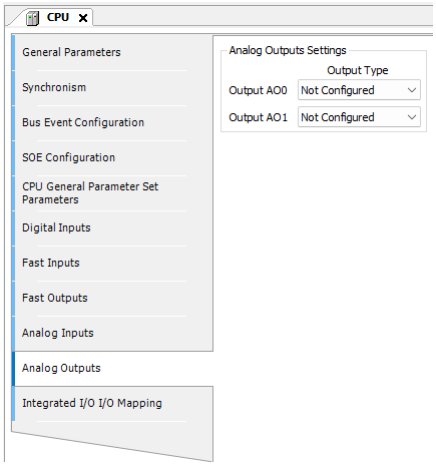



Figure 72: Analog Outputs Parameters

The table below shows the possible configuration values:

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Output Type** | Selects the output type | Not configured | Not configured<br>Voltage 0 - 10 Vdc<br>Current 0 - 20 mA<br>Current 4 - 20 mA |

Table 65: Analog Outputs Parameters

### 5.6.6. I/O Mapping

In the *I/O Mapping* tab, it is possible to configure the name and description for each input and output variable.

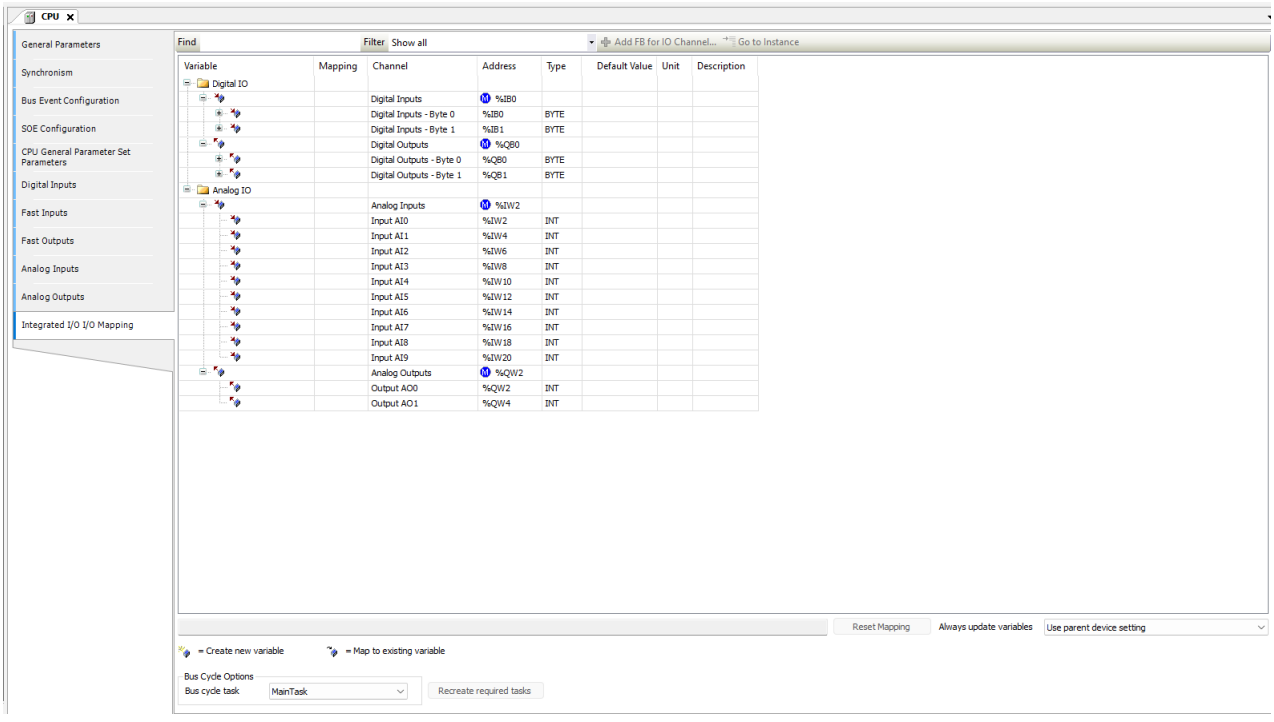



Figure 73: I/O tag mapping

## 5.7. Management Tab Access

Developed to perform configuration and diagnostics access to some features. The *Management* tab of the System Web Page has its access protected by user and password, with *admin* as the default value for both fields.

On the Management tab, there are other resources such as *System*, *Network*, *SNMP*, *USB Device*, *Firewall*, *OpenVPN*, and *FTP Server*. The resources available on this tab vary according to the features available for the controller used and can only be accessed after the user has logged in, as shown in the figure below.

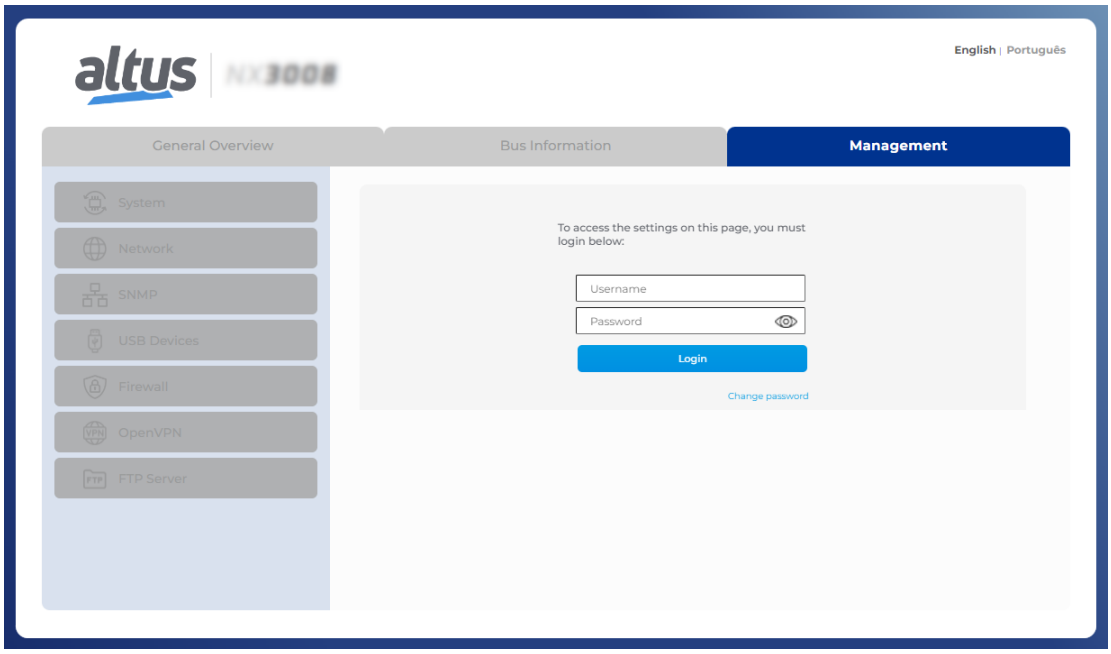



Figure 74: Management Tab Access

### 5.7.1. System Section

In the *System* section, you can perform a CPU firmware update. For cases in which the update is done remotely (through a radio or satellite connection, for example), the minimum speed of this link must be 128 kbps.

#### 5.7.1.1. Clock Setting

On the System Web Page, it is possible to adjust the controller's clock, which is found in the *System* section of the Management tab. The date and time format follows the ISO 8601 standard for date and time sampling (YYYY/MM/DD hh:mm:ss), as shown in the image below:

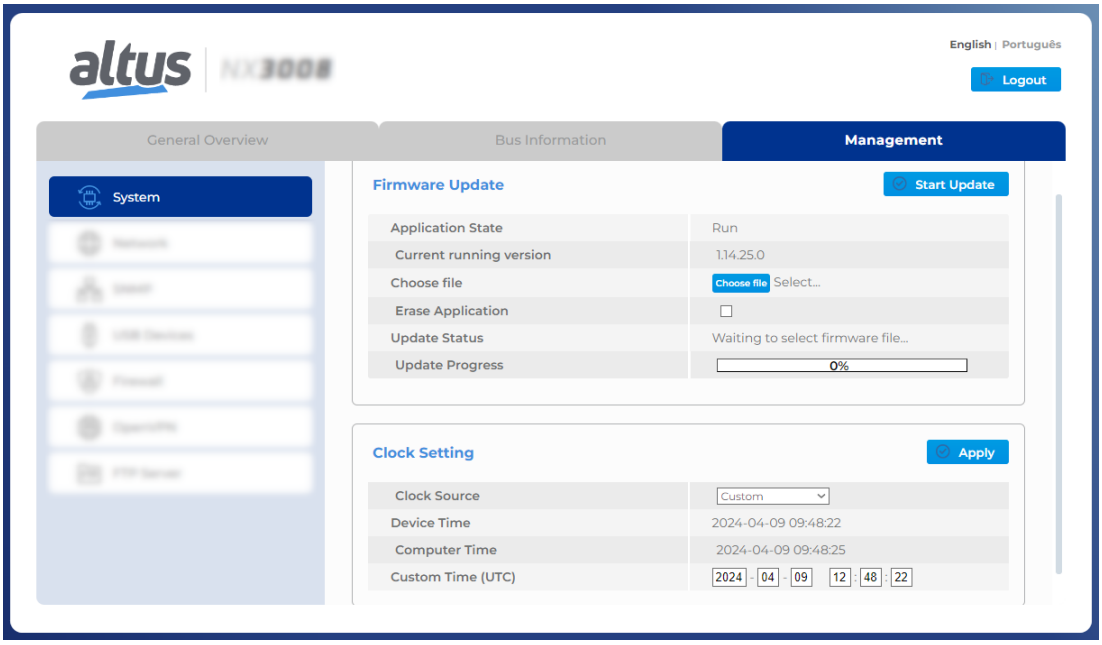



Figure 75: Clock Setting

This feature has two modes for adjusting the device's time, which can be selected in the item "Clock Source", providing the user with two options for synchronizing the clock.

#### *5.7.1.1.1. Computer Time (UTC)*

In computer time mode, user can apply the time configured on his computer in UTC for his device. To do so, select the option "Computer" in the "Clock Source" item. After clicking on the "Apply" button, it is necessary to validate the device's credentials, then the CPU will receive the date and UTC time that are configured on the computer.

#### *5.7.1.1.2. Custom Time (UTC)*

In the custom time mode, the user can prepare a custom time in UTC standard to be applied to the device's internal date and time. To do so, select the "Custom" option in the "Clock Source" item. With the mode selected, the user must configure the desired date and time in the "Custom Time (UTC)" item, which will be initialized with the browser's local time. So, after the user clicks on the "Apply" button and validates the device's credentials, it will have its internal time configured with the time configured in the item "Custom Time (UTC)". For configuration limits, refer to section RTC Operating Limits.

### 5.7.2. Network Section

Designed to assist in the usability of the controller, the *Network* section (figure below) allows you to change network addresses and run the Network Sniffer.

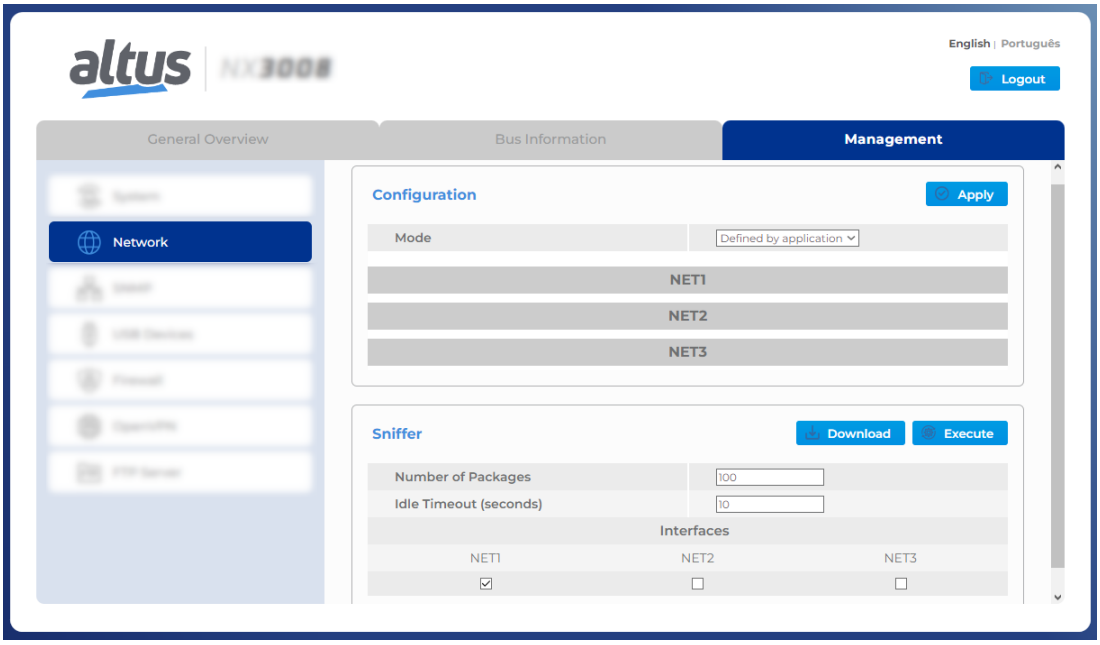



Figure 76: Network Section

#### 5.7.2.1. Network Section Configurations

##### *5.7.2.1.1. Defined by Application*

The Mode field defines which configuration the controller should load for its interfaces. This field can be configured as *Defined By Web Page* or *Defined By Application*.

When set to *Defined by Application*, the interface table is disabled, not allowing changes, as shown in the figure below. In this mode, the settings applied to the controller are those defined by the application.

**ATTENTION**

The table for network configuration is displayed only when there is no application on the controller or the controller is not running. It is not possible to change the network settings while an application is running on the controller.

Below is an image with *Defined by Application* mode selected, showing the interface table disabled.

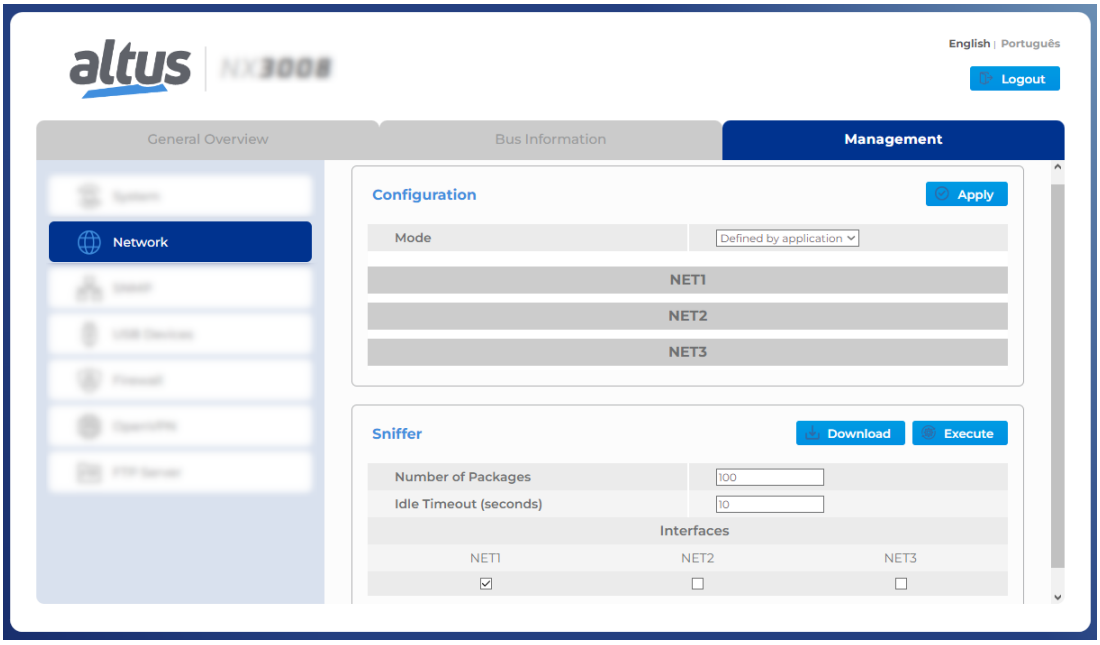



Figure 77: Interfaces Table - Application Mode

#### 5.7.2.1.2. *Defined by web page*

For *Defined by Web Page* mode, the interface table remains enabled as shown in the figure below.

In this mode, the user can set the IP Address, Network Mask, and Gateway for each of the available Ethernet interfaces, as well as enabling and disabling NETs 2 and 3.

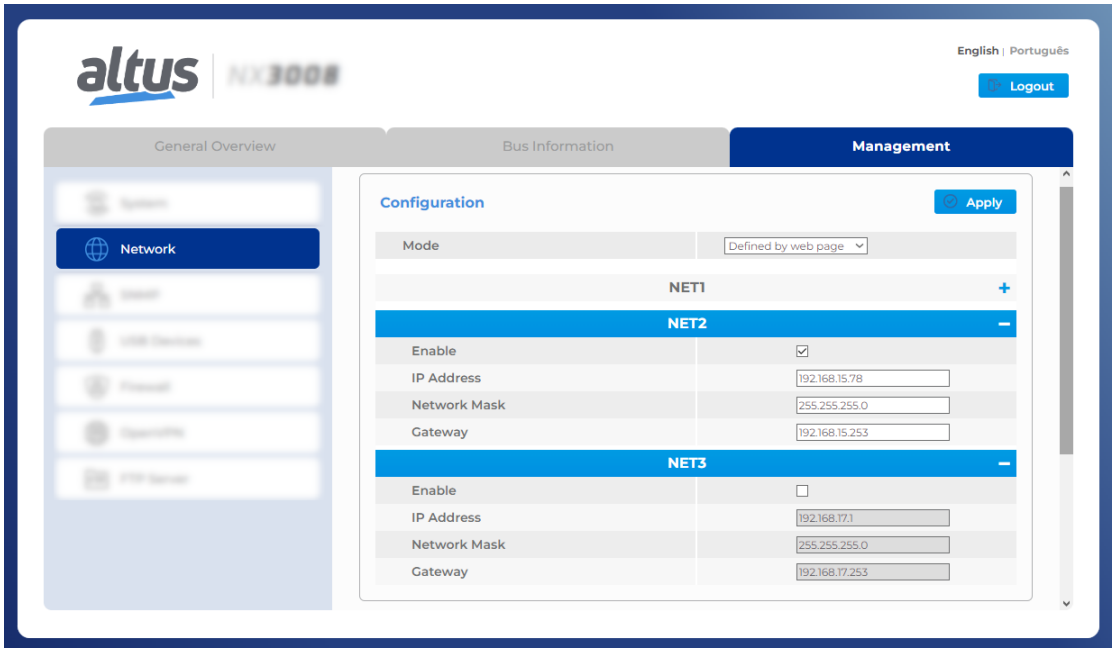



Figure 78: Interfaces Table - Web Mode

The *Enable* checkbox allows the user to enable and disable the Ethernet interfaces. This is only available to be checked or unchecked when the configuration is *Defined by web page*.

When this checkbox is unchecked, it indicates that the NET interface is disabled, i.e., it will not receive configuration and will be deactivated, as the NET 3 of the figure above. When the interface is enabled (with the checkbox checked), according to NET 2 in the figure above, the settings are available for editing.

To have the settings applied to the controller, simply click the *Apply* button. This process checks if there were any errors in the configuration made and, if so, displays a message on the browser screen indicating the error. If the settings are correct, after clicking *Apply*, a confirmation window appears in the browser to apply the new settings. By clicking *OK*, the settings are sent to the controller and applied.

**ATTENTION**

When making network changes in the controller, the interfaces will be restarted, which may cause a communication loss. Especially when changing the IP address value.

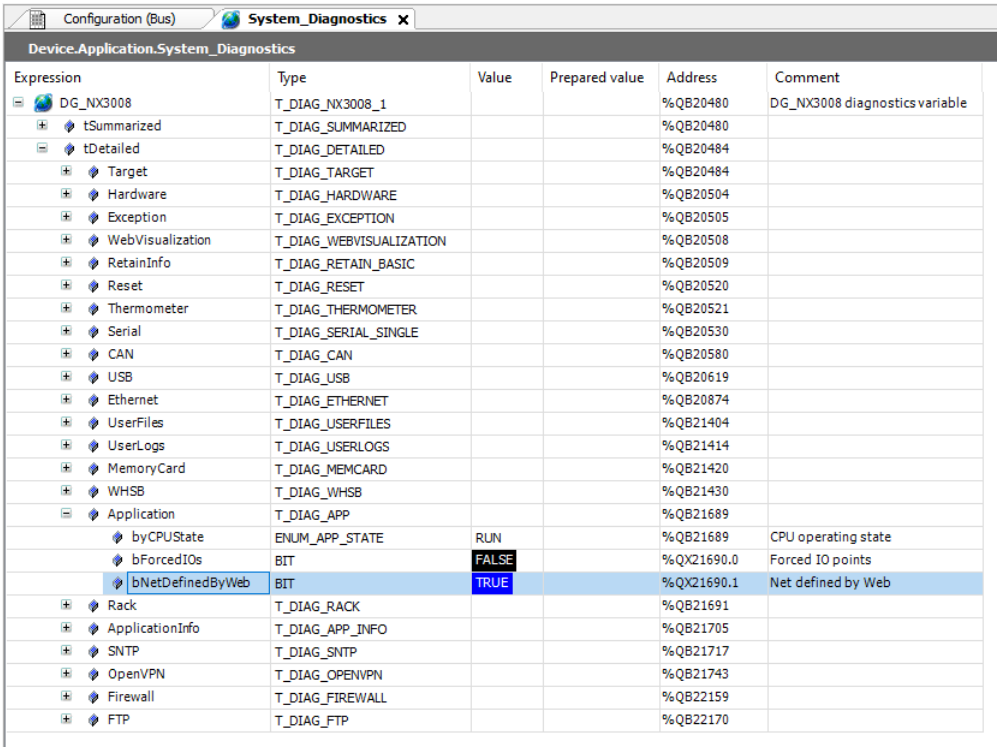
When applying settings using the *Defined by Application* mode, the controller will assume the configuration that was defined by the loaded application. If there is no application, the current configuration will be maintained, with only the configuration mode being changed.

Using the *Defined by Web Page* mode, the addresses indicated on the web page will be loaded.

**ATTENTION**

The *Defined by Web Page* mode configures the interfaces to operate in Simple Mode.

It is possible to monitor through Mastertool whether the IP address is configured from the Web Page or from the application by the *bNetDefinedByWeb* diagnostic BIT in the *Application* group, which will change to *TRUE* if the IP is configured from the Web Page and to *FALSE* if it is configured from the application.




| Expression | Type | Value | Prepared value | Address | Comment |
|---|---|---|---|---|---|
| DG_NX3008 | T_DIAG_NX3008_1 | | | %QB20480 | DG_NX3008 diagnostics variable |
| tSummarized | T_DIAG_SUMMARIZED | | | %QB20480 | |
| tDetailed | T_DIAG_DETAILED | | | %QB20484 | |
| Target | T_DIAG_TARGET | | | %QB20484 | |
| Hardware | T_DIAG_HARDWARE | | | %QB20504 | |
| Exception | T_DIAG_EXCEPTION | | | %QB20505 | |
| WebVisualization | T_DIAG_WEBVISUALIZATION | | | %QB20508 | |
| RetainInfo | T_DIAG_RETAIN_BASIC | | | %QB20509 | |
| Reset | T_DIAG_RESET | | | %QB20520 | |
| Thermometer | T_DIAG_THERMOMETER | | | %QB20521 | |
| Serial | T_DIAG_SERIAL_SINGLE | | | %QB20530 | |
| CAN | T_DIAG_CAN | | | %QB20580 | |
| USB | T_DIAG_USB | | | %QB20619 | |
| Ethernet | T_DIAG_ETHERNET | | | %QB20874 | |
| UserFiles | T_DIAG_USERFILES | | | %QB21404 | |
| UserLogs | T_DIAG_USERLOGS | | | %QB21414 | |
| MemoryCard | T_DIAG_MEMCARD | | | %QB21420 | |
| WHSB | T_DIAG_WHSB | | | %QB21430 | |
| Application | T_DIAG_APP | | | %QB21689 | |
| byCPUState | ENUM_APP_STATE | RUN | | %QB21689 | CPU operating state |
| bForcedIOs | BIT | FALSE | | %QX21690.0 | Forced IO points |
| bNetDefinedByWeb | BIT | TRUE | | %QX21690.1 | Net defined by Web |
| Rack | T_DIAG_RACK | | | %QB21691 | |
| ApplicationInfo | T_DIAG_APP_INFO | | | %QB21705 | |
| SNTP | T_DIAG_SNTP | | | %QB21717 | |
| OpenVPN | T_DIAG_OPENVPN | | | %QB21743 | |
| Firewall | T_DIAG_FIREWALL | | | %QB22159 | |
| FTP | T_DIAG_FTP | | | %QB22170 | |

Figure 79: Diagnostics - IP defined by the Web Page

#### 5.7.2.2. Network Sniffer

The network sniffer, shown in the figure below, can be used to observe traffic on physical interfaces, except for USB devices such as modems and wifi adapters. It has two basic settings:

**Number of Packets:** This is the number of packets you want to capture. The configured value of this parameter must be within the range of 100 to 25000 packets;

**Idle Timeout (seconds):** If there is no packet traffic on the interface after this configured timeout, Sniffer is terminated. It can be configured with values between 1 and 3600 seconds.

Only a few moments after the screen opens will the *Run* button, which starts Sniffer's execution, become available. The *Download* button will only be unlocked if there is a Sniffer related file available for download. If the Sniffer has never been run or the file is deleted, the button will not be available.

When running the Network Sniffer, the page will disable the edit fields, the *Download* button will be locked, and the *Run* button will become the *Stop* button, as shown in the figure below.

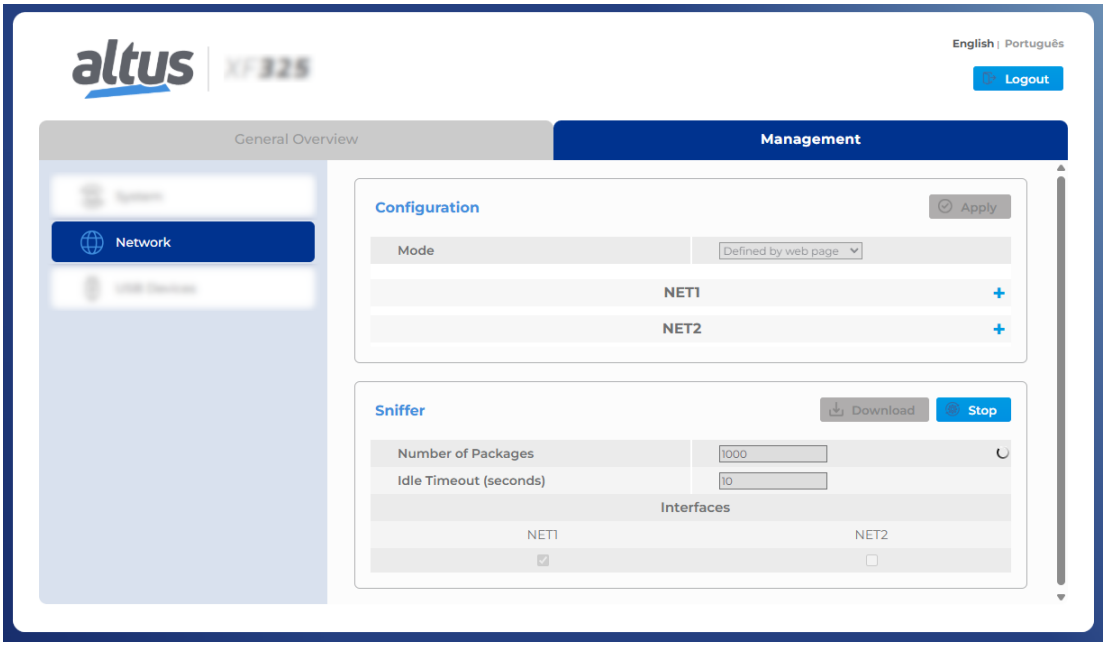



Figure 80: Network Sniffer Running

The *Stop* button can be used to end the sniffer execution at any time after it has been started.

For each interface on which Sniffer runs, it generates a **.pcap** file. These files are named according to the controller model and the analyzed interface, following the format `[PLC_MODEL]_[INTERFACE].pcap`. They are stored inside a **.zip** file, also named according to the controller model, following the format `[PLC_MODEL]_capture.zip`.

At the end of the sniffer execution, a message is displayed asking whether or not to automatically download the generated files. These files are stored in the *InternalMemory* folder of the User Files Memory and can be accessed through the controller's programming software. The downloaded file is always in the **.zip** extension, which groups the other files.

If any problems occur related to insufficient memory due to the generation of sniffer files, it will be indicated to the user. It is recommended to try running the analyzer again with a smaller *Number of Packets* configuration.

The network sniffer can terminate its execution for three reasons: insufficient memory, idle time limit of interfaces exceeded, and manual cancellation.

## 5.8. USB Port

The USB Host port present on Nexto Series controllers allows to extend the controller's functionalities by using several types of USB dongles.

The management of USB devices is done through a dedicated section located on the *Management* tab of the controller's System Web Page as shown below:

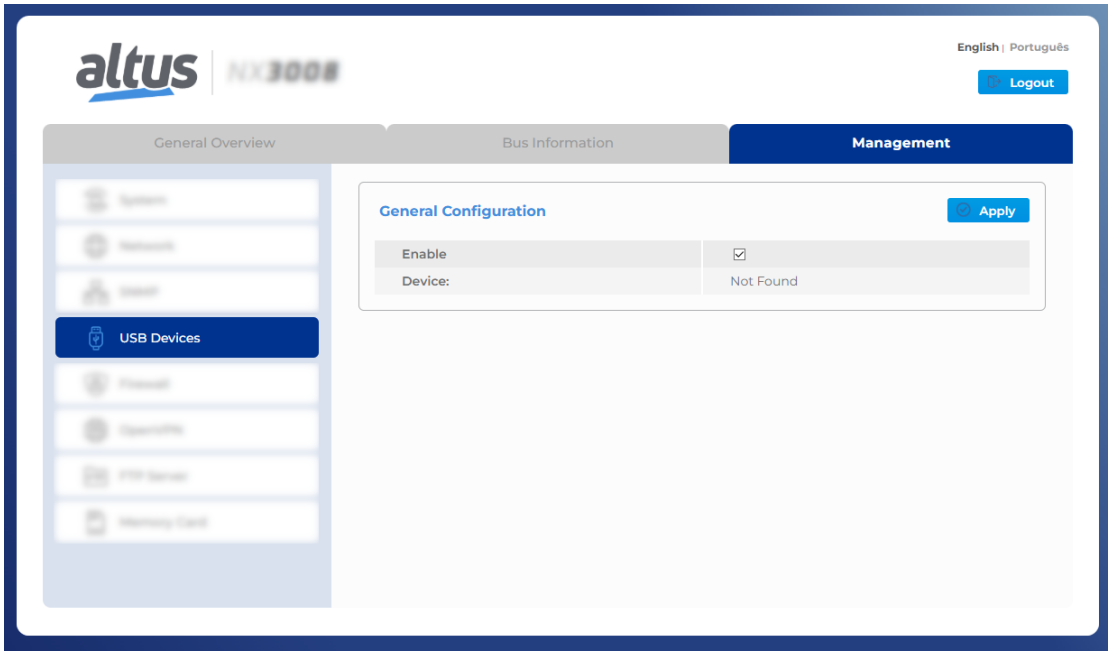



Figure 81: USB Devices Section

On this page, the USB interface's state can be configured by marking or unmarking the checkbox and applying the configuration with the button Apply. If the USB interface is enabled, the content of this page changes dynamically according to the type of USB device that is connected. In the example above, the USB interface is enabled and there is no device connected. When USB is disabled, the content is the presented in the figure, always showing Device as "Not Found", but with the checkbox unmarked.

The following sections describe all the types of USB devices currently supported. If an unsupported device is connected, the page will inform that device is unknown:

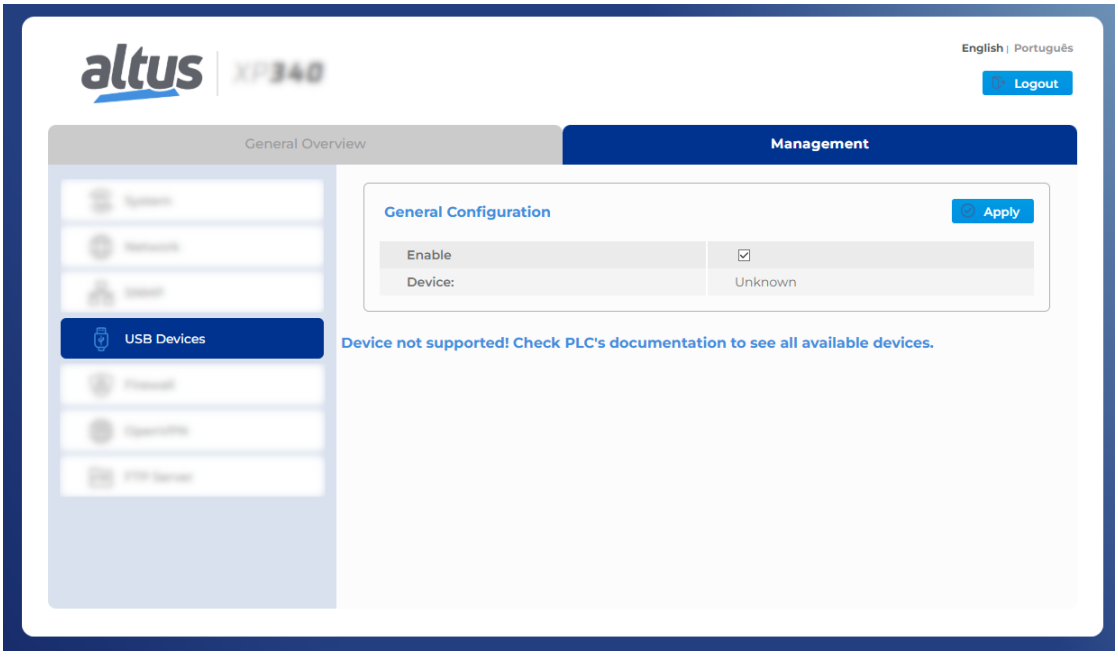



Figure 82: USB Devices - Unknown

### 5.8.1. Mass Storage Device

#### 5.8.1.1. General Storage

Mass storage devices can be used to expand the controller's flash memory to store big amount of data, like on datalogger applications, for instance. To use a USB mass storage device, simply connect it to the USB port. After a few seconds, when the device is properly detected and mounted, the USB LED will turn on and the device information will appear on section *USB Devices* located at the *Management* tab of the controller's System Web Page as shown below:

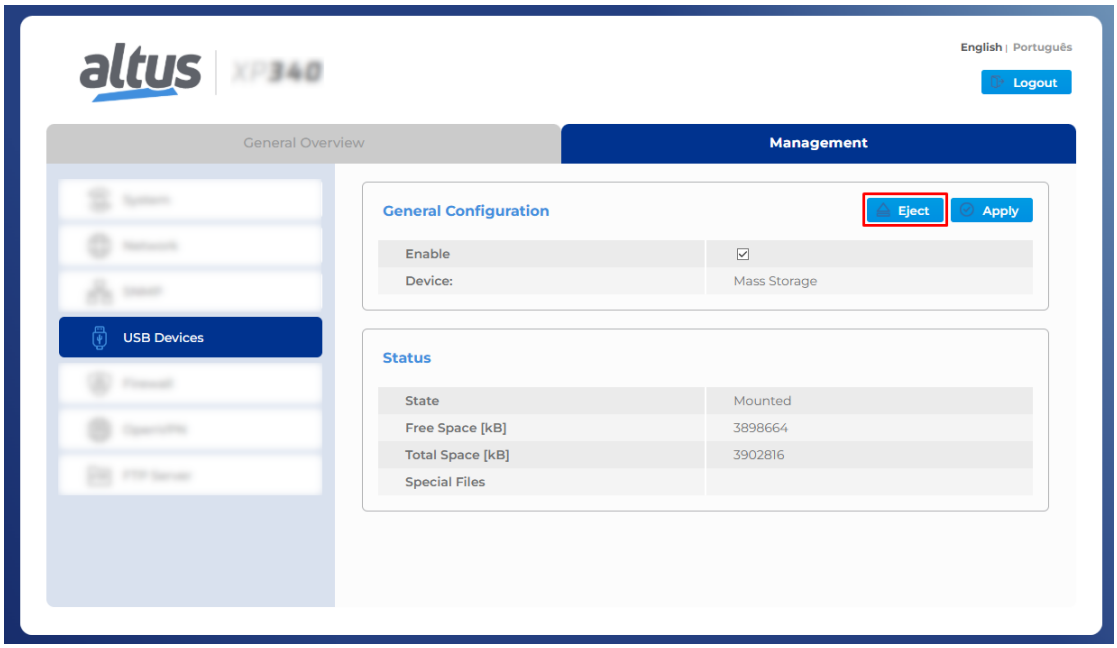



Figure 83: Mass Storage Device Information

The information shown on the status section of this page is also available in the symbolic variables diagnostics structure (see Section Diagnostics via Variables).

**ATTENTION**

The USB mass storage device must be formatted as a FAT32 volume. Other filesystem formats are not supported.

The device can be ejected using the command provided on the *Commands* area of this page as indicated on the picture above.

After the device is properly detected and mounted, a new folder called *Mass_Storage* will appear on the controller's memory as shown on the picture below:

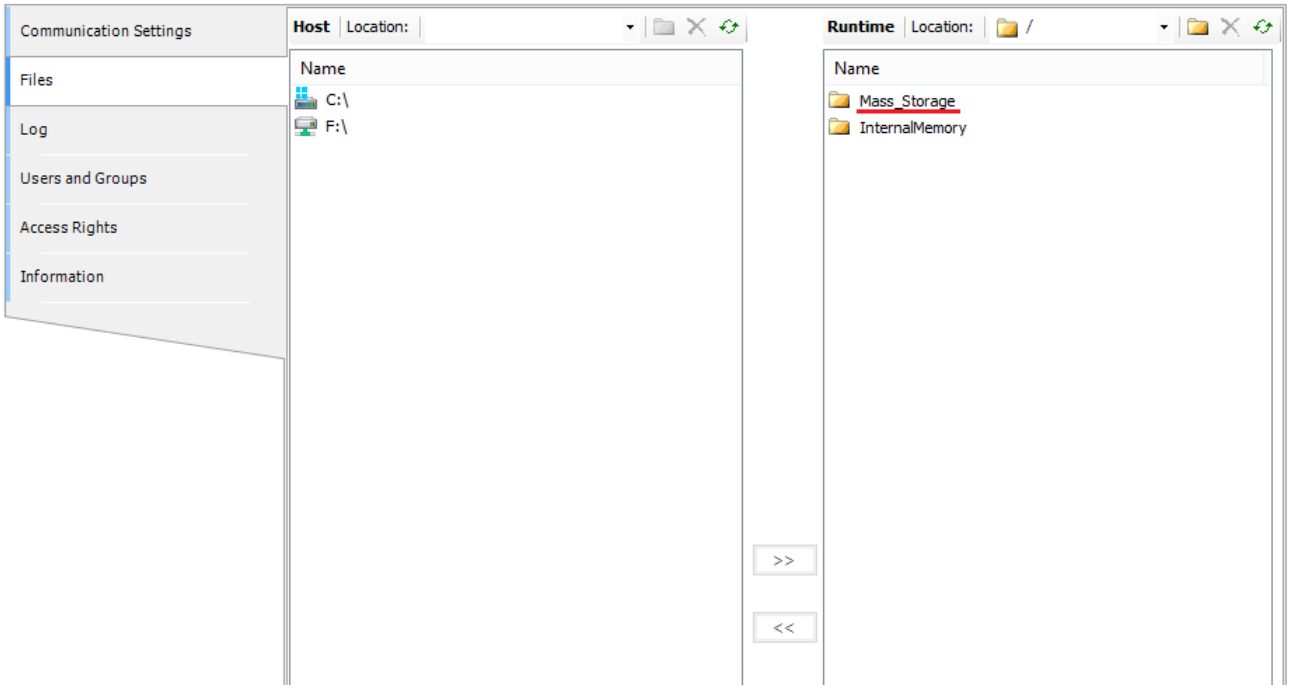



Figure 84: USB Mass Storage Folder

#### 5.8.1.2. Application Not Loading on Startup

The USB mass storage device can be used to prevent the controller from automatically loading the application after the power on. To do that, simply place an empty text file called *"dontbootapp.txt"* on the root folder of mass storage device. The presence of this file is informed in the *Special Files* field on controller's System Web Page as shown below.

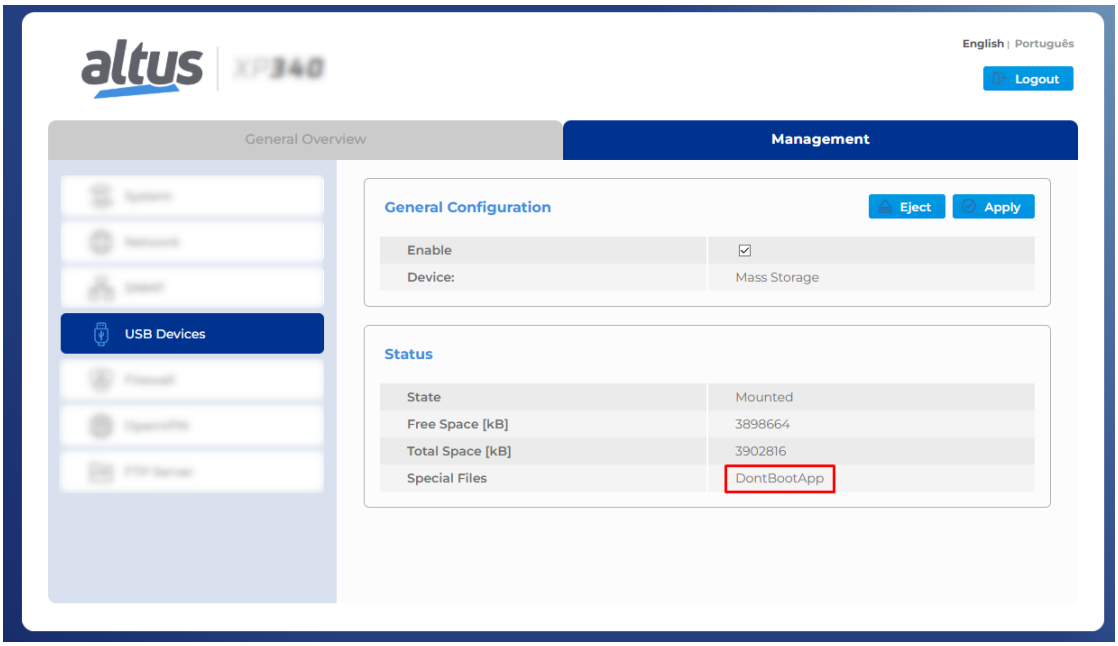



Figure 85: DontBootApp in the Mass Storage Device

#### 5.8.1.3. Transferring an Application from a USB Device

The USB mass storage device can also be used to transfer an application to the controller. To do this, place the two files *Application.app* and *Application.crc* in the root folder of the mass storage device. If there is a WebVisu declared in the project, the folder *PLC Folder* must also be copied (these files are created using Mastertool, executing the command *Online -> Create boot application* when offline). After the power on, if the controller detects the presence of these files on the USB mass storage device, the following sequence of actions will occur:

- The controller will start copy of the application from USB device to internal memory
- After finishing the copy process, the USB device will be ejected (USB LED will turn off)
- The new application will start (RUN) automatically (if *"dontbootapp.txt"* is not present)

The presence of the application is informed in the *Special Files* field on controller's System Web Page as shown below:

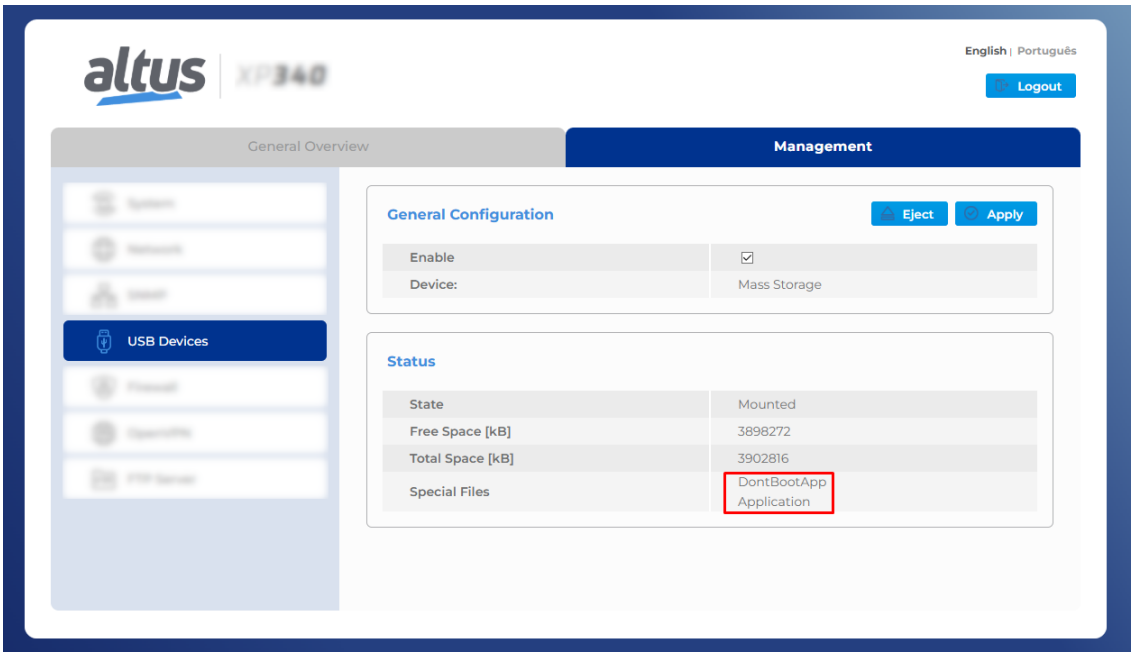



Figure 86: Application in the Mass Storage Device

Note that it is possible to have multiple *Special Files* in the same Mass Storage. In the example above, the PLC will transfer the new application to the internal memory but not load it on startup (hence, will not go to RUN).

### 5.8.2. USB to RS-232 Converter

The Nexto Series allows to implement a RS-232 port using a USB to Serial converter. These converters are based on an internal controller chip. The following table shows the list of supported controllers:

| Controller | Manufacturer |
|---|---|
| FT232 | FTDI |
| PL2303 | Prolific |

Table 66: Supported USB to RS-232 converters

This port is intended to be used exclusively with the Serial communication function blocks provided by the *NextoSerial* library, allowing to implement a point-to-point communication with equipments that use simple protocols (non time critical) like *Radio modems*, *Barcode readers*, *RFID readers*, etc... Additionally, this kind of solution has the following limitations:

- **Baud Rate:** values lower than 4800 bps are not supported
- **Data Bits:** value "5" is not supported (only 6, 7 or 8)

- **Parity:** values “mark” and “space” are not supported (only Odd, Even and None)
- **Stop Bits:** value “1.5” is not supported (only 1 or 2)

After plugging the converter into the USB port, and the device is detected and mounted correctly, the device information will appear in the *USB Devices* section, located in the *Management* tab of the controller’s System Web Page as shown below:

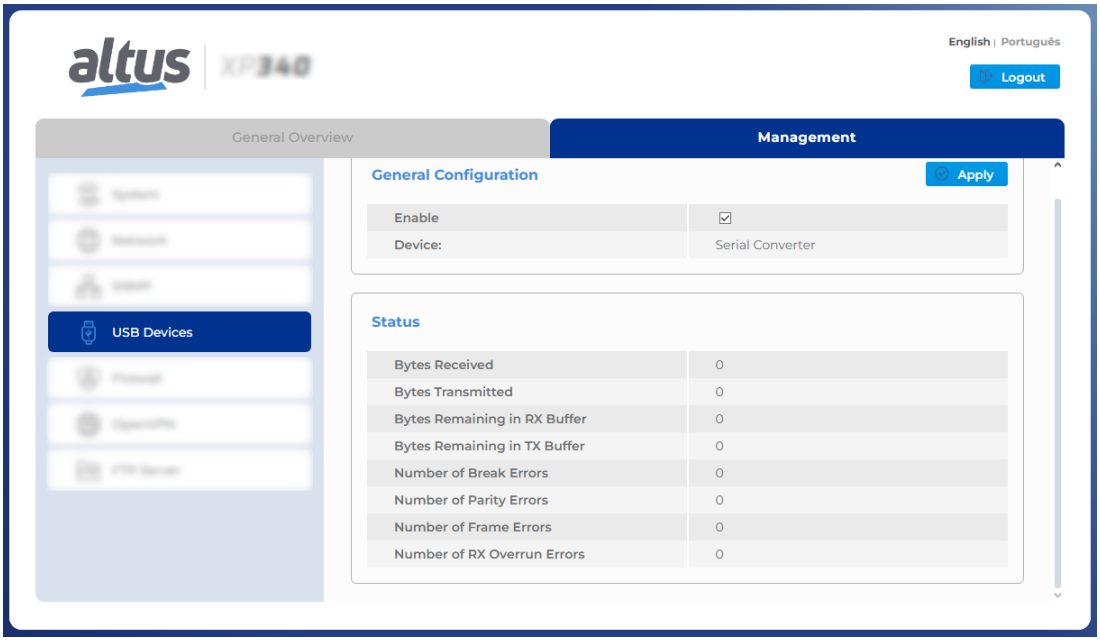



Figure 87: USB Devices - Serial Converter

The information shown on this page is also available in the symbolic variables diagnostics structure (see Section Diagnostics via Variables).

This additional serial port will be identified internally as *COM10*, and will not have a representation on the project treeview. From this point, this port can be used for communication using the *NextoSerial* functions similar to the native ports. For this kind of port, the handshake configuration is limited to *RS232_MANUAL* only (must be considered when configuring the port with *SERIAL_CFG* function).

### 5.8.3. USB Interface Control User Function

To facilitate the management of the USB interface, a function was developed that can be called directly by the user’s application code. The **SetUSBState** function was implemented within the **NextoStandard** library. The image below shows the library information, as presented in the *Library Manager*.

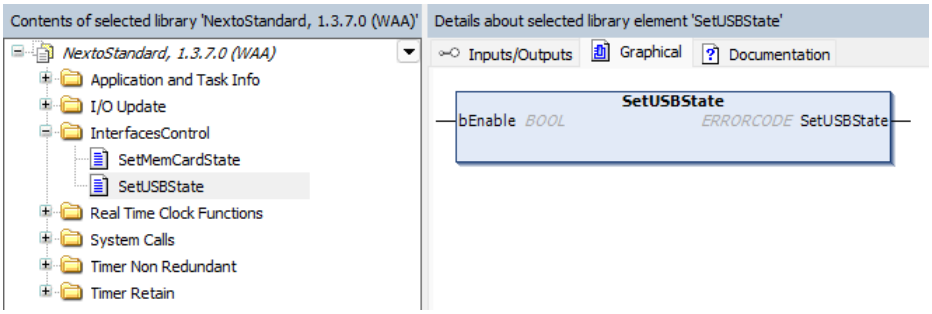



Figure 88: SetUSBState information in Library Manager

The function has an input variable of type *bool*, **bEnable**, which receives the value to enable or disable the USB interface. The function has three return values: *NoError* on success, *SetUSBStateFail* on failure, or *ImportFunctionNotFound* if the function is not supported. The image below shows a basic example of variable declaration and function call.

```
PROGRAM UserPrg
VAR
bSetUSBInterfaceState : BOOL;
stErrorCode           : NextoStandard.ERRORCODE;
END_VAR
-------------------------------------------------------------------------------
stErrorCode := NextoStandard.SetUSBState(bEnable := bSetUSBInterfaceState);
```

**ATTENTION**

The function executes the command to set the desired value for the USB interface. It is neither necessary nor recommended that the function be called cyclically.

## 5.9. Communication Protocols

Independently of the protocols used in each application, the Nexto Series CPUs has some maximum limits for each CPU model. There are basically two different types of communication protocols: symbolic and direct representation mappings. The maximum limit of mappings as well as the maximum protocol quantity (instances) is defined on table below:

| | XF300 | XF300-B | XF315 | XF325 | XF325-W |
|---|---|---|---|---|---|
| **Mapped Points** | 20480 | | | | |
| **Mappings (Per Instance / Total)** | 5120 / 20480 | | | | |
| **Requests** | 512 | | | | |
| **NETs – Client or Server instances (Per NET / Total)** | 4 / 8 | | | | |
| **COM (n) – Master or Slave instances** | 1 | | | | |
| **Control Centers** | - | | | | |

Table 67: Protocol Limits per CPU

**Notes:**

**Mapped Points:** It is the maximum number of mapped points that the CPU supports. Each mapping can contain one or more mapped points, depending on the data size. This varies depending on whether simple variables or ARRAY-type variables are used. Each simple variable, as well as each index of an ARRAY, is counted as a mapped point, even if it occupies more than one address in the driver. For example, a simple DWORD-type variable mapped in the MODBUS protocol will be counted as a single point, even though it occupies two consecutive addresses/registers in the driver.

**Mappings:** A "mapping" is the relationship between an internal application variable and an object of the application protocol. This field informs the maximum number of mappings supported by the CPU. It corresponds to the sum of all mappings made within the instances of communication protocols and their respective devices.

**Requests:** The sum of requests for communication protocols, declared on the devices, cannot exceed the maximum number of requests supported by the CPU.

**NETs – Clients or Servers Instances:** This field defines the maximum number of protocol instances per Ethernet interface, and also the total maximum distributed along all the Ethernet interfaces of the system.

**COM (n) – Master or Slave Instances:** Due to its characteristics, each serial interface supports only one communication protocol instance. Examples of instances compatible with serial interfaces: MODBUS RTU Master and MODBUS RTU Slave.

The limitations of the MODBUS protocol for symbolic mappings can be seen in the table below.

| Limitations | MODBUS RTU Master | MODBUS RTU Slave | MODBUS Ethernet Client | MODBUS Ethernet Server |
|---|---|---|---|---|
| **Devices per instance** | 64 | 1(1) | 64 | 64(2) |
| **Requests per device** | 32 | - | 32 | - |

| Limitations | MODBUS RTU Master | MODBUS RTU Slave | MODBUS Ethernet Client | MODBUS Ethernet Server |
|---|---|---|---|---|
| **Simultaneous requests per instance** | - | - | 128 | 64 |
| **Simultaneous requests per device** | - | - | 8 | 64 |

Table 68: MODBUS Protocol Limitations for Symbolic Mappings

**Notes:**

**Devices per instance:**

- Master or Client Protocol: Number of slave or server devices supported by each Master or Client protocol instance.
- MODBUS RTU Slave Protocol: the limit (1) informed relates to serial interfaces that do not allow a Slave to establish communication through the same serial interface, simultaneously, with more than one Master device. It's not necessary, nor is it possible to declare or configure the Master device in the instance of the MODBUS RTU slave protocol. The master device will have access to all the mappings made directly on the instance of MODBUS RTU slave protocol.
- MODBUS RTU Server Protocol: the limit (2) informed relates to the Ethernet interfaces, which limit the amount of connections that can be established with other devices through a single Ethernet interface. It is not necessary, nor is it possible to declare or configure Clients devices in the instance of the MODBUS Server protocol. All Clients devices will have access to all the mappings made directly in the instance of the MODBUS Server protocol.

**Requests by device:** Number of requests, such as reading or writing holding registers, which can be configured for each of the devices (slaves or servers) from Master or Client protocols instances. This parameter does not apply to instances of Slave or Server protocols.

**Simultaneous Requests per Instance:** Number of requests that can be simultaneously transmitted by each client protocol instance or that can be received simultaneously by each server protocol instance. MODBUS RTU protocol instances, Master or Slave, do not support simultaneous requests.

**Simultaneous Requests per Device:** Number of requests that can be simultaneously transmitted for each MODBUS server device, or may be received simultaneously from each MODBUS client device. MODBUS RTU devices, Master or Slave do not support simultaneous requests.

### 5.9.1. Protocol Behavior x CPU State

The table below shows in detail the behavior of each configurable protocol in Nexto Series CPUs in every state of operation.

| Protocol | Type | CPU operational state | | | | | |
|---|---|---|---|---|---|---|---|
| | | STOP | | | RUN | | |
| | | After download, before application starts | After the application goes to STOP (PAUSE) | After an exception | Non redundant or Active | Redundant in Stand-by | After a breakpoint in MainPrg |
| **MODBUS Symbol** | Slave/Server | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Master/Client | × | × | × | ✓ | ✓ | ✓ |
| **EtherCAT** | Master | ✓ | ✓ | × | ✓ | NA | ✓ |
| **OPC DA** | Server | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| **OPC UA** | Server | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SNTP** | Client | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **HTTP** | Server | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SNMP** | Agent | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **EtherNet/IP** | Scanner | ✓ | ✓ | × | ✓ | NA | × |
| | Adapter | × | ✓ | × | ✓ | NA | × |

Table 69: Protocol Behavior x CPU State

**Notes:**

**Symbol ✓:** Protocol remains active and operating normally.

**Symbol ✗:** Protocol is disabled.

**MODBUS Symbol Slave/Server:** To keep the protocol communicating when the CPU isn't in RUN or after a breakpoint, it's need to check the option "*Keep the communication running on CPU stop*".

### 5.9.2. MODBUS RTU Master

This protocol is available for the Nexto Series CPUs in its serial channels. By selecting this option at MasterTool, the CPU becomes MODBUS communication master, allowing the access to other devices with the same protocol, when it is in the execution mode (*Run* Mode).

The available configuration uses of Symbolic Mapping where its variables are defined by their name.

The procedure to insert a protocol instance is found in detail in Mastertool's user manual or in the section Inserting a Protocol Instance. The remaining configuration steps are described below for each mode.

- Add the MODBUS RTU Master Protocol instance to the serial channel. To execute this procedure, see Inserting a Protocol Instance section.
- Configure the serial interface, choosing the transmission speed, the parity, the channel stop bits, among others configurations by a double click on the COM 1 serial channel (or COM 2 - if available). See Serial Interface section.

#### 5.9.2.1. MODBUS Master Protocol Configuration by Symbolic Mapping

To configure this protocol using symbolic mapping, you must perform the following steps:

- Configure the general parameters of the MODBUS Master protocol, like: transmission delay times and minimum interframe as in Figure 89.
- Add and configure devices via the General Parameters tab, defining the slave address, communication time-out and number of communication retries as can be seen in Figure 90.
- Add and configure the MODBUS mappings on Mappings tab as Figure 91, specifying the variable name, data type, and the data initial address, the data size and range are filled automatically.
- Add and configure the MODBUS requests as presented in Figure 92, specifying the function, the scan time of the request, the starting address (read/write), the data size (read/write) and generate diagnostic variables and disabling the request via the buttons at the bottom of the window.

##### *5.9.2.1.1. MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration*

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as:

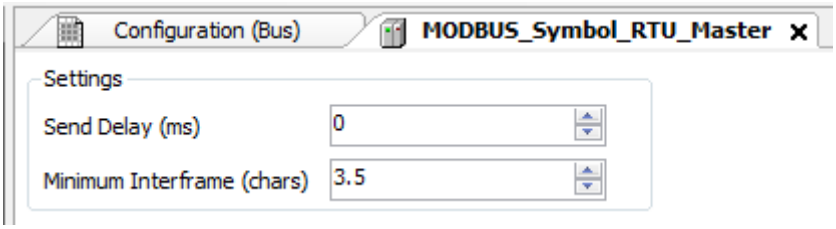



Figure 89: MODBUS RTU Master Configuration Screen

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Send Delay (ms)** | Delay for the answer transmission. | 0 | 0 to 65535 |

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Minimum Interframe (chars)** | Minimum silence time between different frames. | 3.5 | 3.5 to 100.0 |

Table 70: MODBUS RTU Master General Configurations

**Notes:**

**Send Delay:** The answer to a MODBUS protocol may cause problems in certain moments, as in the RS-485 interface or other half-duplex. Sometimes there is a delay between the slave answer time and the physical line silence (slave delay to put RTS in zero and put the RS-485 in high impedance state). To solve this problem, the master can wait the determined time in this field before sending the new request. Otherwise, the first bytes transmitted by the master could be lost.

**Minimum Interframe:** The MODBUS standard defines this time as 3.5 characters, but this parameter is configurable in order to attend the devices which do not follow the standard.

The MODBUS protocol diagnostics and commands configured are stored in *T_DIAG_MODBUS_RTU_MASTER_1* variables, which are described in table below:

| T_DIAG_MODBUS_RTU_MASTER_1.* | Size | Description |
|---|---|---|
| **Diagnostic Bits:** | | |
| tDiag.bRunning | BIT | The master is running. |
| tDiag.bNotRunning | BIT | The master is not running (see bit: bInterruptedByCommand). |
| tDiag.bInterruptedByCommand | BIT | The bit bNotRunning was enabled as the master was interrupted by the user through command bits. |
| tDiag.bConfigFailure | BIT | Configuration failure. |
| tDiag.bModuleFailure | BIT | Not implemented. |
| **Error codes:** | | |
| eErrorCode | SERIAL_STATUS (BYTE) | 0: there are no errors<br>1: invalid serial port<br>2: invalid serial port mode<br>3: invalid baud rate<br>4: invalid data bits<br>5: invalid parity<br>6: invalid stop bits<br>7: invalid modem signal parameter<br>8: invalid UART RX Threshold parameter<br>9: invalid time-out parameter<br>10: busy serial port<br>11: UART hardware error<br>12: remote hardware error<br>20: invalid transmission buffer size<br>21: invalid signal modem method<br>22: CTS time-out = true<br>23: CTS time-out = false<br>24: transmission time-out error<br>30: invalid reception buffer size<br>31: reception time-out error<br>32: flow control configured differently from manual<br>33: invalid flow control for the configured serial port<br>34: data reception not allowed in normal mode<br>35: data reception not allowed in extended mode<br>36: DCD interruption not allowed<br>37: CTS interruption not allowed<br>38: DSR interruption not allowed<br>39: serial port not configured<br>50: internal error in the serial port |
| **Command bits, automatically initialized:** | | |
| tCommand.bStop | BIT | Stop master. |
| tCommand.bRestart | BIT | Restart master. |
| tCommand.bResetCounter | BIT | Restart diagnostics statistics (counters). |
| **Communication Statistics:** | | |
| tStat.wTXRequests | WORD | Counter of request transmitted by the master (0 to 65535). |

| T_DIAG_MODBUS_RTU_MASTER_1.* | Size | Description |
|---|---|---|
| tStat.wRXNormalResponses | WORD | Counter of normal responses received by the master (0 to 65535). |
| tStat.wRXExceptionResponses | WORD | Counter of responses with exception codes received by the master (0 to 65535). |
| tStat.wRXIllegalResponses | WORD | Counter of illegal responses received by master – invalid syntax, not enough received bytes, invalid CRC – (0 to 65535). |
| tStat.wRXOverrunErrors | WORD | Counter of overrun errors during reception - UART FIFO or RX line – (0 to 65535). |
| tStat.wRXIncompleteFrames | WORD | Counter of answers with construction errors, parity or failure during reception (0 to 65535). |
| tStat.wCTSTimeOutErrors | WORD | Counter of CTS time-out error, using RTS/CTS handshake, during transmission (0 to 65535). |

Table 71: MODBUS RTU Master Diagnostics

**Note:**

**Counters:** All MODBUS RTU Master diagnostics counters return to zero when the limit value 65535 is exceeded.

#### *5.9.2.1.2. Devices Configuration – Symbolic Mapping configuration*

The devices configuration, shown on figure below, follows the following parameters:

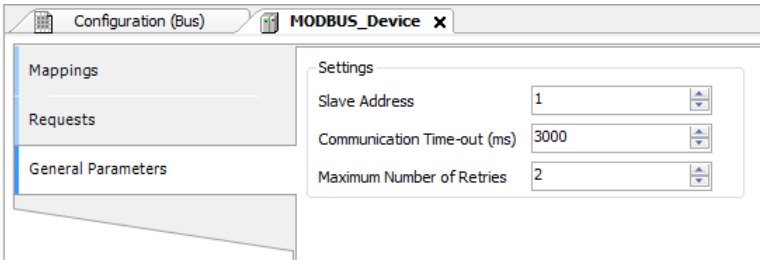



Figure 90: Device General Parameters Settings

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Slave Address** | MODBUS slave address | 1 | 0 to 255 |
| **Communication Time-out (ms)** | Defines the application level time-out | 3000 | 10 to 65535 |
| **Maximum Number of Retries** | Defines the numbers of retries before reporting a communication error | 2 | 0 to 9 |

Table 72: Device Configurations

**Notes:**

**Slave Address:** According to the MODBUS standard, the valid slave addresses are from 0 to 247, where the addresses from 248 to 255 are reserved. When the master sends a writing command with the address configured as zero, it is making broadcast requests in the network.

**Communication Time-out:** The communication time-out is the time that the master waits for a response from the slave to the request. For a MODBUS RTU master device it must be taken into account at least the following system variables: the time it takes the slave to transmit the frame (according to the baud rate), the time the slave takes to process the request and the response sending delay if configured in the slave. It is recommended that the time-out is equal to or greater than the time to transmit the frame plus the delay of sending the response and twice the processing time of the request. For more information, see Communication Performance section.

**Maximum number of retries:** Sets the number of retries before reporting a communication error. For example, if the slave does not respond to a request and the master is set to send three retries, the error counter number is incremented by one unit when the execution of these three retries. After the increase of the communication error trying to process restarts and if the number of retries is reached again, new error will increment the counter.

#### 5.9.2.1.3. *Mappings Configuration – Symbolic Mapping Settings*

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

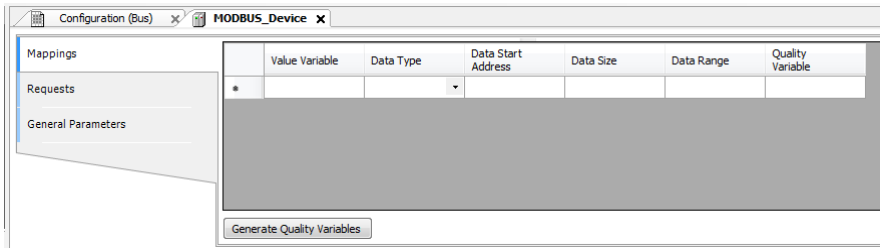



Figure 91: MODBUS Data Mappings Screen

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Value Variable** | Symbolic variable name | - | Name of a variable declared in a program or GVL |
| **Data Type** | MODBUS data type | - | Coil - Write (1 bit)<br>Coil - Read (1 bit)<br>Holding Register - Write (16 bits)<br>Holding Register - Read (16 bits)<br>Holding Register – Mask AND (16 bits)<br>Holding Register – Mask OR (16 bits)<br>Input Register (16 bits)<br>Input Status (1 bit) |
| **Data Start Address** | Initial address of the MODBUS data | - | 1 to 65536 |
| **Data Size** | Size of the MODBUS data | - | 1 to 65536 |
| **Data Range** | The address range of configured data | - | - |

Table 73: MODBUS Mappings Settings

**Notes:**

**Value Variable:** this field is used to specify a symbolic variable in MODBUS relation.

**Data type:** this field is used to specify the data type used in the MODBUS relation.

| Data Type | Size [bits] | Description |
|---|---|---|
| **Coil - Write** | 1 | Writing digital output. |
| **Coil - Read** | 1 | Reading digital output. |
| **Holding Register - Write** | 16 | Writing analog output. |
| **Holding Register - Read** | 16 | Reading analog output. |
| **Holding Register - Mask AND** | 16 | Analog output which can be read or written with AND mask. |
| **Holding Register - Mask OR** | 16 | Analog output which can be read or written with OR mask. |
| **Input Register** | 16 | Analog input which can be only read. |
| **Input Status** | 1 | Digital input which can be only read. |

Table 74: Data Types Supported in MODBUS

**Data Start Address:** Data initial address of a MODBUS mapping.

**Data Size:** The size value specifies the maximum amount of data that a MODBUS interface can access, from the initial address. Thus, to read a continuous address range, it is necessary that all addresses are declared in a single interface. This field varies with the MODBUS data type configured.

**Data Range:** This field shows to the user the memory address range used by the MODBUS interface.

#### *5.9.2.1.4. Requests Configuration – Symbolic Mapping Settings*

The configuration of the MODBUS requests, viewed in figure below, follow the parameters described in table below:

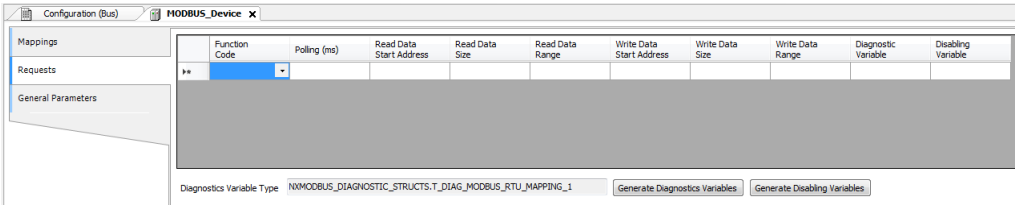



Figure 92: Data Requests Screen MODBUS Master

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Function Code** | MODBUS function type | - | 01 – Read Coils<br>02 – Read Input Status<br>03 – Read Holding Registers<br>04 – Read Input Registers<br>05 – Write Single Coil<br>06 – Write Single Register<br>15 – Write Multiple Coils<br>16 – Write Multiple Registers<br>22 – Mask Write Register<br>23 – Read/Write Multiple Registers |
| **Polling (ms)** | Communication period (ms) | 100 | 0 to 3600000 |
| **Read Data Start Address** | Initial address of the MODBUS read data | - | 1 to 65536 |
| **Read Data Size** | Size of MODBUS Read data | - | Depends on the function used |
| **Read Data Range** | MODBUS Read data address range | - | 0 to 2147483646 |
| **Write Data Start Address** | Initial address of the MODBUS write data | - | 1 to 65536 |
| **Write Data Size** | Size of MODBUS Write data | - | Depends on the function used |
| **Write Data Range** | MODBUS Write data address range | - | 0 to 2147483647 |
| **Diagnostic Variable** | Diagnostic variable name | - | Name of a variable declared in a program or GVL |
| **Disabling Variable** | Variable used to disable MODBUS relation | - | Field for symbolic variable used to disable, individually, MODBUS requests configured. This variable must be of type BOOL. The variable can be simple or array element and can be in structures. |

Table 75: MODBUS Relations Configuration

**Notes:**

**Setting:** the number of factory default settings and the values for the column Options may vary according to the data type and MODBUS function (FC).

**Function Code:** MODBUS (FC) functions available are the following:

| Code | | Description |
|---|---|---|
| **DEC** | **HEX** | |
| 1 | 0x01 | Read Coils (FC 01) |
| 2 | 0x02 | Read Input Status (FC 02) |
| 3 | 0x03 | Read Holding Registers (FC 03) |
| 4 | 0x04 | Read Input Registers (FC 04) |
| 5 | 0x05 | Write Single Coil (FC 05) |
| 6 | 0x06 | Write Single Holding Register (FC 06) |
| 15 | 0x0F | Write Multiple Coils (FC 15) |
| 16 | 0x10 | Write Multiple Holding Registers (FC 16) |
| 22 | 0x16 | Mask Write Holding Register (FC 22) |
| 23 | 0x17 | Read/Write Multiple Holding Registers (FC 23) |

Table 76: MODBUS Functions Supported by Nexto CPUs

**Polling:** this parameter indicates how often the communication set for this request must be performed. By the end of a communication will be awaited a time equal to the value configured in the field polling and after that, a new communication will be executed.

**Read Data Start Address:** field for the initial address of the MODBUS read data.

**Read Data Size:** the minimum value for the read data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Read Coils (FC 01): 2000
- Read Input Status (FC 02): 2000
- Read Holding Registers (FC 03): 125
- Read Input Registers (FC 04): 125
- Read/Write Multiple Registers (FC 23): 121

**Read Data Range:** this field shows the MODBUS read data range configured for each request. The initial address, along with the read data size will result in the range of read data for each request.

**Write Data Start Address:** field for the initial address of the MODBUS write data.

**Write Data Size:** the minimum value for the write data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Write Single Coil (FC 05): 1
- Write Single Register (FC 06): 1
- Write Multiple Coils (FC 15): 1968
- Write Multiple Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Multiple Registers (FC 23): 121

**Write Data Range:** this field shows the MODBUS write data range configured for each request. The initial address, along with the read data size will result in the range of write data for each request.

**Diagnostic Variable:** The MODBUS request diagnostics configured by symbolic mapping are stored in variables of type *T_DIAG_MODBUS_RTU_MAPPING_1* for Master devices and *T_DIAG_MODBUS_ETH_CLIENT_1* for Client devices.

| T_DIAG_MODBUS_RTU_MAPPING_1.* | Size | Description |
|---|---|---|
| **Communication Status Bits:** | | |
| byStatus.bCommIdle | BIT | Communication idle (waiting to be executed). |
| byStatus.bCommExecuting | BIT | Active communication. |
| byStatus.bCommPostponed | BIT | Communication deferred, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of time-out. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true. |
| byStatus.bCommDisabled | BIT | Communication disabled. The bCommIdle bit is restarted in this condition. |
| byStatus.bCommOk | BIT | Communication terminated previously was held successfully. |
| byStatus.bCommError | BIT | Communication terminated previously had an error. Check error code. |
| **Last error code (enabled when bCommError = true):** | | |
| eLastErrorCode | MASTER_ERROR_CODE (BYTE) | Informs the possible cause of the last error in the MODBUS mapping. Consult Table 95 for further details. |
| **Last exception code received by master:** | | |
| eLastExceptionCode | MODBUS_EXCEPTION (BYTE) | NO_EXCEPTION (0)<br>FUNCTION_NOT_SUPPORTED (1)<br>MAPPING_NOT_FOUND (2)<br>ILLEGAL_VALUE (3)<br>ACCESS_DENIED (128)*<br>MAPPING_DISABLED (129)*<br>IGNORE_FRAME (255)* |
| **Communication statistics:** | | |
| wCommCounter | WORD | Communications counter terminated, with or without errors. The user can test when communication has finished testing the variation of this counter. When the value 65535 is reached, the counter returns to zero. |
| wCommErrorCounter | WORD | Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero. |

| T_DIAG_MODBUS_RTU_MAPPING_1.* | Size | Description |
|---|---|---|

Table 77: MODBUS RTU Relations Diagnostics

**Notes:**

**Exception Codes:** The exception codes presented in this field are values returned by the slave. The definitions of the exception codes 128, 129 and 255 presented in the table are valid only when using Altus slaves. Slaves from other manufacturers might use other definitions for each code.

**Disabling Variable:** variable of Boolean type used to disable, individually, MODBUS requests configured on request tab via button at the bottom of the window. The request is disabled when the variable, corresponding to the request, is equal to 1, otherwise the request is enabled.

**Last Error Code:** The codes for the possible situations that cause an error in the MODBUS communication can be consulted below:

| Code | Enumerable | Description |
|---|---|---|
| **1** | ERR_EXCEPTION | Reply is in an exception code (see eLastExceptionCode = Exception Code). |
| **2** | ERR_CRC | Reply with invalid CRC. |
| **3** | ERR_ADDRESS | MODBUS address not found. The address that replied the request was different than expected. |
| **4** | ERR_FUNCTION | Invalid function code. The reply's function code was different than expected. |
| **5** | ERR_FRAME_DATA_COUNT | The amount of data in the reply was different than expected. |
| **7** | ERR_NOT_ECHO | The reply is not an echo of the request (FC 05 and 06). |
| **8** | ERR_REFERENCE_NUMBER | Invalid reference number (FC 15 and 16). |
| **9** | ERR_INVALID_FRAME_SIZE | Reply shorter than expected. |
| **20** | ERR_CONNECTION | Error while establishing connection. |
| **21** | ERR_SEND | Error during transmission stage. |
| **22** | ERR_RECEIVE | Error during reception stage. |
| **40** | ERR_CONNECTION_TIMEOUT | Application level time-out during connection. |
| **41** | ERR_SEND_TIMEOUT | Application level time-out during transmission. |
| **42** | ERR_RECEIVE_TIMEOUT | Application level time-out while waiting for reply. |
| **43** | ERR_CTS_OFF_TIMEOUT | Time-out while waiting CTS = false in transmission. |
| **44** | ERR_CTS_ON_TIMEOUT | Time-out while waiting CTS = true in transmission. |
| **128** | NO_ERROR | No error since startup. |

Table 78: MODBUS Relations Error Codes

**ATTENTION**

Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of a Master MODBUS RTU instance and any other MODBUS task will stop running at the moment that it tries to perform a writing in a memory area. It occurs in order to keep the consistency of the memory areas data while a MainTask is not running.

### 5.9.3. MODBUS RTU Slave

This protocol is available for the Nexto Series on its serial channels. At selecting this option in Mastertool, the CPU becomes a MODBUS communication slave, allowing the connection with MODBUS RTU master devices. This protocol is available only in execution mode (*Run* Mode).

The procedure to insert an instance of the protocol is found in detail in Mastertool's user manual. The remaining configuration steps are described below for each mode:

- Add the MODBUS RTU Slave Protocol instance to the serial channel. To execute this procedure see Inserting a Protocol Instance section.
- Configure the serial interface, choosing the communication speed, the parity, the stop bits channel, among others. See Serial Interface section.

#### 5.9.3.1. MODBUS Slave Protocol Configuration via Symbolic Mapping

To configure this protocol using symbolic mapping, you must perform the following steps:

- Configure the MODBUS slave protocol general parameters, as: slave address and communication times (available at the Slave advanced configurations button).
- Add and configure MODBUS relations, specifying the variable name, MODBUS data type and data initial address. Automatically, the data size and range will be filled, in accordance to the variable type declared.

##### *5.9.3.1.1. MODBUS Slave Protocol General Parameters – Configuration via Symbolic Mapping*

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as.

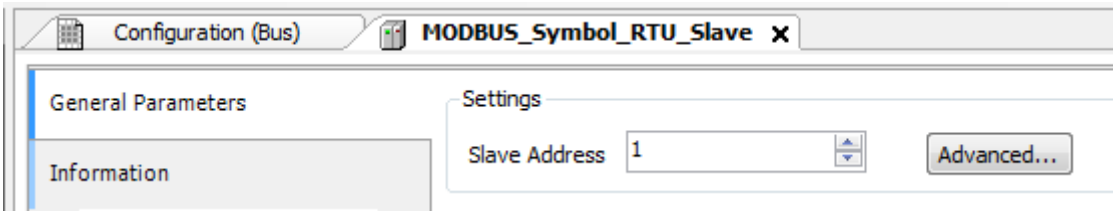



Figure 93: MODBUS RTU Slave Configuration Screen

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Slave Address** | MODBUS slave address | 1 | 1 to 255 |

Table 79: Slave Configurations

The MODBUS slave protocol communication times, found in the *Advanced...* button on the configuration screen, are divided in: *Task Cycle*, *Send Delay* and *Minimum Interframe* as shown in figure below and in table below.

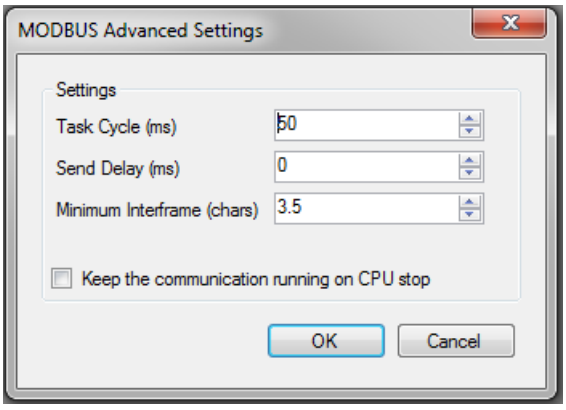



Figure 94: Modbus Slave Advanced Configurations

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Task Cycle (ms)** | Time for the instance execution within the cycle, without considering its own execution time | 50 | 20 to 100 |
| **Send Delay (ms)** | Delay for the transmission response | 0 | 0 to 65535 |
| **Minimum Interframe (chars)** | Minimum silence time between different frames | 3.5 | 3.5 to 100.0 |

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Keep the communication running on CPU stop** | Enable the MODBUS Symbol Slave to run while the CPU is in STOP or standing in a breakpoint | Unchecked | Checked or unchecked |

Table 80: Modbus Slave Advanced Configurations

**Notes:**

**Task Cycle:** the user will have to be careful when changing this parameter as it interferes directly in the answer time, data volume for scan and mainly in the CPU resources balance between communications and other tasks.

**Send Delay:** the answer to a MODBUS protocol may cause problems in certain moments, as in the RS-485 interface or other half-duplex. Sometimes there is a delay between the time of the request from the master and the silence on the physical line (slave delay to put RTS in zero and put the RS-485 in high impedance state). To solve this problem, the master can wait the determined time in this field before sending the new request. On the opposite case, the first bytes transmitted by the master could be lost.

**Minimum Interframe:** the MODBUS standard defines this time as 3.5 characters, but this parameter is configurable in order to attend the devices which don't follow the standard.

The MODBUS Slave protocol diagnostics and commands configured by symbolic mapping are stored in variables described in table below (*T_DIAG_MODBUS_RTU_SLAVE_1*):

| T_DIAG_MODBUS_RTU_SLAVE_1.* | Size | Description |
|---|---|---|
| **Diagnostic Bits:** | | |
| tDiag.bRunning | BIT | The slave is in execution mode. |
| tDiag.bNotRunning | BIT | The slave is not in execution (see bit: bInterruptedByCommand). |
| tDiag.bInterruptedByCommand | BIT | The bit bNotRunning was enabled as the slave was interrupted by the user through command bits. |
| tDiag.bConfigFailure | BIT | Configuration failure. |
| tDiag.bModuleFailure | BIT | Not implemented. |
| **Error codes:** | | |
| eErrorCode | SERIAL_STATUS (BYTE) | 0: there are no errors<br>1: invalid serial port<br>2: invalid serial port mode<br>3: invalid baud rate<br>4: invalid data bits<br>5: invalid parity<br>6: invalid stop bits<br>7: invalid modem signal parameter<br>8: invalid UART RX Threshold parameter<br>9: invalid time-out parameter<br>10: busy serial port<br>11: UART hardware error<br>12: remote hardware error<br>20: invalid transmission buffer size<br>21: invalid signal modem method<br>22: CTS time-out = true<br>23: CTS time-out = false<br>24: transmission time-out error<br>30: invalid reception buffer size<br>31: reception time-out error<br>32: flow control configured differently from manual<br>33: invalid flow control for the configured serial port<br>34: data reception not allowed in normal mode<br>35: data reception not allowed in extended mode<br>36: DCD interruption not allowed<br>37: CTS interruption not allowed<br>38: DSR interruption not allowed<br>39: serial port not configured<br>50: internal error in the serial port |
| **Command bits, automatically initialized:** | | |
| tCommand.bStop | BIT | Stop slave. |

| T_DIAG_MODBUS_RTU_SLAVE_1.* | Size | Description |
|---|---|---|
| tCommand.bRestart | BIT | Restart slave. |
| tCommand.bResetCounter | BIT | Restart diagnostics statistics (counters). |
| **Communication Statistics:** | | |
| tStat.wRXRequests | WORD | Counter of normal requests received by the slave and answered normally. In case of a broadcast command, this counter is incremented, but it is not transmitted (0 to 65535). |
| tStat.wTXExceptionResponses | WORD | Counter of normal requests received by the slave and answered with exception code. In case of a broadcast command, this counter is incremented, but it isn't transmitted (0 to 65535). |
| tStat.wRXFrames | WORD | Counter of frames received by the slave. It's considered a frame something which is processed and it is followed by a Minimum Interframe Silence, in other words, an illegal message is also computed (0 to 65535). |
| tStat.wRXIllegalRequests | WORD | Illegal request counter. These are frames which start with address 0 (broadcast) or with the MODBUS slave address, but aren't legal requests – invalid syntax, smaller frames, invalid CRC – (0 to 65535). |
| tStat.wRXOverrunErrors | WORD | Counter of frames with overrun errors during reception – UART FIFO or RX line – (0 to 65535). |
| tStat.wRXIncompleteFrames | WORD | Counter of frames with construction errors, parity or failure during reception (0 to 65535). |
| tStat.wCTSTimeOutErrors | WORD | Counter of CTS time-out error, using the RTS/CTS handshake, during the transmission (0 to 65535). |

Table 81: MODBUS RTU Slave Diagnostic

**Note:**
**Counters:** all MODBUS RTU Slave diagnostics counters return to zero when the limit value 65535 is exceeded.

#### *5.9.3.1.2. Configuration of the Relations – Symbolic Mapping Setting*

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

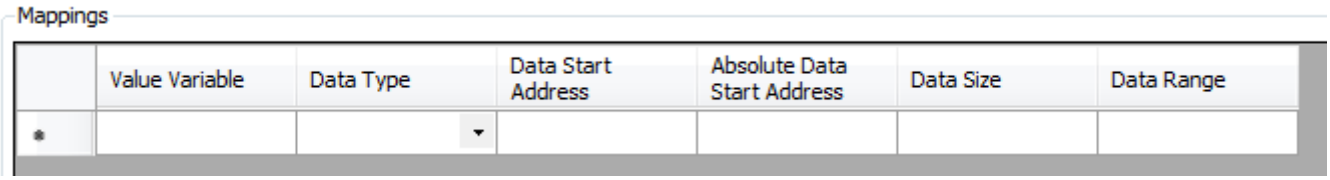



Figure 95: MODBUS Data Mappings Screen

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Value Variable** | Symbolic variable name | - | Name of a variable declared in a program or GVL |
| **Data Type** | MODBUS data type | - | Coil<br>Input Status<br>Holding Register<br>Input Register |
| **Data Start Address** | MODBUS data initial address | - | 1 to 65536 |
| **Absolute Data Start Address** | Absolute initial address of MODBUS data according to its type | - | - |
| **Data Size** | MODBUS data size | - | 1 to 65536 |
| **Data Range** | Data address range configured | - | - |

Table 82: MODBUS Mappings Configurations

**Notes:**

**Value Variable:** this field is used to specify a symbolic variable in MODBUS relation.

**Data Type:** this field is used to specify the data type used in the MODBUS relation.

| Data Type | Size [bits] | Description |
|---|---|---|
| **Coil** | 1 | Digital output that can be read or written. |
| **Input Status** | 1 | Digital input (read only). |
| **Holding Register** | 16 | Analog output that can be read or written. |
| **Input Register** | 16 | Analog input (read only). |

Table 83: MODBUS data types supported by Nexto CPUs

**Data Start Address:** data initial address of the MODBUS relation.

**Data Size:** the Data Size value sets the maximum amount of data that a MODBUS relation can access from the initial address. Thus, in order to read a continuous range of addresses, it is necessary that all addresses are declared in a single relation. This field varies according to the configured type of MODBUS data.

**Data Range:** this field shows the user the memory address range used by the MODBUS relation.

**ATTENTION**

Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of a MODBUS RTU Slave instance and any other MODBUS task will stop running at the moment that it tries to perform a writing in a memory area. It occurs in order to keep the consistency of the memory areas data while a MainTask is not running.

### 5.9.4. MODBUS Ethernet

The multi-master communication allows the Nexto CPUs to read or write MODBUS variables in other controllers or HMIs compatible with the MODBUS TCP protocol or MODBUS RTU via TCP. The Nexto CPU can, at the same time, be client and server in the same communication network, or even have more instances associated to the Ethernet interface. It does not matter if they are MODBUS TCP or MODBUS RTU via TCP, as described on Table 67.

The figure below represents some of the communication possibilities using the MODBUS TCP protocol simultaneously with the MODBUS RTU via TCP protocol.

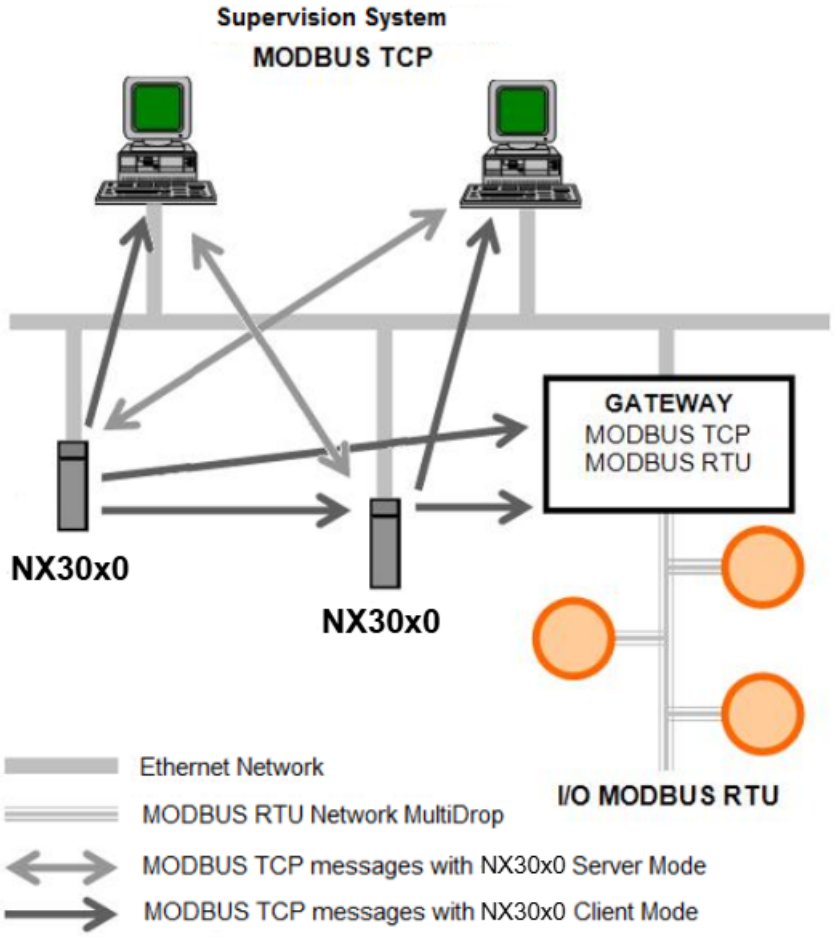



Figure 96: MODBUS TCP Communication Network

The association of MODBUS variables with CPU symbolic variables is made by the user through relations definition via Mastertool configuration tool. It's possible to configure up to 32 relations for the server mode and up to 128 relations for the client mode. The relations in client mode, on the other hand, must respect the data maximum size of a MODBUS function: 125 registers (input registers or holding registers) or 2000 bits (coils or input status). This information is detailed in the description of each protocol.

All relations, in client mode or server mode, can be disabled through direct representation variables (%Q) identified as Disabling Variables by Mastertool. The disabling may occur through general bits which affect all relations of an operation mode, or through specific bits, affecting specific relations.

For the server mode relations, IP addresses clusters can be defined with writing and reading allowance, called filters. This is made through the definition of an IP network address and of a subnet mask, resulting in a group of client IPs which can read and write in the relation variables. Reading/writing functions are filtered, in other words, they cannot be requested by any client, independent from the IP address. This information is detailed in the MODBUS Ethernet Server protocol.

When the MODBUS TCP protocol is used in the client mode, it's possible to use the multiple requests feature, with the same TCP connection to accelerate the communication with the servers. When this feature isn't desired or isn't supported by the server, it can be disabled (relation level action). It is important to emphasize that the maximum number of TCP connections between the client and server is 63. If some parameters are changed, inactive communications can be closed, which allows the opening of new connections.

The tables below bring, respectively, the complete list of data and MODBUS functions supported by the Nexto CPUs.

| Data Type | Size [bits] | Description |
|---|---|---|
| **Coil** | 1 | Digital output that can be read or written. |
| **Input Status** | 1 | Digital input (read only). |
| **Holding Register** | 16 | Analog output that can be read or written. |
| **Input Register** | 16 | Analog input (read only). |

| Data Type | Size [bits] | Description |
|---|---|---|

Table 84: MODBUS data types supported by Nexto CPUs

| Code | | Description |
|---|---|---|
| DEC | HEX | |
| 1 | 0x01 | Read Coils (FC 01) |
| 2 | 0x02 | Read Input Status (FC 02) |
| 3 | 0x03 | Read Holding Registers (FC 03) |
| 4 | 0x04 | Read Input Registers (FC 04) |
| 5 | 0x05 | Write Single Coil (FC 05) |
| 6 | 0x06 | Write Single Holding Register (FC 06) |
| 15 | 0x0F | Write Multiple Coils (FC 15) |
| 16 | 0x10 | Write Multiple Holding Registers (FC 16) |
| 22 | 0x16 | Mask Write Holding Register (FC 22) |
| 23 | 0x17 | Read/Write Multiple Holding Registers (FC 23) |

Table 85: MODBUS Functions Supported by Nexto CPUs

Independent of the configuration mode, the steps to insert an instance of the protocol and configure the Ethernet interface are equal. The remaining configuration steps are described below for each modality.

- Add one or more instances of the MODBUS Ethernet client or server protocol to Ethernet channel. To perform this procedure, refer to the section Inserting a Protocol Instance.
- Configure the Ethernet interface. To perform this procedure, see section Ethernet Interface.

### 5.9.5. MODBUS Ethernet Client

This protocol is available for all Nexto Series CPUs on its Ethernet channels. When selecting this option in Mastertool, the CPU becomes a MODBUS communication client, allowing the access to other devices with the same protocol, when it's in execution mode (*Run* Mode).

The procedure to insert an instance of the protocol is found in detail in Mastertool's user manual or on Inserting a Protocol Instance section.

#### 5.9.5.1. MODBUS Ethernet Client Configuration via Symbolic Mapping

To configure this protocol using *Symbolic Mapping*, it's necessary to execute the following steps:

- Configure the general parameters of MODBUS protocol client, with the Transmission Control Protocol (TCP) or RTU via TCP.
- Add and configure devices by setting IP address, port, address of the slave and time-out of communication (available on the Advanced Settings button of the device).
- Add and configure the MODBUS mappings, specifying the variable name, data type, data initial address, data size and variable that will receive the quality data.
- Add and configure the MODBUS request, specifying the desired function, the scan time of the request, the initial address (read/write), the size of the data (read/write), the variable that will receive the data quality and the variable responsible for disabling the request.

##### *5.9.5.1.1. MODBUS Client Protocol General Parameters – Configuration via Symbolic Mapping*

The general parameters, found on the MODBUS protocol configuration initial screen (figure below), are defined as:

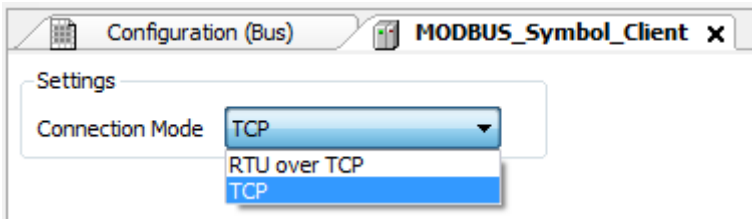



Figure 97: MODBUS Client General Parameters Configuration Screen

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Connection Mode** | Protocol selection | TCP | RTU via TCP<br>TCP |

Table 86: MODBUS Client General Configurations

The MODBUS Client protocol diagnostics and commands configured by symbolic mapping are described in table below, stored in *T_DIAG_MODBUS_ETH_CLIENT_1* variables:

| T_DIAG_MODBUS_ETH_CLIENT_1.* | Size | Description |
|---|---|---|
| **Diagnostic Bits:** | | |
| tDiag.bRunning | BIT | The client is in execution mode. |
| tDiag.bNotRunning | BIT | The client is not in execution mode (see bit bInterruptedByCommand). |
| tDiag.bInterruptedByCommand | BIT | The bit bNotRunning was enabled, as the client was interrupted by the user through command bits. |
| tDiag.bConfigFailure | BIT | Configuration failure. |
| tDiag.bModuleFailure | BIT | Indicates if there is failure in the module or the module is not present. |
| tDiag.bAllDevicesCommFailure | BIT | Indicates that all devices configured in the Client are in failure. |
| **Command bits, automatically initialized:** | | |
| tCommand.bStop | BIT | Stop client. |
| tCommand.bRestart | BIT | Restart client. |
| tCommand.bResetCounter | BIT | Restart the diagnostic statistics (counters). |
| **Communication Statistics:** | | |
| tStat.wTXRequests | WORD | Counter of number of requests transmitted by the client (0 to 65535). |
| tStat.wRXNormalResponses | WORD | Counter of normal answers received by the client (0 to 65535). |
| tStat.wRXExceptionResponses | WORD | Counter of answers with exception code (0 to 65535). |
| tStat.wRXIllegalResponses | WORD | Counter of illegal answers received by the client – invalid syntax, invalid CRC or not enough bytes received (0 to 65535). |

Table 87: MODBUS Client Protocol Diagnostics

**Note:**
**Counters:** all MODBUS TCP Client diagnostics counters return to zero when the limit value 65535 is exceeded.

#### *5.9.5.1.2. Device Configuration – Configuration via Symbolic Mapping*

The devices configuration, shown on figure below, follows the following parameters:

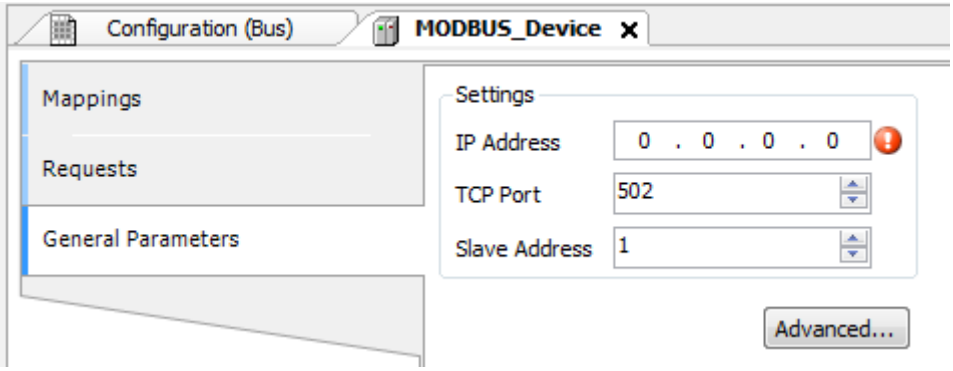



Figure 98: Device General Parameters Settings

| Configuration | Description | Default | Options |
|---|---|---|---|
| **IP Address** | Server IP address | 0.0.0.0 | 1.0.0.1 to 223.255.255.254 |
| **TCP Port** | TCP port | 502 | 2 to 65534 |
| **Slave Address** | MODBUS Slave address | 1 | 0 to 255 |

Table 88: MODBUS Client General Configurations

**Notes:**

**IP Address:** IP address of Modbus Server Device.

**TCP Port:** if there are multiple instances of the protocol added in a single Ethernet interface, different TCP ports must be selected for each instance. Some TCP ports, among the possibilities mentioned above, are reserved and therefore cannot be used. See table Reserved TCP/UDP ports.

**Slave address:** according to the MODBUS standard, the valid address range for slaves is 0 to 247, where addresses 248 to 255 are reserved. When the master sends a command of writing with the address set to zero, it is performing broadcast requests on the network.

The parameters in the advanced settings of the MODBUS Client device, found on the button *Advanced...* in the *General Parameters* tab are divided into: *Maximum Simultaneous Requests*, *Communication Time-out*, *Mode* of Connection Time-out and *Inactive Time*.

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Maximum Simultaneous Request** | Number of simultaneous request the client can ask from the server | 1 | 1 to 8 |
| **Communication Time-out (ms)** | Application level time-out in ms | 3000 | 10 to 65535 |
| **Mode** | Defines when the connection with the server finished by the client | Connecting stays open while a time-out does not occur. | Connecting stays open while a time-out does not occur.<br>Connection is closed at the end of each communication.<br>Connection is closed after an inactive time of (s): 10 to 3600. |
| **Inactive Time (s)** | Inactivity time | 10 | 10 to 3600 |

Table 89: MODBUS Client Advanced Configurations

**Notes:**

**Maximum Simultaneous Requests:** it is used with a high scan cycle. This parameter is fixed in 1 (not editable) when the configured protocol is MODBUS RTU over TCP.

**Communication Time-out:** the Communication time-out is the time that the client will wait for a server response to the request. For a MODBUS Client device, two variables of the system must be considered: the time the server takes to process a request and the response sending delay in case it is set in the server. It is recommended that the time-out is equal or higher than twice the sum of these parameters. For further information, check Communication Performance section.

**Mode:** defines when the connection with the server is finished by the client. Below follows the available options:

- **Connecting stays open while a time-out does not occur:** This option presents the same behavior of Client, close the connection due non response of a request by the Server before reaching the Communication Time-out.
- **Connection is closed at the end of each communication:** The connection is closed by the Client after finish each request.
- **Connection is closed after an Inactive Time:** The connection will be closed by the Client if it reach the Inactive Time without performing a request to the Server.

**Inactive Time:** inactivity connection time.

#### *5.9.5.1.3. Mappings Configuration – Configuration via Symbolic Mapping*

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

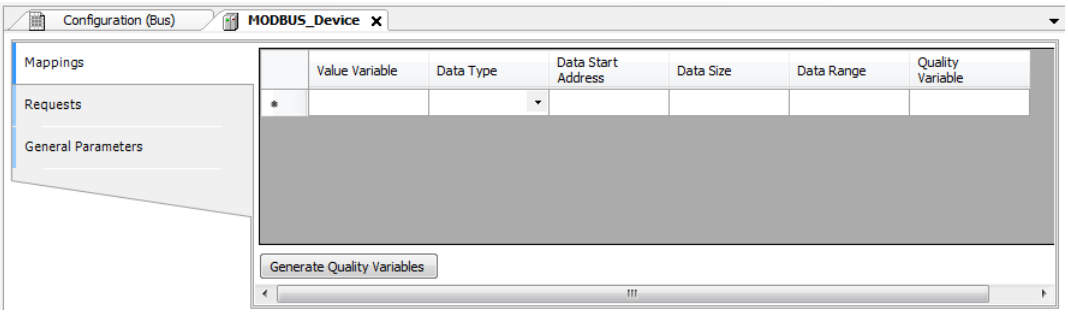



Figure 99: MODBUS Data Type

| Configuration | Description | Default | Options |
|---|---|---|---|
| **Value Variable** | Symbolic variable name | - | Name of a variable declared in a program or GVL |
| **Data Type** | MODBUS data type | - | Coil - Write (1 bit)<br>Coil - Read (1 bit)<br>Holding Register - Write (16 bits)<br>Holding Register - Read (16 bits)<br>Holding Register – Mask AND (16 bits)<br>Holding Register – Mask OR (16 bits)<br>Input Register (16 bits)<br>Input Status (1 bit) |
| **Data Start Address** | Initial address of the MODBUS data | - | 1 to 65536 |
| **Data Size** | Size of the MODBUS data | - | 1 to 65536 |
| **Data Range** | The address range of configured data | - | - |

Table 90: MODBUS Mappings Settings

**Notes:**

**Value Variable:** this field is used to specify a symbolic variable in MODBUS relation.

**Data type:** this field is used to specify the data type used in the MODBUS relation.

| Data Type | Size [bits] | Description |
|---|---|---|
| **Coil - Write** | 1 | Writing digital output. |
| **Coil - Read** | 1 | Reading digital output. |
| **Holding Register - Write** | 16 | Writing analog output. |
| **Holding Register - Read** | 16 | Reading analog output. |

| Data Type | Size [bits] | Description |
|---|---|---|
| **Holding Register - Mask AND** | 16 | Analog output which can be read or written with AND mask. |
| **Holding Register - Mask OR** | 16 | Analog output which can be read or written with OR mask. |
| **Input Register** | 16 | Analog input which can be only read. |
| **Input Status** | 1 | Digital input which can be only read. |

Table 91: Data Types Supported in MODBUS

**Data Start Address:** Data initial address of a MODBUS mapping.

**Data Size:** The size value specifies the maximum amount of data that a MODBUS interface can access, from the initial address. Thus, to read a continuous address range, it is necessary that all addresses are declared in a single interface. This field varies with the MODBUS data type configured.

**Data Range:** This field shows to the user the memory address range used by the MODBUS interface.

#### *5.9.5.1.4. Requests Configuration – Configuration via Symbolic Mapping*

The configuration of the MODBUS requests, viewed in figure below, follow the parameters described in table below:

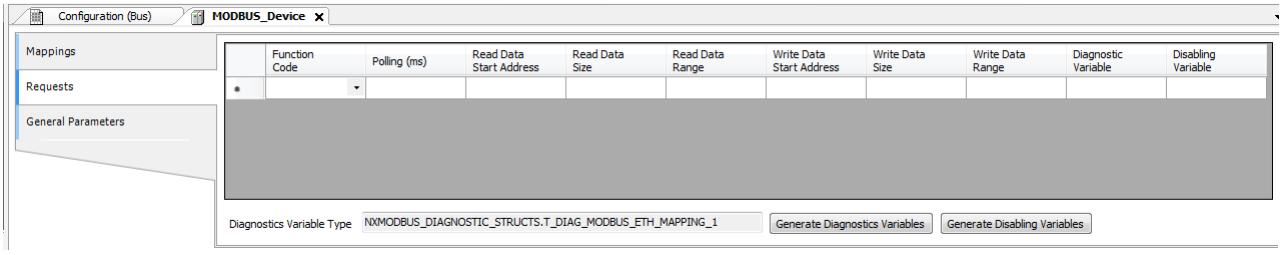



Figure 100: MODBUS Data Request Screen

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Function Code** | MODBUS function type | - | 01 – Read Coils<br>02 – Read Input Status<br>03 – Read Holding Registers<br>04 – Read Input Registers<br>05 – Write Single Coil<br>06 – Write Single Register<br>15 – Write Multiple Coils<br>16 – Write Multiple Registers<br>22 – Mask Write Register<br>23 – Read/Write Multiple Registers |
| **Polling (ms)** | Communication period (ms) | 100 | 0 to 3600000 |
| **Read Data Start Address** | Initial address of the MODBUS read data | - | 1 to 65536 |
| **Read Data Size** | Size of MODBUS Read data | - | Depends on the function used |
| **Read Data Range** | MODBUS Read data address range | - | 0 to 2147483646 |
| **Write Data Start Address** | Initial address of the MODBUS write data | - | 1 to 65536 |
| **Write Data Size** | Size of MODBUS Write data | - | Depends on the function used |
| **Write Data Range** | MODBUS Write data address range | - | 0 to 2147483647 |
| **Diagnostic Variable** | Diagnostic variable name | - | Name of a variable declared in a program or GVL |

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Disabling Variable** | Variable used to disable MODBUS relation | - | Field for symbolic variable used to disable, individually, MODBUS requests configured. This variable must be of type BOOL. The variable can be simple or array element and can be in structures. |

Table 92: MODBUS Relations Configuration

**Notes:**

**Setting:** the number of factory default settings and the values for the column Options may vary according to the data type and MODBUS function (FC).

**Function Code:** MODBUS (FC) functions available are the following:

| Code | | Description |
|---|---|---|
| DEC | HEX | |
| 1 | 0x01 | Read Coils (FC 01) |
| 2 | 0x02 | Read Input Status (FC 02) |
| 3 | 0x03 | Read Holding Registers (FC 03) |
| 4 | 0x04 | Read Input Registers (FC 04) |
| 5 | 0x05 | Write Single Coil (FC 05) |
| 6 | 0x06 | Write Single Holding Register (FC 06) |
| 15 | 0x0F | Write Multiple Coils (FC 15) |
| 16 | 0x10 | Write Multiple Holding Registers (FC 16) |
| 22 | 0x16 | Mask Write Holding Register (FC 22) |
| 23 | 0x17 | Read/Write Multiple Holding Registers (FC 23) |

Table 93: MODBUS Functions Supported by Nexto CPUs

**Polling:** this parameter indicates how often the communication set for this request must be performed. By the end of a communication will be awaited a time equal to the value configured in the field polling and after that, a new communication will be executed.

**Read Data Start Address:** field for the initial address of the MODBUS read data.

**Read Data Size:** the minimum value for the read data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Read Coils (FC 01): 2000
- Read Input Status (FC 02): 2000
- Read Holding Registers (FC 03): 125
- Read Input Registers (FC 04): 125
- Read/Write Multiple Registers (FC 23): 121

**Read Data Range:** this field shows the MODBUS read data range configured for each request. The initial address, along with the read data size will result in the range of read data for each request.

**Write Data Start Address:** field for the initial address of the MODBUS write data.

**Write Data Size:** the minimum value for the write data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Write Single Coil (FC 05): 1
- Write Single Register (FC 06): 1
- Write Multiple Coils (FC 15): 1968
- Write Multiple Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Multiple Registers (FC 23): 121

**Write Data Range:** this field shows the MODBUS write data range configured for each request. The initial address, along with the read data size will result in the range of write data for each request.

**Diagnostic Variable:** The MODBUS request diagnostics configured by symbolic mapping are stored in variables of type *T_DIAG_MODBUS_RTU_MAPPING_1* for Master devices and *T_DIAG_MODBUS_ETH_CLIENT_1* for Client devices.

| T_DIAG_MODBUS_ETH_MAPPING_1.* | Size | Description |
|---|---|---|
| **Communication Status Bits:** | | |
| byStatus.bCommIdle | BIT | Communication idle (waiting to be executed). |
| byStatus.bCommExecuting | BIT | Active communication. |
| byStatus.bCommPostponed | BIT | Communication deferred, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of time-out. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true. |
| byStatus.bCommDisabled | BIT | Communication disabled. The bCommIdle bit is restarted in this condition. |
| byStatus.bCommOk | BIT | Communication terminated previously was held successfully. |
| byStatus.bCommError | BIT | Communication terminated previously had an error. Check error code. |
| byStatus.bCommAborted | BIT | Previously terminated communication was interrupted due to connection failure. |
| **Last error code (enabled when bCommError = true):** | | |
| eLastErrorCode | MASTER_ERROR_CODE (BYTE) | Informs the possible cause of the last error in the MODBUS mapping. Consult Table 95 for further details. |
| **Last exception code received by master:** | | |
| eLastExceptionCode | MODBUS_EXCEPTION (BYTE) | NO_EXCEPTION (0)<br>FUNCTION_NOT_SUPPORTED (1)<br>MAPPING_NOT_FOUND (2)<br>ILLEGAL_VALUE (3)<br>ACCESS_DENIED (128)*<br>MAPPING_DISABLED (129)*<br>IGNORE_FRAME (255)* |
| **Communication statistics:** | | |
| wCommCounter | WORD | Communications counter terminated, with or without errors. The user can test when communication has finished testing the variation of this counter. When the value 65535 is reached, the counter returns to zero. |
| wCommErrorCounter | WORD | Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero. |

Table 94: MODBUS Client Relations Diagnostics

**Notes:**

**Exception Codes:** the exception codes show in this filed is the server returned values. The definitions of the exception codes 128, 129 and 255 are valid only with Altus slaves. For slaves from other manufacturers these exception codes can have different meanings.

**Disabling Variable:** field for the variable used to disable MODBUS requests individually configured within requests. The request is disabled when the variable, corresponding to the request, is equal to 1, otherwise the request is enabled.

**Last Error Code:** The codes for the possible situations that cause an error in the MODBUS communication can be consulted below:

| Code | Enumerable | Description |
|---|---|---|
| **1** | ERR_EXCEPTION | Reply is in an exception code (see eLastExceptionCode = Exception Code). |
| **2** | ERR_CRC | Reply with invalid CRC. |

| Code | Enumerable | Description |
|---|---|---|
| **3** | ERR_ADDRESS | MODBUS address not found. The address that replied the request was different than expected. |
| **4** | ERR_FUNCTION | Invalid function code. The reply's function code was different than expected. |
| **5** | ERR_FRAME_DATA_COUNT | The amount of data in the reply was different than expected. |
| **7** | ERR_NOT_ECHO | The reply is not an echo of the request (FC 05 and 06). |
| **8** | ERR_REFERENCE_NUMBER | Invalid reference number (FC 15 and 16). |
| **9** | ERR_INVALID_FRAME_SIZE | Reply shorter than expected. |
| **20** | ERR_CONNECTION | Error while establishing connection. |
| **21** | ERR_SEND | Error during transmission stage. |
| **22** | ERR_RECEIVE | Error during reception stage. |
| **40** | ERR_CONNECTION_TIMEOUT | Application level time-out during connection. |
| **41** | ERR_SEND_TIMEOUT | Application level time-out during transmission. |
| **42** | ERR_RECEIVE_TIMEOUT | Application level time-out while waiting for reply. |
| **43** | ERR_CTS_OFF_TIMEOUT | Time-out while waiting CTS = false in transmission. |
| **44** | ERR_CTS_ON_TIMEOUT | Time-out while waiting CTS = true in transmission. |
| **128** | NO_ERROR | No error since startup. |

Table 95: MODBUS Relations Error Codes

**ATTENTION**

Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS Ethernet Client instance task or any other MODBUS task will stop being executed at the moment it tries to write in the memory area. This occurs in order to maintain data consistency of memory areas while MainTask is not running.

#### 5.9.5.2. MODBUS Client Relation Start in Acyclic Form

To start a MODBUS Client relation in acyclic form, it is suggested the following method which can be implemented in a simple way in the user application program:

- Define the maximum polling time for the relations;
- Keep the relation normally disabled;
- Enable the relation at the moment the execution is desired;
- Wait for the confirmation of the relation execution finishing and, at this moment, disable it again.

### 5.9.6. MODBUS Ethernet Server

This protocol is available for all Nexto Series CPUs on its Ethernet channels. When selecting this option in Mastertool, the CPU becomes a MODBUS communication server, allowing the connection with MODBUS client devices. This protocol is only available when the CPU is in execution mode (*Run* Mode).

The procedure to insert an instance of the protocol is found in detail in Mastertool's user manual.

#### 5.9.6.1. MODBUS Server Ethernet Protocol Configuration for Symbolic Mapping

To configure this protocol using *Symbolic Mappings*, it is necessary to execute the following steps:

- Configure the MODBUS server protocol general parameters, as: TCP port, protocol selection, IP filters for Reading and Writing (available at the Filters Configuration button) and communication times (available at the Server Advanced Configurations button).
- Add and configure MODBUS mappings, specifying the variable name, data type, data initial address and data size.

The description of each configuration is related ahead in this section.

#### 5.9.6.1.1. MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as.

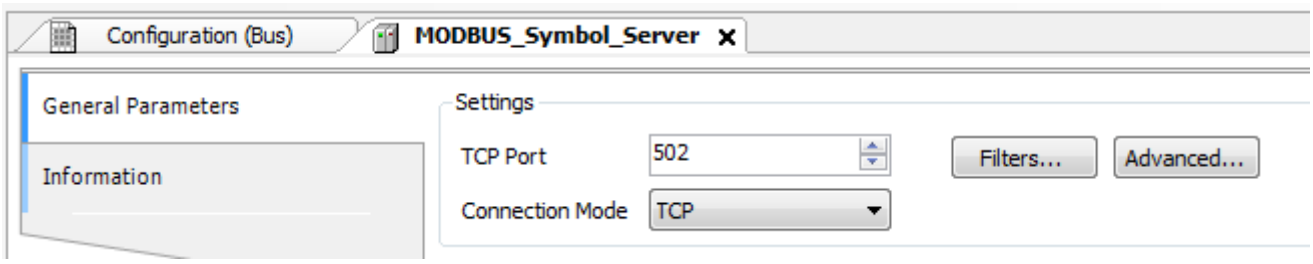



Figure 101: MODBUS Server General Parameters Configuration Screen

| Configuration | Description | Default | Options |
|---|---|---|---|
| **TCP Port** | TCP port | 502 | 2 to 65534 |
| **Connection Mode** | Protocol selection | TCP | RTU via TCP<br>TCP |

Table 96: MODBUS Server General Configurations

**Notes:**

**TCP Port:** if there are multiple instances of the protocol added in a single Ethernet interface, different TCP ports must be selected for each instance. Some TCP ports, among the possibilities mentioned above, are reserved and therefore cannot be used. See table Reserved TCP/UDP ports.

The settings present on the *Filters...* button, described in table below, are relative to the TCP communication filters:

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Write Filter IP Address** | Specifies a range of IPs with write access in the variables declared in the MODBUS interface. | 0.0.0.0 | 0.0.0.0 to 255.255.255.255 |
| **Write Filter Mask** | Specifies the subnet mask in conjunction with the IP filter parameter for writing. | 0.0.0.0 | 0.0.0.0 to 255.255.255.255 |
| **Read Filter IP Address** | Specifies a range of IPs with read access in the variables declared in the MODBUS interface. | 0.0.0.0 | 0.0.0.0 to 255.255.255.255 |
| **Read Filter Mask** | Specifies the subnet mask in conjunction with the IP filter parameter for reading. | 0.0.0.0 | 0.0.0.0 to 255.255.255.255 |

Table 97: IP Filters

**Note:**

**Filters:** filters are used to establish a range of IP addresses that have write or read access to MODBUS relations, being individually configured. The permission criteria is accomplished through a logical AND operation between the Write Filter Mask and the IP address of the client. If the result is the same as the Write Filter IP Address, the client is entitled to write. For example, if the Write Filter IP Address = 192.168.15.0 and the Write Filter Mask = 255.255.255.0, then only customers with IP address = 192.168.15.x shall be entitled. The same procedure is applied in the Read Filter parameters to define the read rights.

The communication times of the MODBUS server protocol, found on the *Advanced...* button of the configuration screen, are divided into: *Task Cycle* and *Connection Inactivity Time-out*.

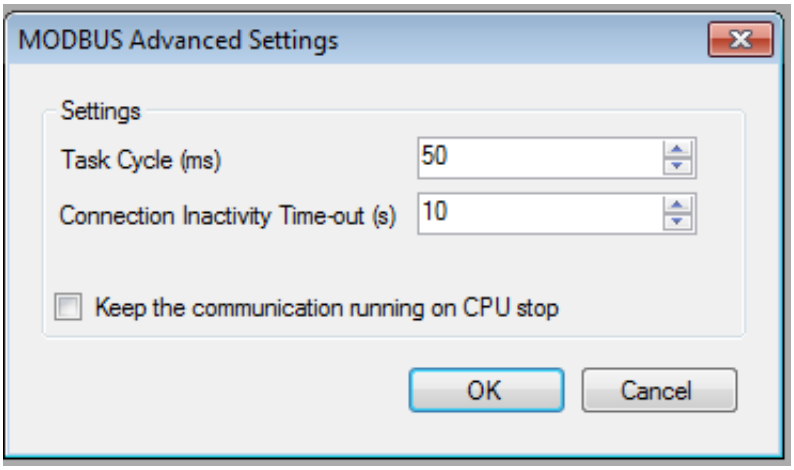



Figure 102: MODBUS Server Advanced Settings Configuration Screen

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Task Cycle (ms)** | Time for the instance execution within the cycle, without considering its own execution time | 50 | 5 to 100 |
| **Connection Inactivity Time-out (s)** | Maximum idle time between client and server before the connection is closed by the server | 10 | 1 to 3600 |
| **Keep the communication running on CPU stop.** | Enable the MODBUS Symbol Slave to run while the CPU is in STOP or after a breakpoint | Unmarked | Marked or Unmarked |

Table 98: MODBUS Server Advanced Configurations

**Notes:**

**Task Cycle:** the user has to be careful when changing this parameter as it interferes directly in the answer time, data volume for scanning and mainly in the CPU resources balance between communications and other tasks.

**Connection Inactivity Time-out:** this parameter was created in order to avoid that the maximum quantity of TCP connections is reached, imagining that inactive connections remain open on account of the most different problems. It indicates how long a connection (client or server) can remain open without being used (without exchanging communication messages). If the specified time is not reached, the connection is closed releasing an input in the connection table.

#### *5.9.6.1.2. MODBUS Server Diagnostics – Configuration via Symbolic Mapping*

The diagnostics and commands of the MODBUS server protocol configured by symbolic mapping are stored in variables described in table below (*T_DIAG_MODBUS_ETH_SERVER_1*):

| T_DIAG_MODBUS_ETH_SERVER_1.* | Size | Description |
|---|---|---|
| **Diagnostic Bits:** | | |
| tDiag.bRunning | BIT | The server is running. |
| tDiag.bNotRunning | BIT | The server is not running (see bit bInterruptedByCommand). |
| tDiag.bInterruptedByCommand | BIT | The bit bNotRunning was enabled, because the server was interrupted by the user through the command bit. |
| tDiag.bConfigFailure | BIT | Configuration failure. |
| tDiag.bModuleFailure | BIT | Indicates if there is failure in the module or the module is not present. |
| **Command bits, automatically initialized:** | | |
| tCommand.bStop | BIT | Stop the server. |
| tCommand.bRestart | BIT | Restart the server. |
| tCommand.bResetCounter | BIT | Reset diagnostics statistics (counters). |
| **Communication Statistics:** | | |

| T_DIAG_MODBUS_ETH_SERVER_1.* | Size | Description |
|---|---|---|
| tStat.wActiveConnections | WORD | Number of established connections between client and server (0 to 64). |
| tStat.wTimeoutClosedConnections | WORD | Connections counter, between the client and server, interrupted after a period of inactivity - time-out (0 to 65535). |
| tStat.wClientClosedConnections | WORD | Connections counter interrupted due to customer request (0 to 65535). |
| tStat.wRXFrames | WORD | Ethernet frames counter received by the server. An Ethernet frame can contain more than one request (0 to 65535). |
| tStat.wRXRequests | WORD | Requests received by the server counter and answered normally (0 to 65535). |
| tStat.wTXExceptionResponses | WORD | Requests received by the server counter and answered with exception codes (0 to 65535). |
| tStat.wRXIllegalRequests | WORD | Illegal requests counter (0 to 65535). |

Table 99: MODBUS Server Diagnostics

**Note:**
**Counters:** all counters of the MODBUS Ethernet Server Diagnostics return to zero when the limit value 65535 is exceeded.
**bModuleFailure:** Diagnosis implemented only for symbolic MODBUS.

#### *5.9.6.1.3. Mapping Configuration – Configuration via Symbolic Mapping*

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

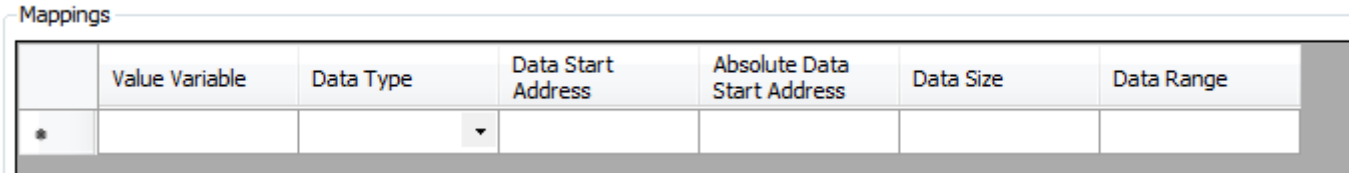



Figure 103: MODBUS Server Data Mappings Screen

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Value Variable** | Symbolic variable name | - | Name of a variable declared in a program or GVL |
| **Data Type** | MODBUS data type | - | Coil<br>Input Status<br>Holding Register<br>Input Register |
| **Data Start Address** | Starting address of the MODBUS data | - | 1 to 65536 |
| **Absolute Data Start Address** | Start address of absolute data of Modbus as its type | - | - |
| **Data Size** | Size of the MODBUS data | - | 1 to 65536 |
| **Data Range** | Data range address configured | - | - |

Table 100: MODBUS Ethernet Mappings Configuration

**Notes:**

**Value Variable:** this field is used to specify a symbolic variable in MODBUS relation.

**Data Type:** this field is used to specify the data type used in the MODBUS relation.

**Data Start Address:** data initial address of the MODBUS relation.

**Absolute Data Start Address:** absolute start address of the MODBUS data according to their type. For example, the Holding Register with address 5 has absolute address 400005. This field is read only and is available to assist in Client/Master MODBUS configuration that will communicate with this device. The values depend on the base address (offset) of each data type and allowed MODBUS address for each data type.

**Data Size:** the Data Size value sets the maximum amount of data that a MODBUS relation can access from the initial address. Thus, in order to read a continuous range of addresses, it is necessary that all addresses are declared in a single relation. This field varies according to the configured type of MODBUS data.

**Data Range:** is a read-only field and reports on the range of addresses that is being used by this mapping. It is formed by the sum of the fields *Data Start Address* and *Data Size*. There can be no range overlays with others mappings of the same *Data Type*.

**ATTENTION**

Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS Ethernet Server instance task or any other MODBUS task will stop being executed at the moment it tries to write in the memory area. This occurs in order to maintain data consistency of memory areas while MainTask is not running.

### 5.9.7. OPC DA Server

It's possible to communicate with the Nexto Series CPUs using the OPC DA (*Open Platform Communications Data Access*) technology. This open communication platform was developed to be the standard in industrial communications. Based on client/server architecture, it offers several advantages in project development and communication with automation systems.

A very common analogy to describe the OPC DA technology is of a printer. When correctly connected, the computer needs a driver to interface with the equipment. Similarly, the OPC helps with the interface between the supervision system and the field data on the PLC.

When it comes to project development, to configure the communication and exchange information between the systems is extremely simple using OPC DA technology. Using other drivers, based on addresses, it's necessary to create tables to relate tags from the supervision system with variables from the programmable controller. When the data areas are changed during the project, it's necessary to remap the variables and create new tables with the relations between the information on the PLC with the Supervisory Control And Data Acquisition system (SCADA).

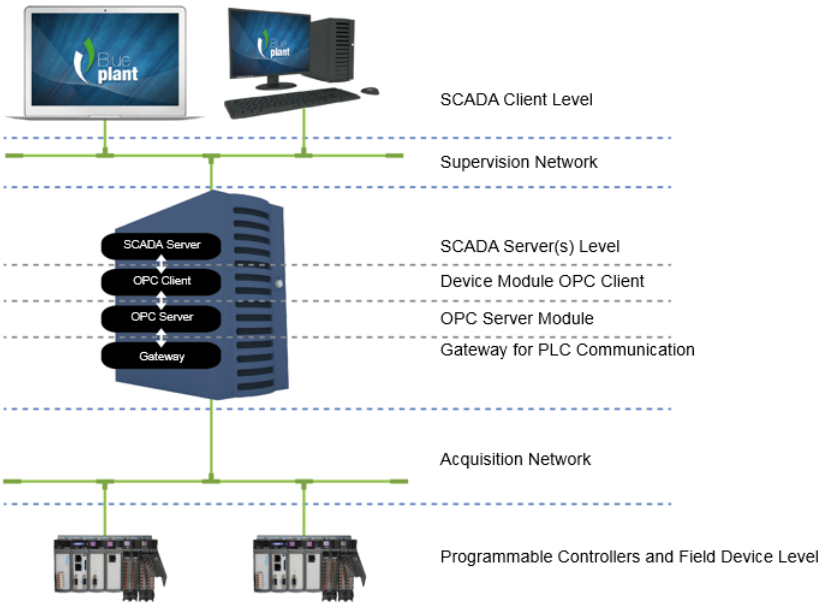



Figure 104: OPC DA Architecture

The figure above shows an architecture to communicate a SCADA system and PLCs in automation projects. All the roles present on a communication are explicit on this figure regardless of the equipment in which it's executed, since they can be done in the same equipment or in various ones. Each of the roles of this architecture are described on table below.

| Role | Description |
|---|---|
| **Programmable Controllers and Field Devices Level** | The field devices and the PLCs are where the operation state and plant control information are stored. The SCADA system access the information on these devices and store on the SCADA server, so that the SCADA clients can consult it during the plant operation. |
| **Acquisition Network** | The acquisition network is where the requests for data collected by field devices travel, to request the data collected from the field devices. |
| **Gateway for PLC Communication** | A gateway enables the communication between the OPC DA Server and Nexto Series PLCs. A gateway in the same subnet of the PLC is always necessary, as described in chapter Communication Settings of Mastertool's user manual. |
| **OPC Server Module** | The OPC DA Server is a Module responsible of receiving the OPC DA requests and translate them to the communication with the field devices. |
| **Device Module OPC Client** | The OPC Client Device module is responsible for the requests to the OPC DA Server using the OPC DA protocol. The collected data is stored on the SCADA Server database. |
| **SCADA Server Level** | The SCADA Server is responsible for connecting to the various communication devices and store the data collected by them on a database, so that it can be consulted by the SCADA Clients. |
| **Supervision Network** | The supervision network is the network through which the SCADA Clients are connected to the SCADA Servers. In a topology in which there aren't multiple Client or where the Server and the Client are installed on the same equipment, this kind of network doesn't exist. |

| Role | Description |
|---|---|
| **SCADA Client Level** | The SCADA Clients are responsible for requesting to the SCADA Servers the necessary data to be shown in a screen where the operation of a plant is being executed. Through then it is possible to execute readings and writings on data stored on the SCADA Server database. |

Table 101: Roles Description on an OPC DA Server Architecture

The relation between the tags on the supervision system and the process data on the controller variables is totally transparent. This means that, if there's an alteration on the data areas through the development of the project, it isn't necessary to rework the relations between the information on the PLC and the SCADA, just use the new variable provided by the PLC on the systems that request this data.

The use of OPC offers more productivity and connectivity with SCADA systems. It contributes with the reduction of applications development time and with the maintenance costs. It even makes possible the insertion of new data on the communication in a simplified form and with greater flexibility and interoperability between the automation system, due to the fact that it's an open standard.

The installation of the OPC DA Server is done altogether with Mastertool's installation, and its settings are done inside the tool. It's worth notice that the OPC is available only with the integrated Ethernet interface of the Nexto CPUs. The Ethernet expansion modules do not support this functionality.

#### 5.9.7.1. Creating a Project for OPC DA Communication

Unlike the communication with drivers such as MODBUS and PROFIBUS DP, to set an OPC DA communication it's only necessary to correctly set the node and indicate which variables will be used in the communication. There are two ways to indicate which variables of the project will be available in the OPC DA Server. In both cases it's necessary to add the object *Symbol Configuration* to the application, in case it isn't present. To add it, right-click over the object *Application* and select the option.

**ATTENTION**

The variables shown in the objects *IoConfig_Globals*, *IoConfig_Application_Mappings* and *IoConfig_Global_Mappings* are used internally for I/O control and shouldn't be used by the user.

**ATTENTION**

In addition to the variables declared at SFC language POUs, some implicitly created variables are also shown. To each step created, a type *IecSfc.SFCStepType* variable is created, where the step states can be monitored, namely whether it is active or not and the time that it's active as in norm IEC 61131-1. To each transition, a BOOL type variable is created that defines if the transition is true or false. These variables are shown in the object *Symbol Configuration* that can be provided access to the OPC Client.

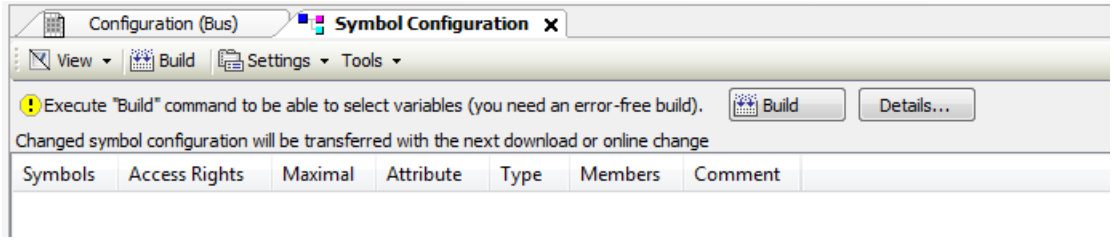



Figure 105: Symbol Configuration Object

The table below presents the descriptions of the *Symbol Configuration* object screen fields.

| Field | Description |
|---|---|
| **Symbols** | Variable identifier that will be provided to the OPC DA Server. |
| **Access Rights** | Indicates what the possible access right level are in the declared symbol. When not utilized, this column remains empty, and the access right level is maximum. Otherwise the access right level can be modified by clicking over this field. The possible options are:<br>Read only  <br>Write only  <br>Read and Write  |
| **Maximal** | Indicates the maximum access right level that is possible to assign to the variable. The symbols hold the same meanings from the ones in Access Rights. It's not possible to change it and it's indicated by the presence or not of the *attribute 'symbol'* |
| **Attribute** | Indicates if *attribute 'symbol'* is being used when the variable is declared. When not used, this column remains empty. For the cases in which the attribute is used, the behavior is the following:<br>attribute 'symbol' := 'read' the column shows  <br>attribute 'symbol' := 'write' the column shows  <br>attribute 'symbol' := 'readwrite' the column shows  |
| **Type** | Data type of the declared variable. |
| **Members** | When the data type is a Struct, a button is enabled in this column. Clicking on the button will allow the selection of which elements of that struct will be provided to the OPC DA Server. |
| **Comment** | Variable comment, inserted on the POU or GVL where the variable was declared. To show up as a variable comment here, the comment must be entered one line before the variable on the editor, while in text mode, or in the comment column when in tabular mode. |

Table 102: Symbol Configuration object screen fields description

When altering the project settings, such as adding or removing variables, it's necessary to run the command *Build*, in order to refresh the list of variables. This command must be executed until the message in Figure 105 disappear. After this, all available variables in the project, whether they are declared on POUs, GVLs or diagnostics, will be shown here and can be selected. The selected variables will be available on the OPC DA Server to be accessed by the Clients.

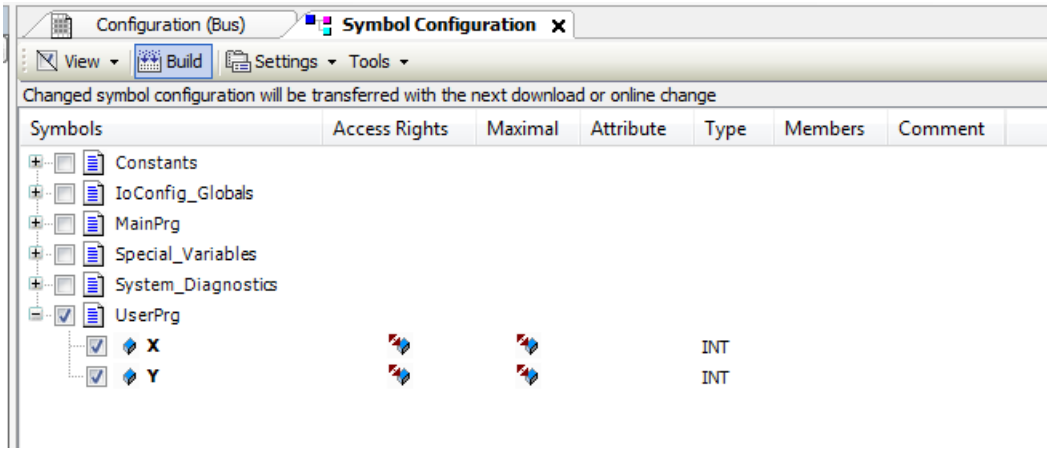



Figure 106: Selecting Variables on the Symbol Configuration

After this procedure, the project must be loaded into a PLC so the variables will be available for communication with the OPC DA Server. If the object Symbol Configuration screen is open and any of the variables, POUs or GVLs selected is changed, its name will appear with the red color. The situations in which this may happen is when a variable is deleted or the attribute value is modified.

It's also possible to set which variables will be available on the OPC DA Server through an attribute inserted directly on the POUs or GVLs where the variables are declared. When the *attribute 'symbol'* is present on the variable declaration, and it may be before the definition of the POU or GVL name, or to each variable individually, these variables are sent directly to the object *Symbol Configuration*, with a symbol in the *Attribute* column. In this case it's necessary, before loading the project into the CPU, to run the command *Build* from within the object *Symbol Configuration*.

The valid syntaxes to use the attribute are:

- *attribute 'symbol' := 'none'* – when the attribute value is *'none'*, the variables won't be available to the OPC DA Server and won't be shown in the object *Symbol Configuration* screen.
- *attribute 'symbol' := 'read'* - when the attribute value is *'read'*, the variables will be available to the OPC DA Server with read only access right.
- *attribute 'symbol' := 'write'* - when the attribute value is *'write'*, the variables will be available to the OPC DA Server with write only access right.
- *attribute 'symbol' := 'readwrite'* – when the attribute value is *'readwrite'*, the variables will be available to the OPC DA Server with read and write access right.

In the following example of variable declaration, the variables A and B settings allow that an OPC DA Server access them with read and write access. However the variable C cannot be accessed, while the variable D can be accessed with read only access rights.

```
{attribute 'symbol' := 'readwrite'}
PROGRAM UserPrg
VAR
A: INT;
B: INT;
{attribute 'symbol' := 'none'}
C: INT;
{attribute 'symbol' := 'read'}
D :INT;
END_VAR
```

When a variable with a type different from the basic types is defined, the use of the attribute must be done inside the declaration of this DUT and not only in the context in which the variable is created. For example, in the case of a DUT instance inside of a POU or a GVL that has an attribute, it will not impact in the behavior of this DUT instance elements. It will be necessary to apply the same access right level on the DUT declaration.

**ATTENTION**

The configurations of the symbols that will be provided to the OPC DA Server are stored inside the PLC project. By modifying these configurations it's necessary to load the application on the PLC so that it's possible to access those variables.

**ATTENTION**

When a variable is removed from the project and loaded on the PLC unchecking it from the object *Symbol Configuration*, the variable can no longer be read with the OPC Client. If the variable is added again to the project, with the same name and same context, and inserted on the object *Symbol Configuration*, it will be necessary to reboot the OPC Client to refresh the variable address reference, which will be created on a different memory area of the PLC.

#### 5.9.7.2. Configuring a PLC on the OPC DA Server

The configuration of the PLC is done inside Mastertool through the option available in the *Online*. It's necessary to run Mastertool as administrator.

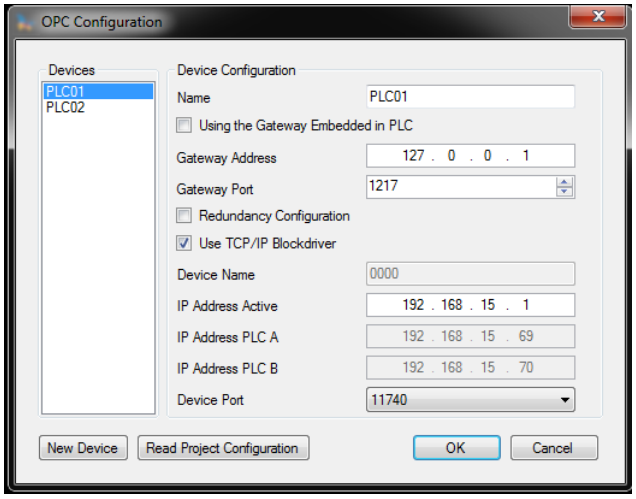



Figure 107: OPC DA Server Settings

The *Gateway Configuration* is the same set in the Gateway used for the communication between Mastertool and the PLC and described in Communication Settings, present in Mastertool's user manual. If the configuration used is *localhost* the *Gateway Address* must be filled with 127.0.0.1. This configuration is necessary because the OPC DA Server uses the same communication gateway and the same protocol used for communication between PLC and Mastertool.

There's the option *Using the Gateway Embedded in PLC* that can be selected when it's desired to use the Gateway that is in PLC itself. This option can be used to optimize the communication, since it prevent excess traffic through a particular station, when more than one station with OPC Client is connected to the same PLC.

To configure the PLC, there are two possible configuration types, depending on the selection of the checkbox *Use TCP/IP Blockdriver*. When the option isn't selected, the field *Device Name* must be filled with the name of the PLC. This is the name displayed by the PLC selected as active in the *Communication Settings* screen.

The other option is to use the *IP Address* of the Ethernet Interfaces. The same address set on the configuration screens must be put in this field. Furthermore, when this method is used, the port number must be set to 11740. The confirmation will save the OPC DA Server configurations.

| Device Configuration | Description | Default Setting | Options |
|---|---|---|---|
| **Name** | PLC description inside the OPC DA Server configuration file. This field can have any name, but for organizational purposes, it's recommended to use the project name that is loaded in the PLC. | 'PLC01' | This field is a STRING and it accepts alphanumeric (letters and numbers) characters and the "_" character. It's not allowed to initiate a STRING with numbers or with "_". It allows up to 49 characters. |
| **Gateway Address** | IP Address of the computer that the OPC DA Server is installed, for the cases in which all PLCs are in the same subnetwork. If there's some PLC that it's in another subnetwork, it must be specified the Gateway used in that subnetwork. | 127.0.0.1 | 0.0.0.0 to 255.255.255.255 |
| **Gateway Port** | TCP Port for the connection with the Gateway. | 1217 | 2 to 65534 |
| **Device Name** | It's the PLC name displayed in the *Communication Settings* of the *Device* tab. The name is the STRING before the hexadecimal value that is between *[ ]*. Enabled only when the checkbox *Use TCP/IP Blockdriver* is not selected. | '0000' | This field is a STRING and it accepts any characters, as done in the PLC name configuration in the Device *Communication Settings* tab. It allows up to 49 characters. |
| **IP Address Active** | IP address of the PLC. Enabled only when the checkbox *Use TCP/IP Blockdriver* is selected. It is used only when the setting is not redundant. | 192.168.15.1 | 0.0.0.0 to 255.255.255.255 |
| **IP Address PLC A** | IP address of the PLC A. Enabled only when the configuration is redundant. It is the primary PLC address to which the server will communicate if there is no failure. | 192.168.15.69 | 0.0.0.0 to 255.255.255.255 |
| **IP Address PLC B** | IP address of the PLC B. Enabled only when the configuration is redundant. It is the secondary PLC address to which the server will communicate if a failure occurs. | 192.168.15.70 | 0.0.0.0 to 255.255.255.255 |
| **Device Port** | TCP Port. Enabled only when the checkbox *Use TCP/IP Blockdriver* is selected. | 11740 | 11740 or 11739 |

Table 103: Configuration Parameter of each PLC for the OPC DA Server

When a new PLC needs to be configured on the OPC DA Server, simply press the *New Device* button and the configuration will be created. When the setup screen is accessed, a list of all PLCs already configured on the OPC DA Server will be displayed. Existing configurations can be edited by selecting the PLC in the *Devices* list and editing the parameters. The PLCs settings that are no longer in use can be deleted. The maximum number of PLCs configured in an OPC DA Server is 16.

If the automation architecture used specifies that the OPC DA Server must be ran on a computer that does not execute communication with the PLC via Mastertool, the tool must be installed on this computer to allow OPC DA Server configuration in the same way as done in other situations.

**ATTENTION**

To store the OPC DA Server configuration, Mastertool must be run with administrator rights on the Operational System. Depending on the OS version, this privilege must be done in the moment that the program is executed: right-click Mastertool's icon and choose *Run as Administrator*.

**ATTENTION**

The settings of a PLC on the OPC DA Server are not stored in the project created in Mastertool. For this reason, it can be performed with an open or closed project. The settings are stored in a configuration file where the OPC DA Server is installed. When changing the settings, it is not required to load the application on the PLC, but depending on the OPC Client it may be necessary to reconnect to the server or load the settings for the data to be updated correctly.

#### *5.9.7.2.1. Importing a Project Configuration*

Using the button *Read Project Configuration*, as shown in Figure 107, you can assign the configuration of the open project to the PLC configuration that is being edited. For this option to work correctly, there must be an open project and an *Active Path* should be set as described in *Communication Settings*, present in Mastertool's user manual. If any of these conditions is not met an error message will be displayed and no data will be modified.

When the above conditions are valid, the PLC settings receive the parameters of the opened project. The *IP Address* and *Gateway Port* information are configured as described in *Communication Settings* according to the *Active Path*. However, the *IP Address* settings are read from NET 1 Ethernet interface settings. The port for connection to the PLC is always assigned in this case as 11740.

### 5.9.7.3. OPC DA Communication Status and Quality Variables

For each PLC created in the OPC DA Server, status variables are generated, named *_CommState* and *_CommStateOK*. The *_CommState* variable indicates the communication between the OPC and the PLC state. This state can interpreted by the OPC Clients according to table below.

| State | Value | Description |
|---|---|---|
| **STATE_TERMINATE** | -1 | If the communication between the OPC DA Server and the OPC Client is terminated, this value will be returned. When there's more than one OPC Client simultaneously connected, this return will occur on the disconnection of the latter connected one. Besides the fact that this state is in the variable, it's value can't be visualized because it only changes when there's no longer a connection with the client. |
| **STATE_PLC_NOT_CONNECTED** | 0 | The PLC configured in the OPC DA Server is not connected. It can happen if the configuration is incorrect (wrong PLC and/or Gateway IP Address) or the PLC is unavailable in that moment. |
| **STATE_PLC_CONNECTED** | 1 | The PLC configured in the OPC DA Server is connected. This is a transitory state during the connection. |

| State | Value | Description |
|---|---|---|
| **STATE_NO_SYMBOLS** | 2 | There are no symbols (variables) available in the PLC configured in the OPC DA Server. It can happen when there are no symbols or there isn't a project loaded on the PLC. |
| **STATE_SYMBOLS_LOADED** | 3 | Finished the process of reading the symbols (variables) from the PLC configured in the OPC DA Server. This is a transitory state during the connection. |
| **STATE_RUNNING** | 4 | After the reading of the symbols (variables) the OPC DA Server is running the periodic update of the values of the available symbols in each configured PLC. |
| **STATE_DISCONNECT** | 5 | There has been a disconnection with the PLC configured in the OPC DA Server. |
| **STATE_NO_CONFIGURATION** | 6 | When the OPC configuration (stored in an OPCServer.ini file) has a wrong syntax, the variable value will be this. Generally, this behavior is not observed for Mastertool maintains this configuration valid. |

Table 104: Description of the Communication states between OPC DA Server and the PLC

The *_CommStateOK* is a variable of the Bool type that indicates if the communication between the OPC DA Server and the PLC is working. When the value is TRUE, it indicates that the communication is working correctly. If the value is FALSE, for some reason it isn't possible to communicate with the PLC.

In addition to monitoring the communication status, the OPC Client can access information on the quality of communication. The quality bits form a byte. They are divided into three groups of bits: *Quality*, *Substatus* and *Limit*. The bits are distributed as follows *QQSSSSLL*, in which *QQ* are the *Quality* bits, *SSSS Substatus* bits and *LL Limit* bits. In this case the *QQ* bits are the most significant in the byte, while the *LL* bits are the least significant.

| QQ | Bits values | Definition | Description |
|---|---|---|---|
| **0** | 00SSSSLL | Bad | The value read can't be used because there's some problem with the connection. It's possible to monitor the value of *_CommState* and diagnose the problem. |
| **1** | 01SSSSLL | Uncertain | The quality can't be defined and may be presented in the Substatus field. |
| **2** | 10SSSSLL | NA | This value is reserved and isn't used by the OPC standard. |
| **3** | 11SSSSLL | Good | The quality is good and the value read can be used. |

Table 105: Description of the OPC Quality value

Table 105 presents the possible quality values. The OPC DA Server only returns *Good* and *Bad* Quality values. A OPC Client can maintain the quality as *Uncertain* due to failures in which it can't establish a connection to the Server. In case of monitoring of the 8 quality bits directly from the OPC DA Server, the *Substatus* and *Limit* fields shall be null and the *Good* Quality will be presented as the value 192 and the *Bad* Quality will be value 0.

#### 5.9.7.4. Limits of Communication with OPC DA Server

The table below presents the OPC DA Server configuration limits.

| | |
|---|---|
| **Maximum number of variables communicating with a single PLC** | See note |
| **Maximum number of PLCs in an OPC DA Server** | 16 |
| **Maximum number of simultaneous connections of an OPC DA Server in a single PLC** | 8 |

Table 106: OPC DA Server Communication Limits

**Note:**

**Maximum number of variables communicating with a single PLC:** This limit is defined by "Communication Tags" informed on General Features table.

**ATTENTION**

The Maximum number of simultaneous connections of an OPC DA Server in a single PLC is shared with connections made with Mastertool. I.e. the sum of connections of OPC DA Server and Mastertool should not exceed the maximum quantity defined in Table 106.

The communication between the OPC DA Server and the PLC uses the same protocol used in Mastertool communication with the PLC. This protocol is only available for the Ethernet interfaces of the Nexto Series CPUs, it's not possible to establish this kind of communication with the Ethernet expansion modules.

When a communication between the OPC DA Server and the PLC is established, these two elements start a series of transactions aimed at solving the addresses of each declared variables, optimizing the communication in data reading regime. Besides, it's also resolved in this stage the communication groups used by some Clients in order to optimize the communication. This initial process demands some time and depends on the quantity of mapped variables and the processing capacity of the device.

#### 5.9.7.5. Accessing Data Through an OPC DA Client

After the configuration of the OPC DA Server, the available data on all PLCs can be accessed via an OPC Client. In the configuration of the OPC Client, the name of the OPC DA Server must be selected. In this case the name is *CoDeSys.OPC.DA*. The figure below shows the server selection on the client driver of the BluePlant SCADA software.

**ATTENTION**

The same way that in Mastertool, some tools must be executed with administrator privileges in the Operational System for the correct functioning of the OPC Client. Depending on the OS version, this privilege must be activated in the moment that the program is executed. To do this, right-click Mastertool's icon and choose *Run as Administrator*.

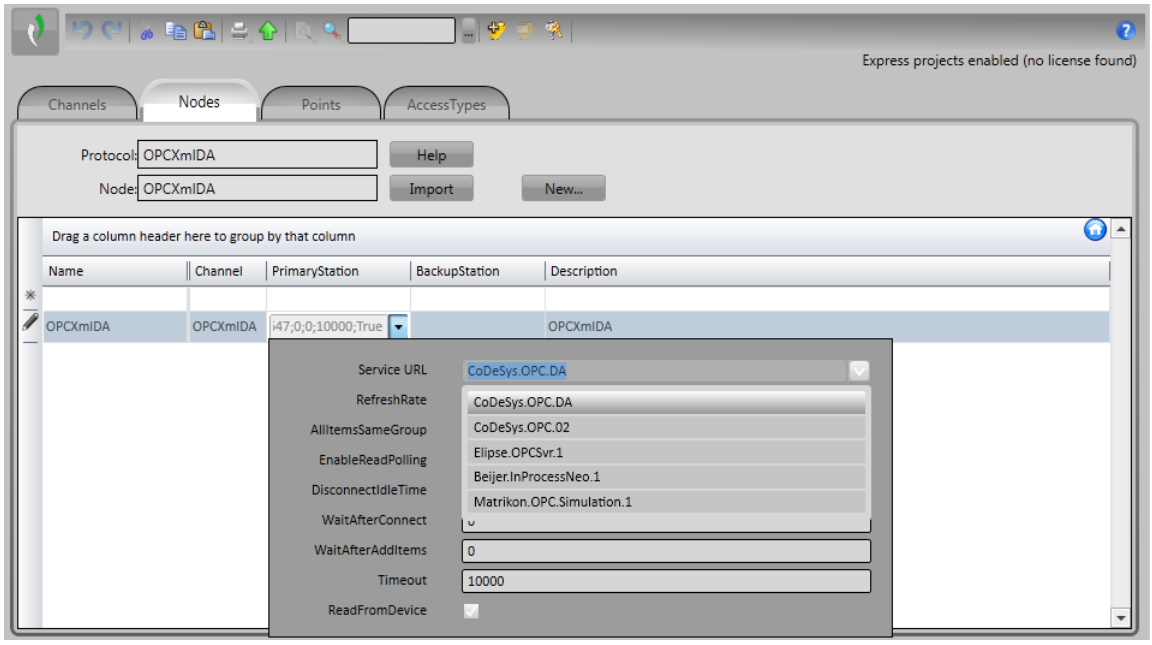



Figure 108: Selecting the OPC DA Server in the Client Configuration

In cases where the server is remotely located, it may be necessary to add the network path or IP address of the computer in which the server is installed. In these cases, there are two configuration options. The first is to directly configure it, being necessary to enable the COM/DCOM Windows Service. However, a simpler way is to use a tunneller tool that abstracts the COM/DCOM settings, and enable a more secure communication between the Client and the Server. For more information on this type of tool, refer to the *NAP151 - Tunneller OPC*.

Once the Client connects with the Server, it's possible to use the TAGs import commands. These commands consult the information declared in the PLC, returning a list with all the symbols available in it.

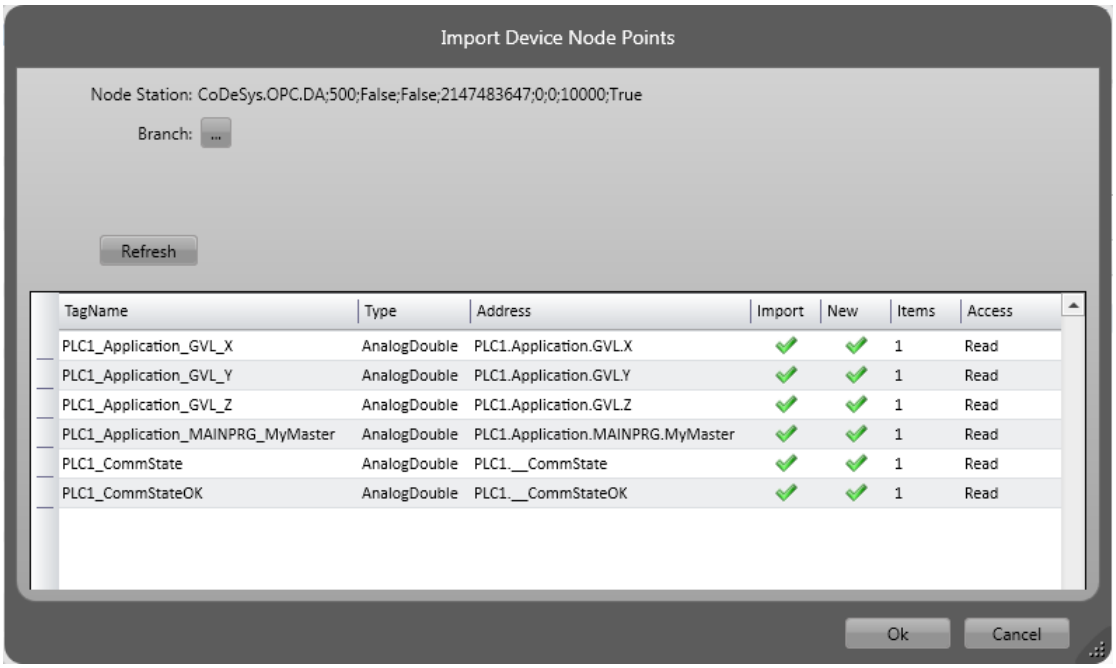



Figure 109: Symbols list consulted by the OPC Client

The list of selected variables will be included in the Client communication list and can be used, for example, in a SCADA system screen.

**ATTENTION**

The simulation mode of Mastertool can be used for OPC communication tests. The information on how to configure it are presented in the *Testing an OPC Communication using the Simulator* section of the Mastertool's user manual.

### 5.9.8. OPC UA Server

The OPC UA protocol is an evolution of the OPC family. Independent of platform, it is designed to be the new standard used in industrial communications.

Based on the client/server architecture, the OPC UA protocol offers numerous advantages in the development of design and facilities in communication with the automation systems.

When it comes to project development, configuring communication and exchanging information between systems is extremely simple using OPC UA technology. Using other address-based drivers, it is necessary to create tables to relate the supervision system tags and programmable controller variables. When data areas change during project development, it is necessary to redo the mappings and new tables with the relationships between the PLC information and the SCADA system.

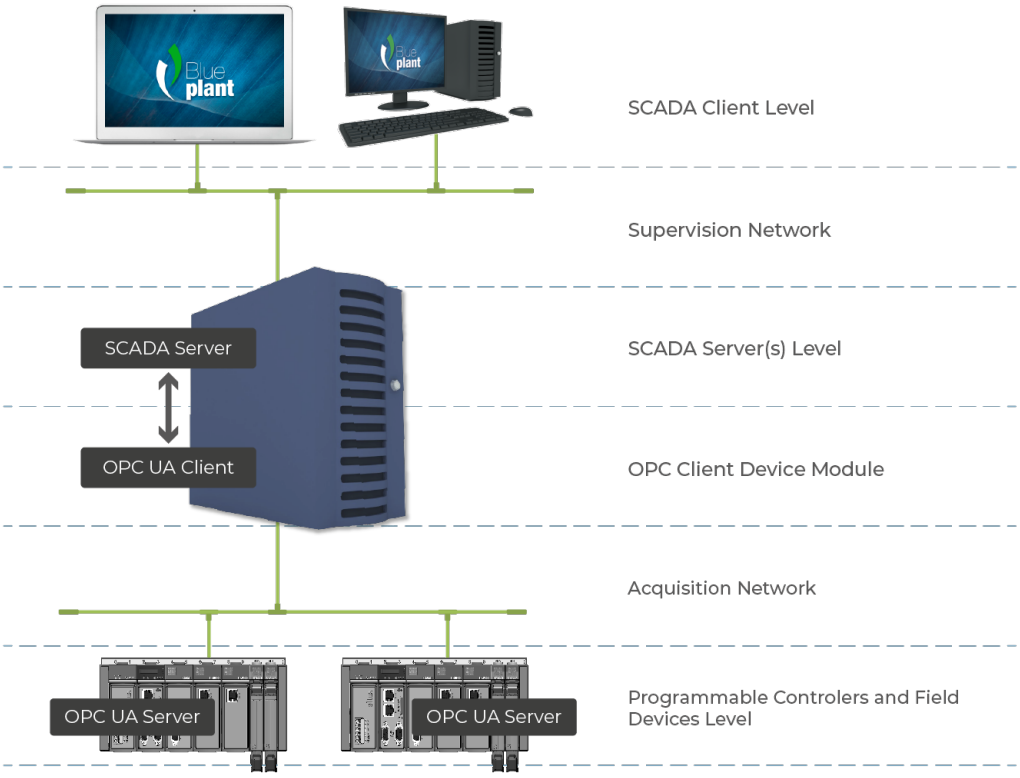



Figure 110: OPC UA Architecture

The figure above presents a typical architecture for SCADA system communication and PLCs in automation design. All roles present in the communication are explicit in this figure regardless of where they are running, they may be on the same equipment or on different equipment. Each of the roles of this architecture is described in table below.

| Role | Description |
|---|---|
| **Programmable Controllers and Field Devices Level** | The field devices and the PLCs are where the operation state and plant control information are stored. The SCADA system access the information on these devices and store on the SCADA server, so that the SCADA clients can consult it during the plant operation. |
| **OPC UA Server Modules** | The OPC UA Server is an internal module of the PLCs responsible for receiving the OPC UA requests and translating them for communication with the field devices. |
| **Acquisition Network** | The acquisition network is the network in which OPC UA messages travel to request the data that is collected from the PLCs and field devices. |
| **OPC Client Device Module** | The OPC UA Client module, which is part of the SCADA Server, is responsible for making requests to the OPC UA Servers using the OPC UA protocol. The data collected by it is stored in the SCADA Server database. |
| **SCADA Server Level** | The SCADA Server is responsible for connecting to the various communication devices and store the data collected by them on a database, so that it can be consulted by the SCADA Clients. |
| **Supervision Network** | The supervisory network is the network by which SCADA Clients are connected to SCADA Servers, often using a proprietary SCADA system protocol. In a topology in which multiple Clients are not used or the Server and Client are installed in the same equipment, there is no such network, and in this case this equipment must directly use the OPC UA protocol for communication with the PLC. |
| **SCADA Client Level** | The SCADA Clients are responsible for requesting to the SCADA Servers the necessary data to be shown in a screen where the operation of a plant is being executed. Through then it is possible to execute readings and writings on data stored on the SCADA Server database. |

Table 107: Roles Description on an OPC UA Server Architecture

When using the OPC UA protocol, the relationship between the tags of the supervisory systems and the process data in the controller variables is completely transparent. This means that if data areas change during project development, there is no need to re-establish relationships between PLC information and SCADA. Simply use the new variable provided by the PLC in the systems that request this data.

The use of OPC UA offers greater productivity and connectivity with SCADA systems. It contributes to reduced application development time and maintenance costs. It also enables the insertion of new data in the communication in a simplified way with greater flexibility and interoperability among the automation systems as it is an open standard.

It is worth noting that the OPC UA is only available on the integrated Ethernet interfaces of the Nexto CPUs. Ethernet expansion modules do not support this functionality.

#### 5.9.8.1. Creating a Project for OPC UA Communication

The steps for creating a project with OPC UA are very similar to the steps described in the section Creating a Project for OPC DA Communication. As with the OPC DA protocol, the configuration of the OPC UA protocol is based on the configuration of the *Symbol Configuration*. To enable the OPC UA, simply enable the *Support OPC UA Features* option in the configuration, as shown in figure below.

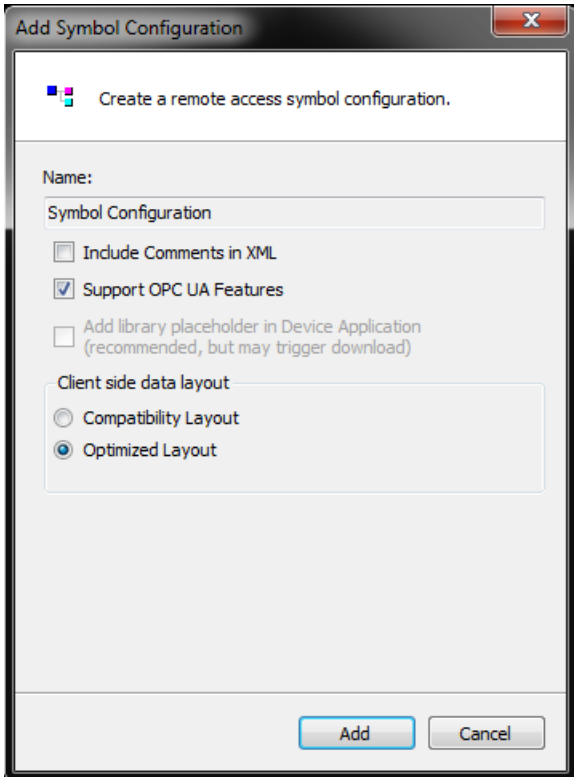



Figure 111: Symbol Configuration Object

**ATTENTION**

When enabling OPC UA protocol support, OPC DA protocol support is still enabled. You can enable OPC UA and OPC DA communications at the same time to report the variables configured on the *Symbol Configuration* object or via attributes.

Another way to access this configuration, once already created a project with the *Symbol Configuration* object, is given by accessing the *Settings* menu of the configuration tab of the *Symbol Configuration*. Simply select the option *Support OPC UA features* to enable support for the OPC UA protocol, as shown in figure below.

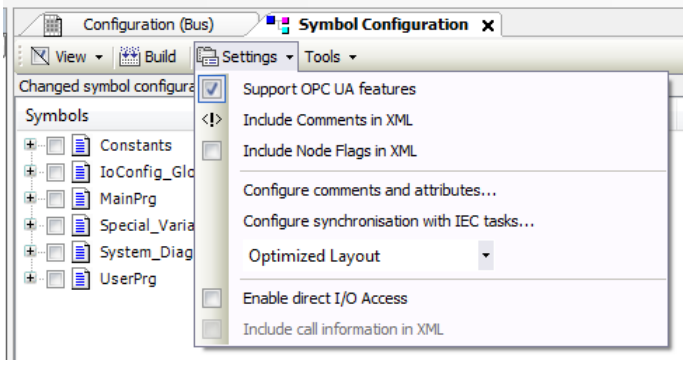



Figure 112: Enabling OPC UA in Object Symbol Configuration

After this procedure the project can be loaded into a PLC and the selected variables will be available for communication with the OPC UA Server.

#### 5.9.8.2. Types of Supported Variables

This section defines the types of variables that support communication via the OPC UA protocol, when declared within GVLs or POUs and selected in the *Symbol Configuration* object (see previous section).

The following types of simple variables are supported:

- BOOL
- SINT
- USINT / BYTE
- INT
- UINT / WORD
- DINT
- UDINT / DWORD
- LINT
- ULINT / LWORD
- REAL
- LREAL
- STRING
- TIME
- LTIME

You can also use structured types (STRUCTs or Function Blocks) created from previous simple types.

Finally, it is also possible to create arrays of simple types or of structured types.

#### 5.9.8.3. Limit Connected Clients on the OPC UA Server

The maximum number of OPC UA clients connected simultaneously in a PLC is 8 (eight).

#### 5.9.8.4. Limit of Communication Variables on the OPC UA Server

There is no configuration limit. The maximum possible number of variables depends on the processing capacity of the device.

When a communication is established between the OPC UA Server and the PLC, these two elements initiate a series of transactions that aim to solve the address of each declared variable, optimizing the communication in regime of reading of data. In addition, at this stage, the classifications of the communication groups used by some Clients are also resolved in order to optimize communication. This initial process takes some time and depends on the amount of variables mapped and the processing capacity of the device.

#### 5.9.8.5. Encryption Settings

If desired, the user can configure encryption for OPC UA communication using the *Basic256SHA256* profile, for a secure connection (cyber security).

To configure encryption on an OPC UA server, you must create a certificate for it using the following steps in the Mastertool programmer:

1. Define an active path for communication with the controller (no login required);
2. From the *View* menu, select *Security Screen*;
3. Click the *Devices* tab on the left side of this screen;
4. Click the icon  to perform a refresh;
5. Click on the *Device* icon, below which will open several certificates (*Own Certificates, Trusted Certificates, Untrusted Certificates, Quarantined Certificates*);
6. Click the icon  to generate a certificate and select the following parameters:
   - *Key length* (bit): 3072
   - *Validity period* (days): 365 (can be modified if desired)
7. Wait while the certificate is calculated and transferred to the controller (this may take a few minutes);
8. Reboot the controller.
9. On the OPC UA client, perform the necessary procedures to connect to the OPC UA server and generate a certificate with the *Basic256Sha256* profile (see specific OPC UA client manual for details);

10. Back to Mastertool, click on the icon of the *Security Screen* to perform a refresh;
11. On the *Security Screen*, select the "*Quarantined Certificates*" folder under the *Device*. In the right panel you should observe a certificate requested by the OPC UA client;
12. Drag this certificate to the folder "*Trusted Certificates*";
13. Proceed with the settings in the OPC UA client (see specific OPC UA client manual for details).

To remove encryption previously configured on a controller, you must do the following:

1. Define an active path for communication with the controller (no login required);
2. From menu *View*, select *Security Screen*;
3. Click on the *Devices* on the left side of this screen;
4. Click the icon to perform a refresh;
5. Click on the *Device* icon, below which will open several certificates (*Own Certificates, Trusted Certificates, Untrusted Certificates, Quarantined Certificates*);
6. Click the folder "*Own Certificates*" and in the right panel select the certificate (OPC UA Server);
7. Click the icon to remove this project and driver certificate;
8. Reset (turn off and on) the controller.

#### 5.9.8.6. Main Communication Parameters Adjusted in an OPC UA Client

Some OPC UA communication parameters are configured on the OPC UA client, and negotiated with the OPC UA server at the time the connection between both is established. The following subsections describe the main OPC UA communication parameters, their meaning, and care to select appropriate values for them.

In an OPC UA client it is possible to group the variables of a server into different *subscriptions*. Each *subscription* is a set of variables that are reported in a single communication packet (*PublishResponse*) sent from the server to the client. The selection of the variables that will compose each subscription is made in the OPC UA client.

**ATTENTION**

Grouping variables into multiple *subscriptions* is interesting for optimizing the processing capacity and consumption of Ethernet communication bandwidth. Such aspects of optimization are analyzed in greater depth in the OPC UA Server user manual MU214609, where some rules for the composition of *subscriptions* are suggested. This user manual also discusses in more depth several concepts about the OPC UA protocol.

Some of the communication parameters described below must be defined for the server as a whole, others for each *subscription*, and others for each variable that makes up a *subscription*.

##### *5.9.8.6.1. Endpoint URL*

This parameter defines the IP address and TCP port of the server, for example:

*opc.tcp://192.168.17.2:4840*

In this example, the IP address of the controller is 192.168.17.2.

The TCP port should always be 4840.

##### *5.9.8.6.2. Publishing Interval (ms) e Sampling Interval (ms)*

The *Publishing Interval* parameter (unit: milliseconds) must be set for each *subscription*.

The *Sampling Interval* parameter must be set for each variable (unit: milliseconds). However, in many OPC UA clients, the *Sampling Interval* parameter can be defined for a *subscription*, being the same for all the variables grouped in the *subscription*.

Only the variables of a *subscription* whose values have been modified are reported to the client through a *Publish Response* communication packet. The *Publishing Interval* parameter defines the minimum interval between consecutive *Publish Response* packets of the same *subscription*, in order to limit the consumption of processing and Ethernet communication bandwidth.

To find out which subscription variables have changed and are to be reported to the client in the next *Publish Response* packet, the server must perform comparisons, and such (*samplings*) are performed by the same with the *Sampling Interval*. It is recommended that the value of *Sampling Interval* varies between 50% and 100% of the value of the *Publishing Interval*, because there is a relatively high processing consumption associated with the comparison process executed in each *Sampling Interval*.

It can be said that the sum between *Publishing Interval* and *Sampling Interval* is the maximum delay between changing a value on the server and the transmission of the *Publish Response* packet that reports this change. Half of this sum is the average delay between changing a value on the server and the transmission of the *Publish Response* packet that reports this change.

#### 5.9.8.6.3. Lifetime Count e Keep-Alive Count

These two parameters must be configured for each *subscription*.

The purpose of these two parameters is to create a mechanism for deactivating a *subscription* on the initiative of the server, in case it does not receive customer's *PublishRequest* communication packets for this *subscription* for a long time. *PublishRequest* packets must be received by the server so that it can broadcast *Publish Response* packets containing the subscription variables that have changed their values.

If the server does not receive *PublishRequest* packets for a time greater than *Lifetime Count* multiplied by *Publishing Interval*, the server deactivates the *subscription*, which must be re-created by the client in the future if desired.

In situations where the variables of a *subscription* do not change, it could be a long time without the transmission of *PublishResponses* and consequently *PublishRequests* that succeed, causing an undesired deactivation of the *subscription*. To prevent this from happening, the *Keep-Alive Count* parameter was created. If there are no *subscription* data changes for a time equal to *Keep-Alive Count* multiplied by *Publishing Interval*, the server will send a small empty *Publish Response* packet indicating that no variable has changed. This empty *Publish Response* will authorize the client to immediately send the next *PublishRequest*.

The *Keep-Alive Count* value must be less than the *Lifetime Count* value to prevent unwanted deactivation of the *subscription*. It is suggested that *LifeTime Count* be at least 3 times larger than *Keep-Alive Count*.

#### 5.9.8.6.4. Queue Size e Discard Oldest

These parameters must be maintained with the following fixed values, which are usually the default values on the clients:

- Queue Size: 1
- Discard Oldest: enable

According to the OPC UA standard, it is possible to define these parameters for each variable. However, many clients allow you to define common values for all variables configured in a *subscription*.

*Queue Size* must be retained with value 1 because there is no event support in this implementation of the OPC UA server, so it is unnecessary to define a queue. Increasing the value of *Queue Size* may imply increase communication bandwidth and CPU processing, and this should be avoided.

*Discard Oldest* must be maintained with the *enable* value, so that the *Publish Response* package always reports the most recent change of value detected for each variable.

#### 5.9.8.6.5. Filter Type e Deadband Type

These parameters must be maintained with the following fixed values, which are usually the default values in the clients:

- *Filter Type: DataChangeFilter*
- *Deadband Type: none*

According to the OPC UA standard, it is possible to define these parameters for each variable. However, many clients allow you to define common values for all variables configured in a *subscription*.

The *Filter Type* parameter must be of *DataChangeFilter*, indicating that value changes in the variables should cause it to be transmitted in a *Publish Response* package.

*Deadband Type* should be kept in *"none"* because there is no implementation of *deadbands* for analog variables. In this way, any change of the analog variable, however minimal, causes its transmission in a *Publish Response* package.

To reduce processing power and Ethernet communication bandwidth, you can deploy *deadbands* on your own as follows:

- Do not include the analog variable in a *subscription*;
- Instead, include in a *subscription* an auxiliary variable linked to the analog variable;
- Copy the analog variable to the auxiliary variable only when the user-managed *deadband* is extrapolated.

#### 5.9.8.6.6. PublishingEnabled, MaxNotificationsPerPublish e Priority

It is suggested that the following parameters be maintained with the following values, which are usually the default values in the clients:

- *PublishingEnabled: true*
- *MaxNotificationsPerPublish: 0*
- *Priority: 0*

These parameters must be configured for each *subscription*.

*PublishingEnable* must be *"true"* so that the *subscription* variables are reported in case of change of value.

*MaxNotificationsPerPublish* indicates how many of the variables that have changed value can be included in the same *Publish Response* package. The special value *"0"* indicates that there is no limit to this, and it is recommended to use this value so that all changed variables are reported in the same *Publish Response* package.

*Priority* indicates the relative priority of this *subscription* over others. If at any given moment the server should send multiple *Publish Response* packages of different *subscriptions*, it will prioritize the one with the highest value of priority. If all *subscriptions* have the same priority, *Publish Response* packets will be transmitted in a fixed sequence.

#### 5.9.8.7. Accessing Data Through an OPC UA Client

After configuration of the OPC UA Server the data available in all PLCs can be accessed via a Client OPC UA. In the configuration of the OPC UA Client, the address of the correct OPC UA Server must be selected. In this case the address *opc.tcp://ip-address-of-device:4840*. The figure below shows the server selection in the SCADA BluePlant client software driver.

**ATTENTION**

Some tools need to be run with administrator rights, like Mastertool, on the Operating System for the correct operation of the OPC UA Client. Depending on the version of the Operating System this right must be authorized when running the program. For this operation right click on the tool executable and choose the option *Run as administrator*.

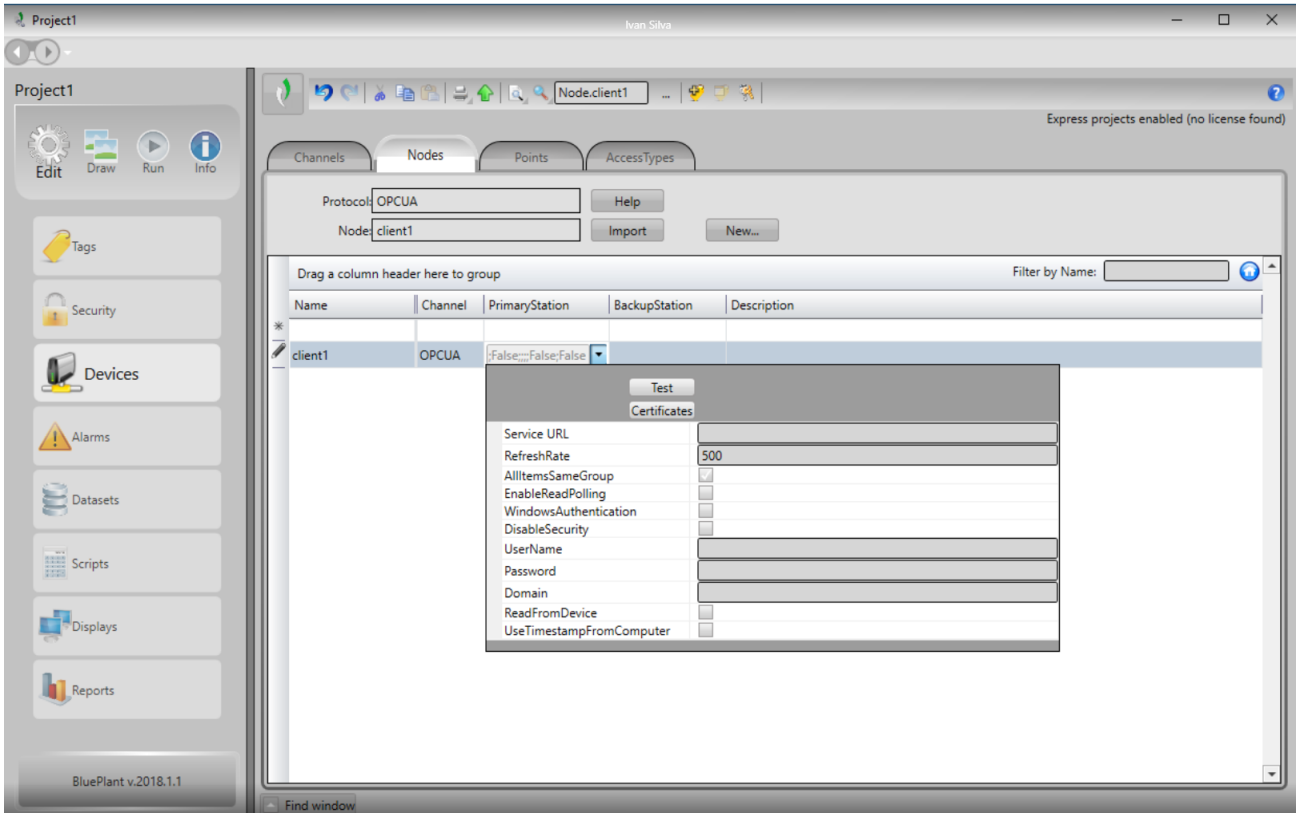



Figure 113: Selecting OPC UA Server in Client Configuration

Once the Client connects to the Server, TAG import commands can be used. These commands query information declared in the PLC, returning a list with all the symbols made available by the PLC.

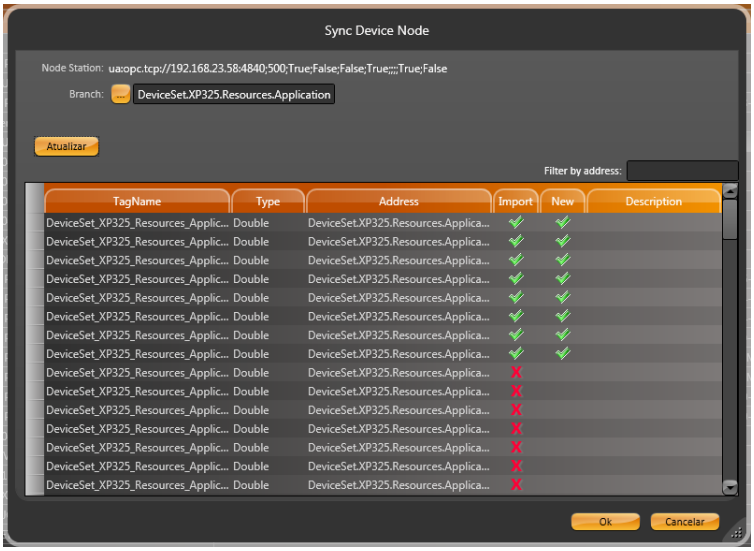



Figure 114: List of Symbols Browsed by OPC UA

The list of selected variables will be included in the Client's communications list and can be used, for example, in screens of a SCADA system.

### 5.9.9. EtherCAT Master

EtherCAT (*Ethernet Control Automation Technology*) is a master-slave architecture protocol with high performance, for deterministic Ethernet, that allows real time performance as it updates 1000 distributed I/O in 30 $\mu$S or 100 servomotors axis each 100 $\mu$S using twisted pair cables or optic fiber. Besides, it supports flexible topology, allowing for line, tree and/or star connections.

An Ethernet frame can be processed in real time instead of being received, interpreted and copied as process data in each connection. The FMMU (*Fieldbus Memory Management Unit*) in each Slave node reads the data that are addressed to it at the same time that the telegram is forwarded to the next device. In a similar way, the input data are inserted as the telegram is passed. Because of this, the frames are delayed just a few nanoseconds. Access on the Ethernet terminals can be made in any order as the data sequence is independent of the physical order. It can perform Broadcast, Multicast and between slaves communications.

The EtherCAT protocol allows a precise synchronization, that is required, for example, in applications where several axis simultaneously perform coordinated movements, it can be done through an exact adjust of the *Distributed Clock*. There's also the possibility to configure devices that, as opposed to synchronous communication, have an elevated tolerance degree inside the communication system.

The configuration of EtherCAT modules is initially determined by the *Device Description Files* of the Master and Slave devices used, and can be modified by the user in the *Configuration Editor* dialog boxes. However, for conventional applications and with the desire of an as easy as possible manipulation, large-scale configurations can be automated by choosing the *Autoconfig* mode in EtherCAT Master - General.

Note the possibility of modifying the Master and Slave configuration parameters also in operational mode, through the Master and Slave instances, according to the availability of the device in question.

#### 5.9.9.1. Installing and inserting EtherCAT Devices

In order to be able to insert and configure EtherCAT devices as objects in the device tree, the Slave devices must be installed.

The Master device is automatically installed by the default MasterTool installation. The EtherCAT Master defines which Slaves can be inserted.

To install the Slave devices the *Device Repository* must be opened, use the *EtherCAT XML Device description Configuration File (*.xml)* filter and select the device description files (*EtherCAT XML Device Description / ESI EtherCAT Slave Information*), supplied with the hardware. The Slave descriptions are available as XML files (file type: *.xml).

An EtherCAT Master can be added to the *Devices Tree* through the *Add Device* command, through the context menu of the CPU NET connectors.

Under a master, one or more slaves can be added, selecting an EtherCAT Master and running the *Add Device* command (context menu of the EtherCAT Master) or running the *Scan For Devices* command.

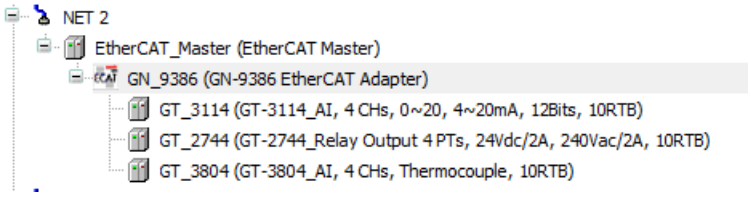



Figure 115: EtherCAT Configuration Example

**ATTENTION**

- Only one EtherCAT Master instance per project is allowed.
- Only available on the NET connectors of the PLC.
- It cannot be used when the NETs are set as redundant.
- It cannot be used when Project has cluster redundancy.
- Other drivers cannot be instanced in the same NET port as the EtherCAT Master.

#### *5.9.9.1.1. EtherCAT - Scan Devices*

The *Scan For Devices* command, available in the EtherCAT Master context menu, runs a search for the Slave devices physically installed in the EtherCAT network of the PLC currently connected. This means that with this command it's possible to detect and visualize the hardware components in the window presented in the figure below, allowing the user to map them directly in the project *Device Tree* do projeto.

It's noteworthy that, when the *Scan For Devices* command is selected, a connection with the PLC will be automatically established before the search begins and terminated when the search ends. So, for the first execution of this command, the Gateway connection must be configured and a program with the EtherCAT Master configured must be loaded into the PLC.

When the command is executed, the *Scanned Devices* field will contain a list of all devices and modules found during the last scan. To add them to the project, just click on the button *Copy All Devices To Project*. It's also possible to perform a comparison of the devices found in the search with the ones in the project by selecting the box *Show differences to project*.

If you add an EtherCAT Master module to the Project and use the *Scan For Devices* command, you will have a list of all the available EtherCAT Slaves. Entries in bold will be shown, if there's more than one device with the same description. With a double click on the entrance a list will open, and so the desired device can be selected.

After completing the changes in the EtherCAT network configuration, it's necessary to do a new project download, for the changes to take effect.

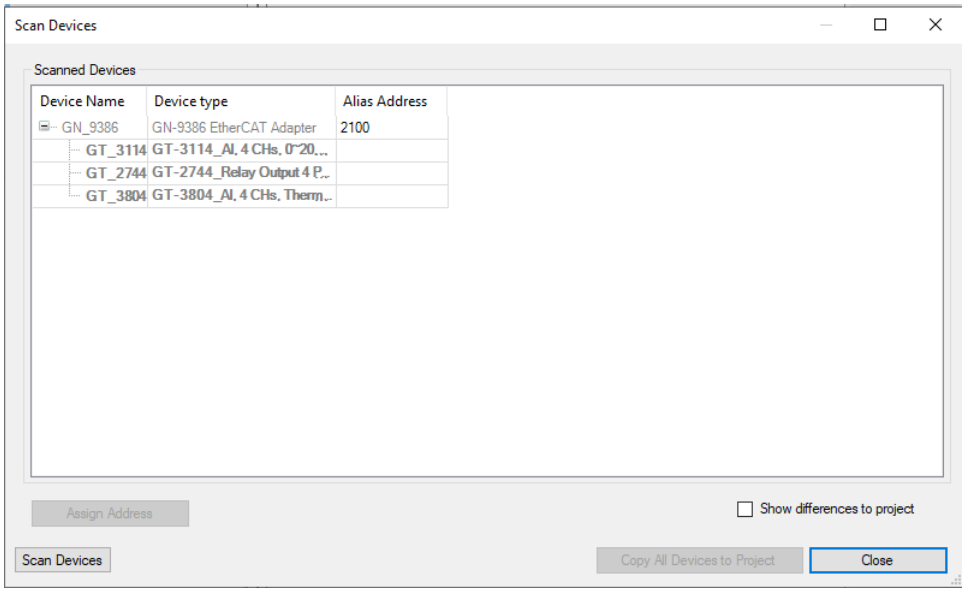



Figure 116: EtherCAT Devices Search Dialog

#### 5.9.9.2. EtherCAT Master Settings

Below are listed the options to carry out the EtherCAT Master configuration, such as defined in *Device Description File*.

##### *5.9.9.2.1. EtherCAT Master - General*

Below are the general parameters found in the initial screen of the EtherCAT Master configuration, according figure below.

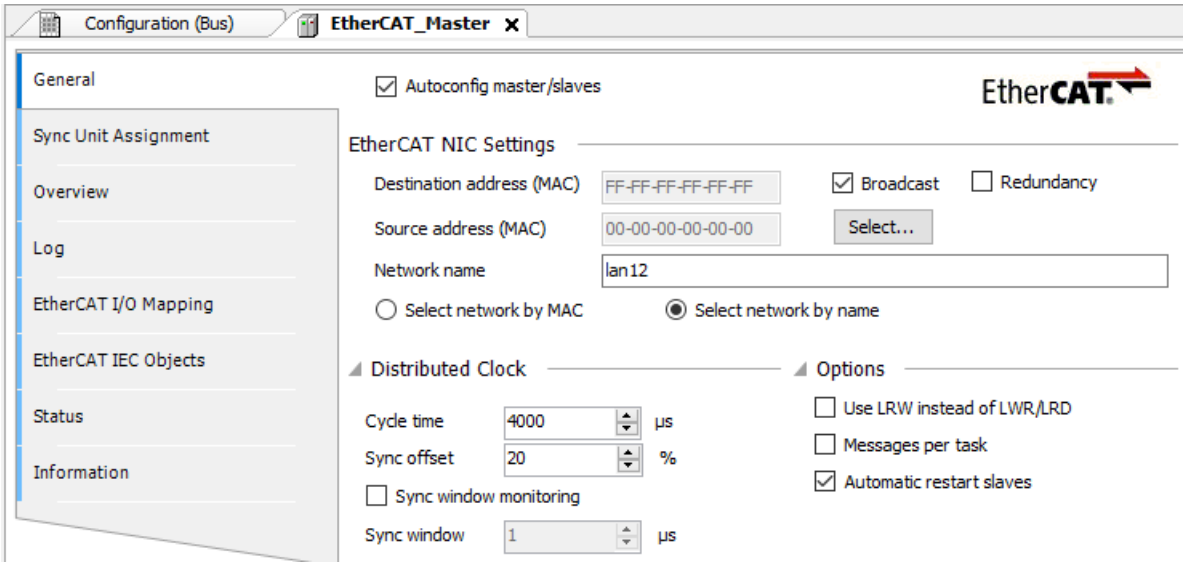



Figure 117: EtherCAT Master Configuration Dialog

| Device Configuration | Description | Factory Default | Possible Values |
|---|---|---|---|
| **Autoconfig master/slaves** | Enable the Master and Slave automatic configuration. | Marked | Marked<br>Unmarked |
| **Cycle time [$\mu$s]** | Sets the time period in which a new data telegram must be send to the bus. | 4000 | 2000 to 1000000 |
| **Sync Offset [%]** | Adjust the offset, from the PLC cycle, of the EtherCAT Slave synchronization interrupt. | 20 | -50 to 50 |
| **Sync window monitoring** | If enabled, this option allows monitoring the Slave synchronization. | Unmarked | Marked<br>Unmarked |
| **Sync window [$\mu$s]** | Time for the Sync Window Monitoring. | 1 | 1 to 32768 |
| **Use LRW instead of LWR/LRD** | Enabling of the combined read and write commands. | Unmarked | Marked<br>Unmarked |
| **Messages per task** | If enabled, the read and write commands that are dealing with input and output messages can be done in different tasks. | Unmarked | Marked<br>Unmarked |

| Device Configuration | Description | Factory Default | Possible Values |
|---|---|---|---|
| **Automatic restart slaves** | Restart the devices when the communication is aborted. | Marked | Marked<br>Unmarked |

Table 108: EtherCAT Master Configuration

**Notes:**

**Autoconfig master/slaves:** If this option is enabled, most of Master and Slave configuration will be made automatically, based on the description files and implicit calculations. In this case, the FMMU / Sync dialog will not be available. If it's unchecked the *Image In Address* and *Image Out Address* options will be available to the user.

**ATTENTION**

The *Autoconfig* mode is enabled by default and usually enough and highly recommended for standard applications. If it's disabled, all configuration definitions will have to be made manually, and thus, some specialized knowledge is required. To configure a Slave-to-Slave communication, the *Autoconfig* option must be disabled.

**Cycle time:** Time period after which, a new data telegram must be sent to the bus. If *Distributed Clock* functionality is enabled, the value of this parameter will be transferred to the Slaves clocks. This way, a precise data exchange synchronization can be achieved, which is especially important in cases where the distributed process demands simultaneous actions. So, a very precise time base, with a jitter significantly smaller than a microsecond, for all the network can be achieved.

**Sync Offset:** This value allows the adjustment of the offset of the EtherCAT Slave synchronization interrupt to the PLC cycle. Normally, the PLC task cycle begins 20% later than the Slaves synchronization interruption. This means that the PLC task can be delayed by 80% of the cycle time and no message will be lost.

**Sync Window:** If the synchronization of all Slaves are inside this time window, the EtherCAT Master *bDistributedClockInSync* diagnostic will be set to TRUE, otherwise it will be set to FALSE. When Distributed Clock is used, it's highly recommended to use a dedicated task with high priority as the *Bus cycle task* of the EtherCAT Master. To do this, it's necessary to use Project Profiles that allows the creation of new tasks, then create a cyclic task with priority 0 (real time task) and link it to the master *Bus cycle task* on the EtherCAT Master - I/O Mapping tab of the EtherCAT Master. The user can also change the value of the *wDCInSyncWindow* variable, configuring the maximum jitter allowed on the synchronization between master and slaves.

**Use LRW instead of LWR/LRD:** Activating this option enables the Slave-to-Slave communication because, instead of using separated reading (LRD) and write (LWR) commands, combined reading/writing (LRW) commands will be used.

**Automatic Restart Slaves:** By enabling this option, the Master will restart the Slaves as soon as the communication is aborted.

#### *5.9.9.2.2. EtherCAT Master - Sync Unit Assignment*

This tab of the EtherCAT Master configuration editor shows all slaves that are entered below a specific master with an assignment to the sync units.

With EtherCAT sync units, multiple slaves are configured into groups and subdivided into smaller units. For each group, the job counter can be monitored for better and more accurate error detection. As soon as a slave is missing from a group of synchronization units, the other slaves in the group are also shown as missing. Detection occurs immediately on the next bus cycle because the job counter is checked continuously. With device diagnostics, the missing group can be remedied as quickly as possible.

Unaffected groups remain operable without any interference.

#### *5.9.9.2.3. EtherCAT Master - Overview*

This tab of the EtherCAT Master configuration editor provides an overview of the states of all slaves, which are entered below this master and have an address. Modules are not displayed.

#### *5.9.9.2.4. EtherCAT Master - I/O Mapping*

This EtherCAT Master configuration editor tab offers the possibility to change the task that will be used for bus updates.

#### 5.9.9.2.5. *EtherCAT Master - IEC Objects*

This tab of the EtherCAT Master configuration editor lists *objects* that allow access to the device from the IEC application. In online mode, this is used for monitoring.

#### 5.9.9.2.6. *EtherCAT Master - Status / Information Tabs*

The Status tab of the EtherCAT Master configuration editor provides status information (e.g. '*Running*', '*Stopped*') and diagnostic messages specific of the device and the internal bus system.

The Information tab, present on the EtherCAT Master configuration editor, shows, if available, the following general information about the module: *Name, Vendor, Type, Version Number, Category, Order Number, Description, Image*.

### 5.9.9.3. EtherCAT Slave Configuration

Below are listed the main EtherCAT Slave configuration options, as defined in the *Device Description File*.

#### 5.9.9.3.1. *EtherCAT Slave - General*

Below are presented the general parameters found in EtherCAT Slave configuration initial screen. This field is only available if the *Autoconfig* mode (Master) isn't enabled.

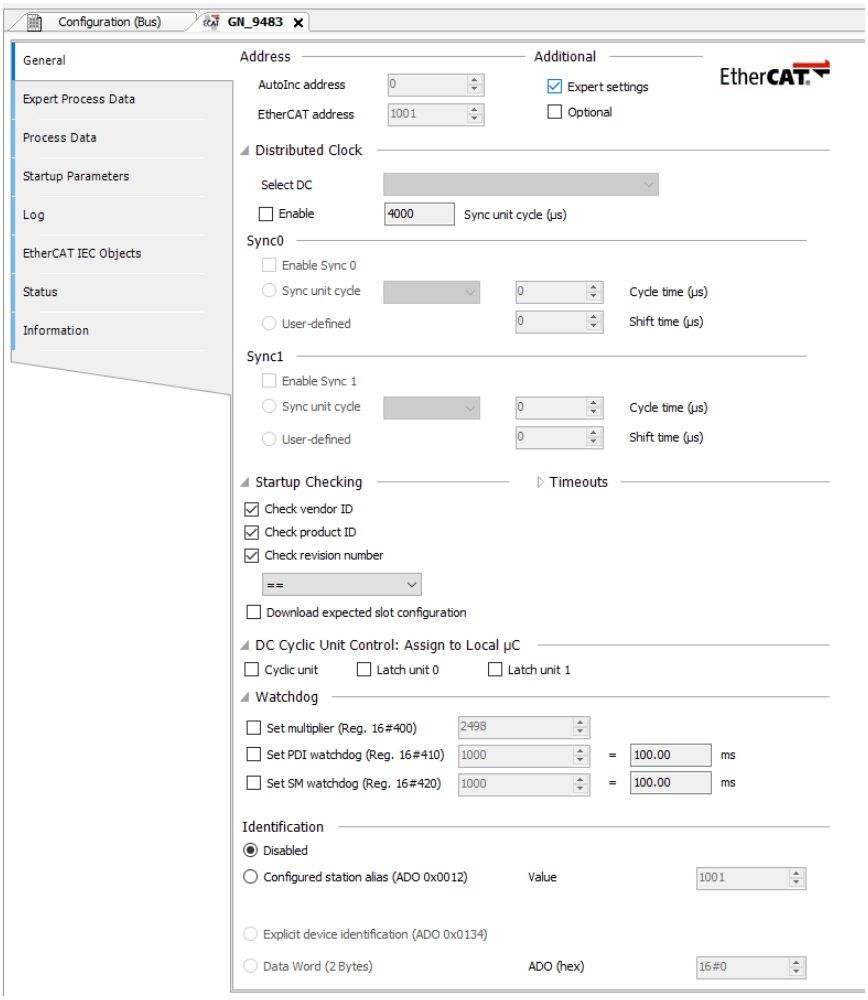



Figure 118: EtherCAT Slave Configuration Dialog

| Device Configuration | Description | Default Value | Options |
|---|---|---|---|
| **AutoInc Address** | Auto incremental Address (16-bit) defined by the Slave position in the network. | - | -65535 to 0 |
| **EtherCAT Address** | Slave final address, assign by the Master during startup. This address is independent from the position in the network. | - | 1 to 65535 |
| **Expert settings** | Enable the Slave advanced Settings options. | Unmarked | Marked<br>Unmarked |
| **Optional** | Declare the Slave as Optional. | Unmarked | Marked<br>Unmarked |
| **Select DC** | Show all Distributed Clock configurations provided by the device description file. | - | - |
| **Enable Distributed Clock** | Enable the Distributed Clock configuration options. | Unmarked | Marked<br>Unmarked |
| **Sync Unit Cycle [$\mu$s]** | Show the Cycle Time set in Master. | 100000 | 2000 to 1000000 |
| **Enable (Sync 0)** | Enable the Sync 0 synchronization unit configurations. | Unmarked | Marked<br>Unmarked |
| **Sync Unit Cycle (Sync 0)** | By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle. | Unmarked | Marked<br>Unmarked |
| **User Defined (Sync 0)** | If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time ($\mu$s) field. | Unmarked | Marked<br>Unmarked |
| **Cycle Time [$\mu$s] (Sync 0)** | Show the cycle time currently set. | 100000 | 1 to 2147483647 |
| **Shift Time [$\mu$s] (Sync 0)** | Time between the sync events and the "Output Valid" or "Input Latch" time. | 0 | -2147483648 to 2147483647 |
| **Enable (Sync 1)** | Enable the Sync 1 synchronization unit configurations. | Unmarked | Marked<br>Unmarked |
| **Sync Unit Cycle (Sync 1)** | By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle. | Unmarked | Marked<br>Unmarked |
| **User Defined (Sync 1)** | If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time ($\mu$s) field. | Unmarked | Marked<br>Unmarked |
| **Cycle Time [$\mu$s] (Sync 1)** | Show the cycle time currently set. | 100000 | 1 to 2147483647 |

| Device Configuration | Description | Default Value | Options |
|---|---|---|---|
| **Shift Time [$\mu$s] (Sync 1)** | Time between the sync events and the "Output Valid" or "Input Latch" time. | 0 | -2147483648 to 2147483647 |
| **Check Vendor ID** | If unmarked, it will disable the Vendor ID Check. | Marked | Marked<br>Unmarked |
| **Check Product ID** | If unmarked, it will disable the Product ID Check. | Marked | Marked<br>Unmarked |
| **SDO Access** | Set a time reference for the timeout check of a SDO Access. | - | 0 to 100000 |
| **I -> P** | Set a time reference for the timeout check of the switch from Init to Pre-Operation mode. | - | 0 to 100000 |
| **P -> S/S -> O** | Set a time reference for the timeout check of the switch from Pre-Operation to Safe-Operation and from Safe-Operation to Operational modes. | - | 0 to 100000 |
| **Cyclic Unit** | Set the Unit Cycle to the local microprocessor. | Unmarked | Marked<br>Unmarked |
| **Latch Unit 0** | Set the Latch Unit 0 to the local microprocessor. | Unmarked | Marked<br>Unmarked |
| **Latch Unit 1** | Set the Latch Unit 1 to the local microprocessor. | Unmarked | Marked<br>Unmarked |

Table 109: EtherCAT Slave Configurations

**Notes:**

**AutoInc Address:** This address is used only during startup, when the Master is assigning the EtherCAT addresses to the Slaves. When for this matter, the first telegram runs through the Slaves, each fast-read Slave increases its *AutoInc* Address by 1. The Slave with address 0 finally will receive the data.

**Optional:** If a Slave is declared as *Optional*, no error message will be created in case the device doesn't exist in the bus system. Thus a *Station alias* address must be defined and written to the EEPROM. This option is only available if the option *Autoconfig Master/Slaves* in the settings of the EtherCAT Master is activated and if this function is supported by the EtherCAT Slave.

**Enable Distributed Clock:** If the *Distributed Clock* functionality is enabled, the data exchange cycle time, displayed in the *Sync Unit Cycle ($\mu$s)* field will be determined by the Master *Cycle Time*. Thus the master clock can synchronize the data exchange within the network. The settings for handling the synchronization unit(s) depend on the Slave.

**Enable Sync 0:** If this option is activated, the *Sync0* synchronization unit is used. A synchronization unit describes a set of process data which is exchanged synchronously.

**Sync Unit Cycle (Sync 0):** If this option is activated, the Master *Cycle Time*, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The *Cycle Time ($\mu$s)* field shows the currently set cycle time.

**Shift Time:** The Shift Time describes the time between the sync events (*Sync0, Sync1*) and the *Output Valid* or *Input Latch* times. Writable value, if the slave supports shifting of *Output Valid* or *Input Latch*.

**Enable Sync 1:** If this option is selected, the synchronization unit *Sync1* is used. A synchronization unit is a set of process data which is exchange synchronously.

**Sync Unit Cycle (Sync1):** If this option is activated, the Master *Cycle Time*, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The *Cycle Time ($\mu$s)* field shows the currently set cycle time.

**Check Vendor ID and Product ID:** By default, at startup of the system the *Vendor ID* and/or the *Product ID* will be checked against the current configured settings. If a mismatch is detected, the bus will be stopped and no further actions will be executed. This serves to avoid the download of an erroneous configuration. This option is intended to switch off the check, if necessary.

**SDO Access:** By default there's no timeout set for the SDO list submit action at system startup. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

**I -> P:** By default there's no timeout set for the state transition from *Init* to *Pre-Operational*. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

**P -> S / S -> O:** By default there's no timeout set for the state transition from *Pre-Operational* to *Safe-Operational* and from *Safe-Operational* to *Operational*. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

**DC cycle unit control:** Choose the desired option(s) concerning the *Distributed Clock* functions in order to define, which should be assigned to the local microprocessor. The control is done in register 0x980 in the EtherCAT slave. The possible settings: *Cyclic Unit, Latch Unit 0, Latch Unit 1*.

**Enable:** If the setting Optional is not activated, this setting can be activated if explicitly supported by the device description of the slave. It allows direct assignment of an alias address in order to get the slaves address independent of its position within the bus. If the option *Optional* is activated, this checkbox is disabled.

#### *5.9.9.3.2. EtherCAT Slave - Process Data*

The *Process Data* tab of the EtherCAT Slave configurator editor shows the slave input and output process data, each defined by name, type and index by the device description file, as seen in figure below.

The selected input (to be read) and output (to be written) of the device are available in the EtherCAT Slave - I/O Mapping dialog as PLC inputs and outputs to which project variables might be mapped.

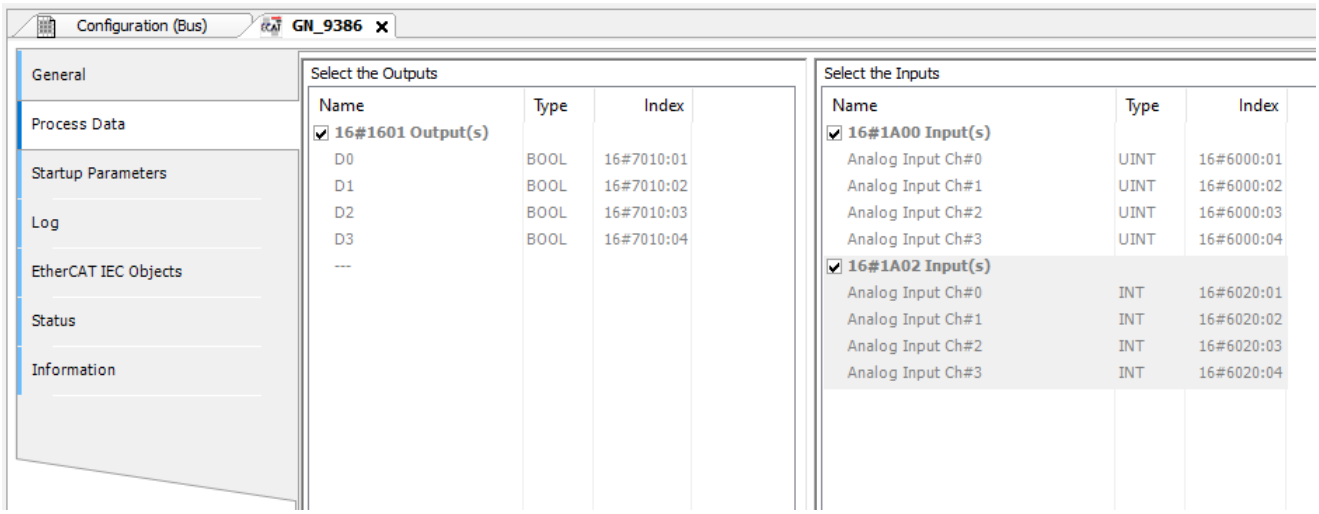



Figure 119: Process Data Dialog

The *Expert Process Data* dialog will only be available in the EtherCAT Slave configuration editor if the *Enable Expert Settings* option is activated. It provides another, more detailed, vision of the process data, adding to what is presented in the *Process Data* tab. Furthermore, the download of the *PDO Assignment* and the *PDO Configuration* can be activated in this dialog.

**ATTENTION**

If the Slave doesn't accept the PDO Configuration, it will stay in Pre-Operational state and none real time data exchange will be possible.

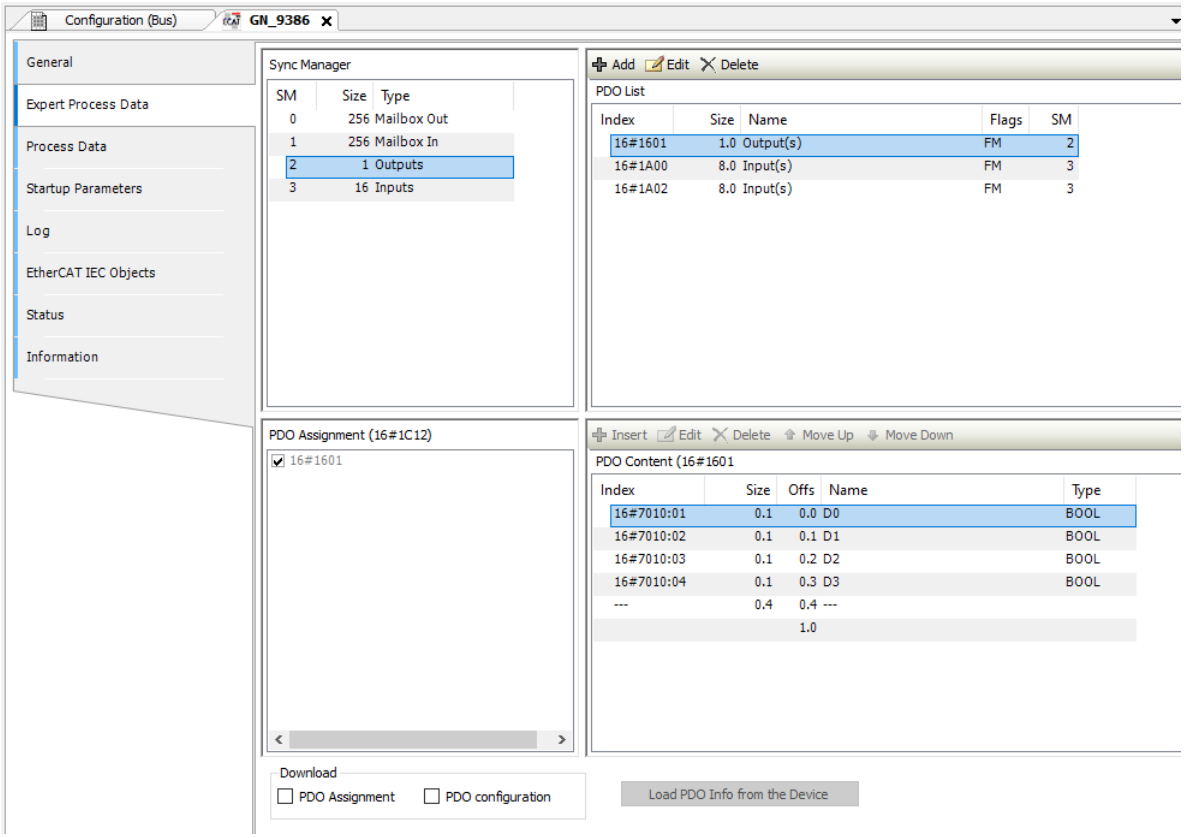



Figure 120: Expert Process Data Dialog

This dialog is divided in four sections and two options:

- *Sync Manager*: List of *Sync Manager* with data size and type of PDOs.
- *PDO Assignment*: List of PDOs assigned to the selected *Sync Manager*. The checkbox activates the PDO and I/O channels are created. It is similar to the simple PDO configuration windows. Here only PDOs can be enabled or disabled.
- *PDO List*: List of all PDOs defined in the device description file. Single PDOs can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Content*: Displays the content of the PDO selected in the section above. Entries can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Assignment*: If activated a CoE write command will be added to index 0x1CXX to write the PDO configuration 0x16XX or 0x1A00.
- *PDO Configuration*: If activated several CoE write commands will be added to write the PDO mapping to the slave.

**ATTENTION**

If a Slave doesn't support the PDO configuration, a download may result in a Slave error. This function should only be used by experts.

#### 5.9.9.3.3. EtherCAT Slave - Edit PDO List

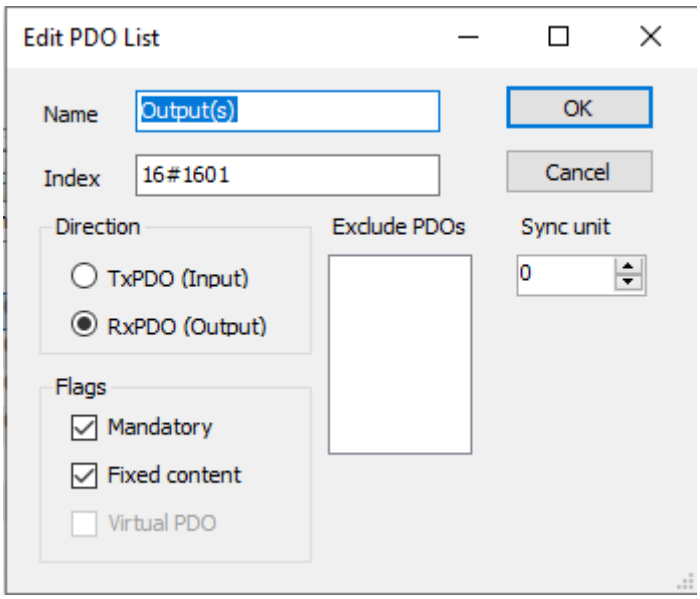



Figure 121: Edit PDO List Dialog

This dialog is opened through the context menu from the PDO List area, presented in Figure 120. Below are some explanations on the configuration options presented in this dialog.

- *Name:* Name of the PDO input.
- *Index:* Index of the PDO in being edited.
- *TxPDO (Input):* If activated, the PDO will be transferred from the Master to the Slave.
- *RxPDO (Output):* If activated, the PDO will be transferred from the Slave to the Master.
- *Mandatory:* The PDO is necessary and can't be unchecked in the *PDO Assignment* area.
- *Fixed Content:* The PDO content is fixed and can't be changed. It's not possible to add entries in the *PDO Content* panel.
- *Virtual PDO:* Reserved for future use.
- *Exclude PDOs:* It's possible to define a list of PDO that can, or can't, be selected along with the PDO being edited in the *PDO Assignment* area, or in the *Process Data* tab. If a PDO is marked in this list, it can't be selected, turning into gray in the *PDO Assignment* area when the PDO in edition is selected.
- *SyncUnit:* ID of the PDO *Sync Manager* shall assigned to.

#### 5.9.9.3.4. EtherCAT Slave - Startup Parameters

In the *Startup Parameters* tab, parameters for the device can be defined, which will be transferred by SDOs (*Service Data Objects*) or IDN at the system's startup. The options available in this tab, as well as the access possibilities, vary according to the EtherCAT Slave used and they are present in the *Device Description File*.

#### 5.9.9.3.5. EtherCAT Slave - I/O Mapping

This tab of the EtherCAT Slave configuration editor offers the possibility to assign the project variables to the EtherCAT inputs or outputs. This way, the EtherCAT Slave variables can be controlled by the *User Application*.

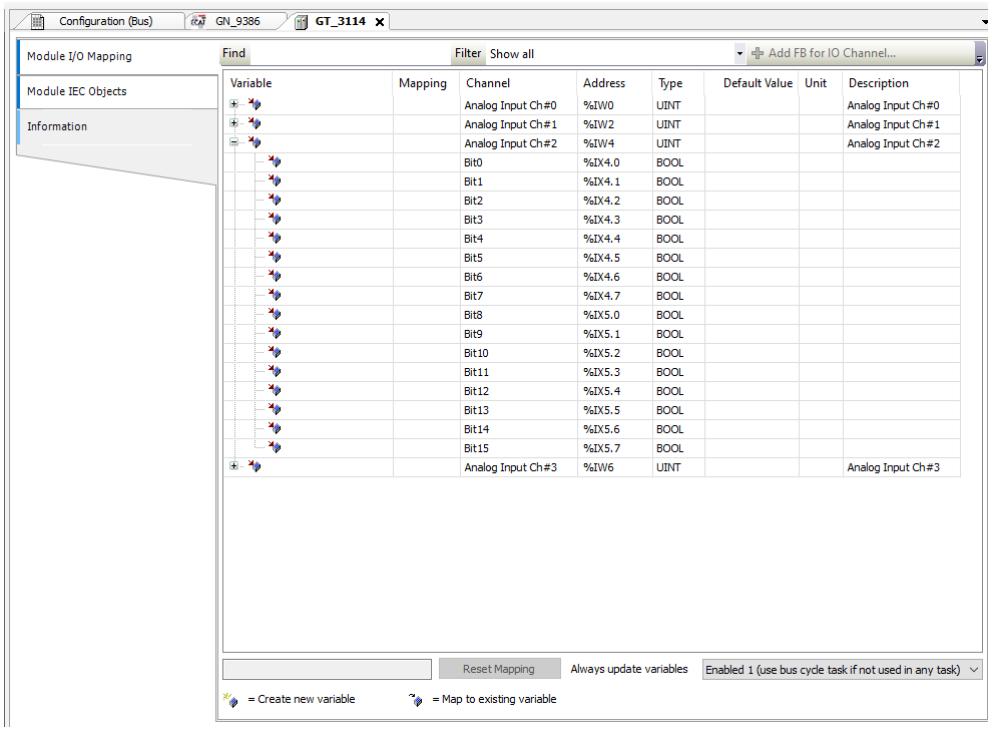



Figure 122: Slave I/O Mapping Dialog

#### *5.9.9.3.6. EtherCAT Slave - Status and Information*

The *Status* tab of the EtherCAT Slave provides status information (e.g. '*Running*', '*Stopped*') and device-specific diagnostic messages, also on the used card and the internal bus system.

The *Information* tab, presented in the EtherCAT Slave configuration editor, shows, if available, the following general information about the module: *Name, Vendor, Type, Version, Categories, Order Number, Description, Image*.

### 5.9.10. EtherNet/IP

The EtherNet/IP is a master-slave architecture protocol, consisting of an EtherNet/IP Scanner (master) and one or more EtherNet/IP Adapters (slave).

The Ethernet/IP protocol is based on CIP (*Common Industrial Protocol*), which have two primary purposes: The transport of control-oriented data associated with I/O devices and other system-related information to be controlled, such as configuration parameters and diagnostics. The first one is done through implicit messages, while the second one is done through explicit messages.

Their runtime system can act as either Scanner or Adapter. Each CPU's NET interface support only one EtherNet/IP instance and it can't be instanced on an Ethernet expansion module.

An EtherNet/IP Adapter instance supports an unlimited number of modules or Input/Output bytes. In these modules, can be added variables of types: BYTE, BOOL, WORD, DWORD, LWORD, USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL and LREAL.

**ATTENTION**

EtherNet/IP can't be used together with Ethernet Redundant Mode or with Half-Cluster's redundancy.

**ATTENTION**

To avoid communication issues, EtherNet/IP Scanner can only have Adapters configured within the same subnetwork.

#### 5.9.10.1. Addition of EtherNet/IP Scanners and Adapters

To add an EtherNet/IP Scanner or Adapter it's needed to add an *Ethernet Adapter* under the desired NET. This can be done through the command *Add Device*. Under this *Ethernet Adapter* it's possible to add a *Scanner* or an *Adapter*.

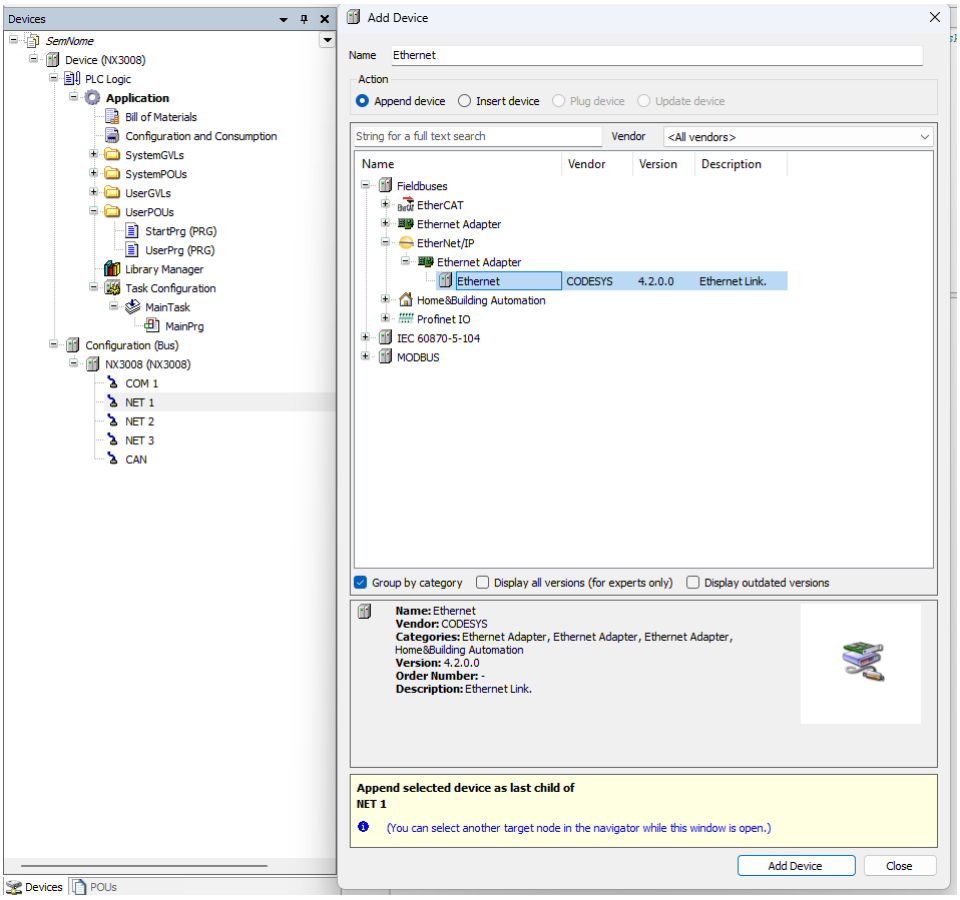



Figure 123: Adding an Ethernet Adapter

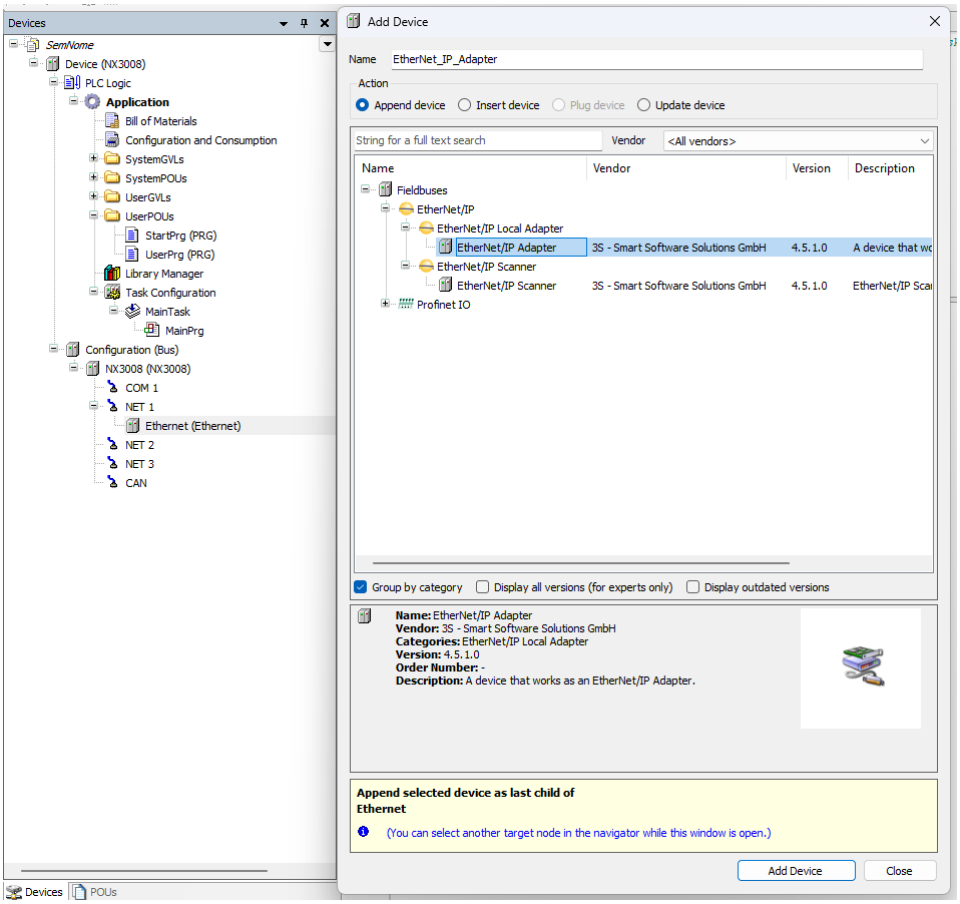



Figure 124: Adding an EtherNet/IP Adapter or Scanner

### 5.9.10.2. EtherNet/IP Scanner Configuration

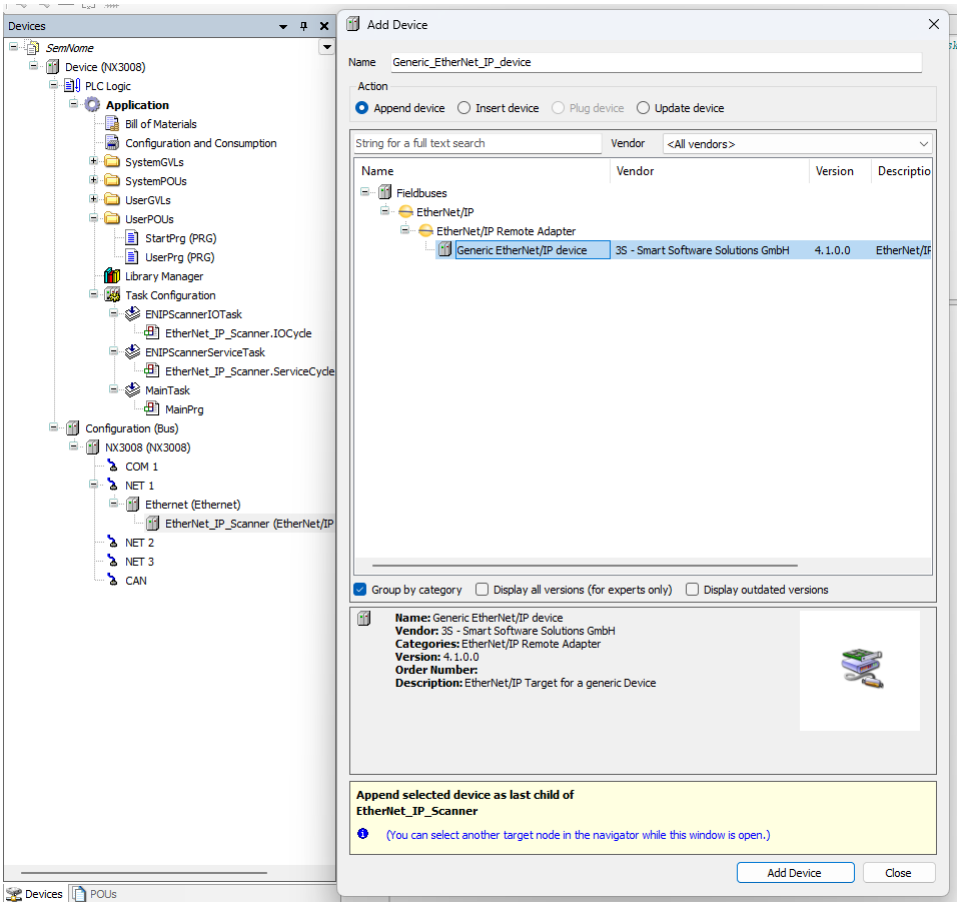



Figure 125: Adding an EtherNet/IP Adapter Under the Scanner

#### *5.9.10.2.1. General*

After open the Adapter declared under the Scanner it's possible to configure it as needed. The first Tab is *General*, on it is possible to configure the *IP address* and the *Electronic Keying* parameters. These parameters must be checked or unchecked if the adapter being used is installed on Mastertool. Otherwise, if the Adapter used is of type Generic. The Vendor ID, Device Type, Product Code, Large Revision, and Small Revision fields must be filled in with the correct vendor's information and the boxes checked as much as necessary. Altus, for its part, has its own ID, which is "1454".

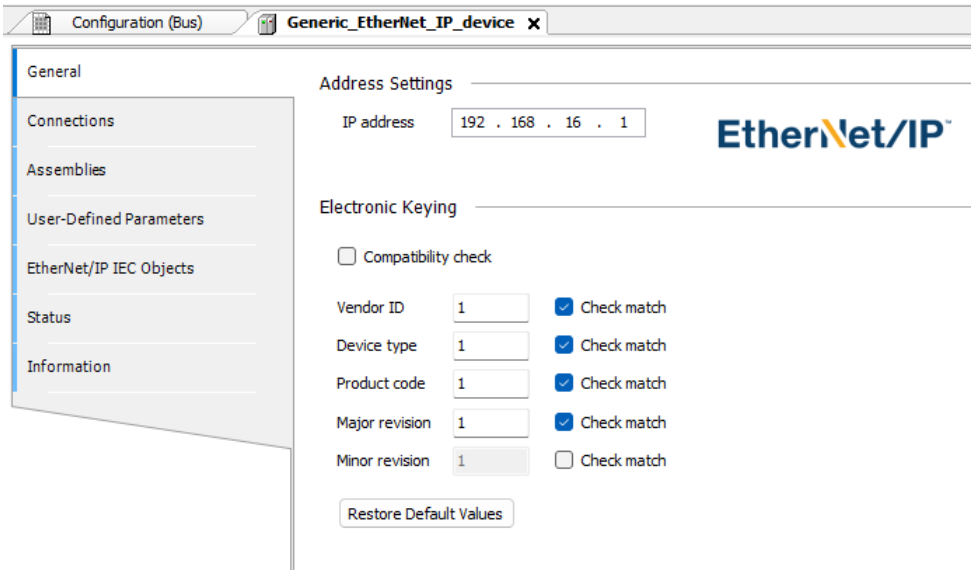



Figure 126: EtherNet/IP General Tab

### 5.9.10.2.2. *Connections*

The upper area of the *Connections* tab displays a list of all configured connections. When there is an *Exclusive Owner* connection in the EDS file, it is inserted automatically when the Adapter is added. The configuration data for these connections can be changed in the lower part of the view.

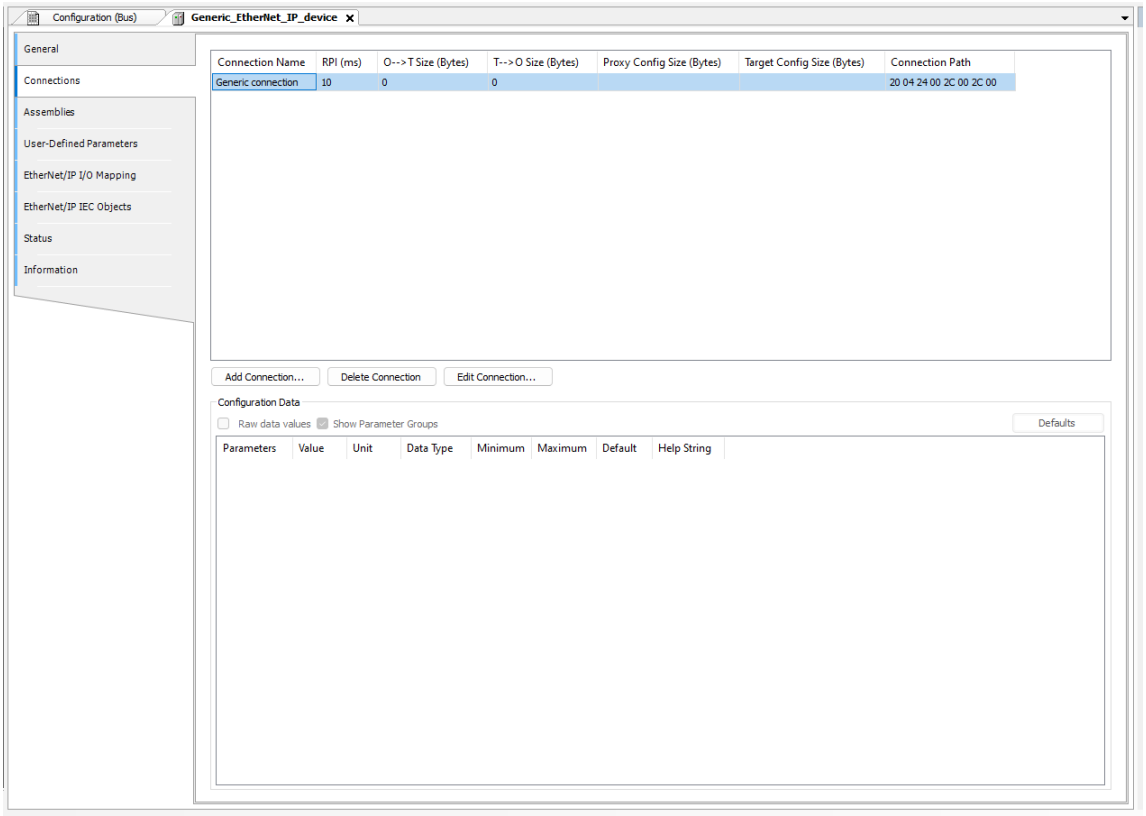



Figure 127: EtherNet/IP Connection Tab

**Notes:**

For two or more EtherNet/IP Scanners to connect to the same Remote Adapter:

1. Only one of the Scanners can establish an *Exclusive Owner* connection.
2. The same value of *RPI(ms)* must be configured for the Scanners.

The configuration data is defined in the EDS file. The data is transmitted to the remote adapter when the connection is opened.

| Configuration | Description | Default Value | Options |
|---|---|---|---|
| **RPI (ms)** | Request Packet Interval: exchange interval of the input and output data. | 10 ms | Multiple the Interval of the Bus Cycle Task to which it is associated |
| **O -> T Size (Bytes)** | Size of the producer data from the Scanner to the Adapter (O -> T) | 0 | 0 - 65527 |
| **T -> O Size (Bytes)** | Size of the consumer data from the Adapter to the Scanner (T -> O) | 0 | 0 - 65531 |
| **Proxy Config Size (Bytes)** | Proxy configuration data size | - | - |
| **Device Config Size (Bytes)** | Device configuration data size. | - | - |
| **Connection Path** | Address of the configuration objects - input objects - output objects. | Automatically generated path | Automatically generated path, User-defined path and Path defined by symbolic name |

Table 110: EtherNet/IP Connection parameters

To *add* new connections there is the button *Add Connection...* which will open the *New connection* window. In this window, you can configure a new connection type from those predefined in the Adapter's EDS or a connection from zero when using a Generic device.

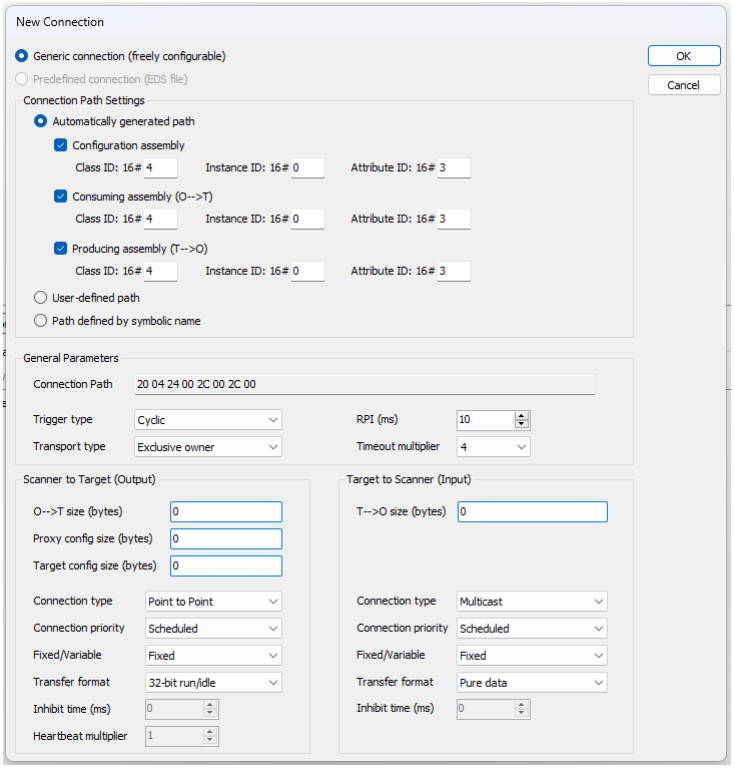



Figure 128: EtherNet/IP New Connection's Window

### 5.9.10.2.3. *Assemblies*

The upper area of the *Assemblies* tab displays a list of all configured connections. When a connection is selected, the associated inputs and outputs are displayed in the lower area of the tab.

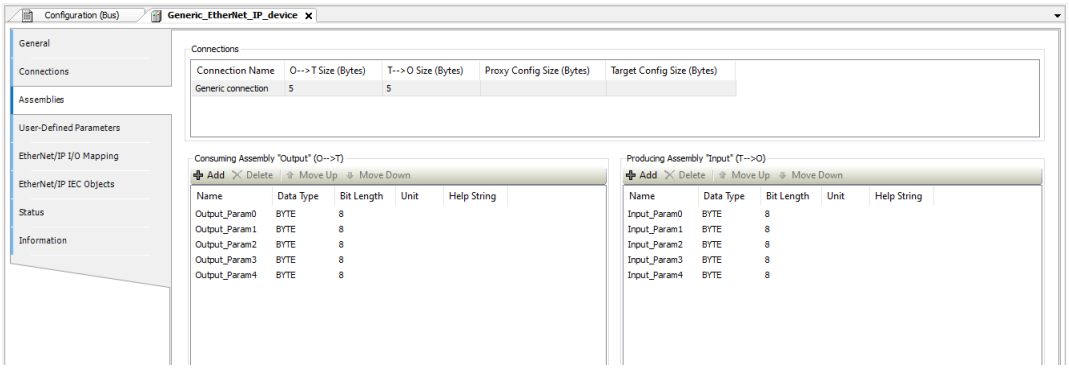



Figure 129: EtherNet/IP Assemblies Tab

*Output Assembly* and *Input Assembly*:

| Configuration | Description |
|---|---|
| **Add** | Opens the dialog box “Add Input/Output” |
| **Delete** | Deletes all selected Inputs/Outputs. |
| **Move Up** | Moves the selected Input/Output within the list. |
| **Move Down** | The order in the list determines the order in the I/O mapping. |

Table 111: EtherNet/IP Assemblies Tab

Dialog box *Add Input/Output*:

| Configuration | Description |
|---|---|
| **Name** | Name of the input/output to be inserted. |
| **Help String** | |
| **Data type** | Type of the input/output to be inserted. This type also define its Bit Length. |
| **Bit Length** | This value must not be edited. |

Table 112: EtherNet/IP “Add Input/Output” window

#### *5.9.10.2.4. EtherNet/IP I/O Mapping*

*I/O Mapping* tab shows, in the *Variable* column, the name of the automatically generated instance of the *Adapter* under *IEC Objects*. In this way, the instance can be accessed by the application. Here the project variables are mapped to adapter's inputs and outputs.

### 5.9.10.3. EtherNet/IP Adapter Configuration

The EtherNet/IP Adapter requires Ethernet/IP Modules. The Modules will provide I/O mappings that can be manipulated by user application through %I or %Q addresses according to its configuration.

New Adapters can be installed on Mastertool with the EDS Files. The configuration options may differ depending on the device description file of the added Adapter.

#### *5.9.10.3.1. General*

The first tab of the EtherNet/IP Adapter is the *General* tab. Here you can set the parameters of the *Electronic Keying* used in the scanner to check compatibility. In this tab, you can also install the EDS of the device directly in the Mastertool device repository or export it.

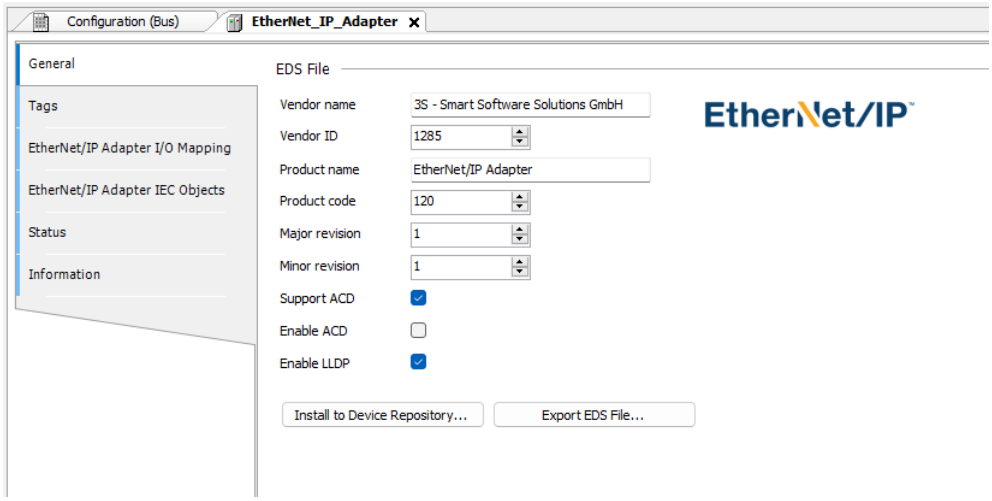



Figure 130: EtherNet/IP General Tab

#### *5.9.10.3.2. EtherNet/IP Adapter: I/O Mapping*

On the *EtherNet/IP I/O Mapping* tab, you can configure which bus cycle task the Adapter will execute.

### 5.9.10.4. EtherNet/IP Module Configuration

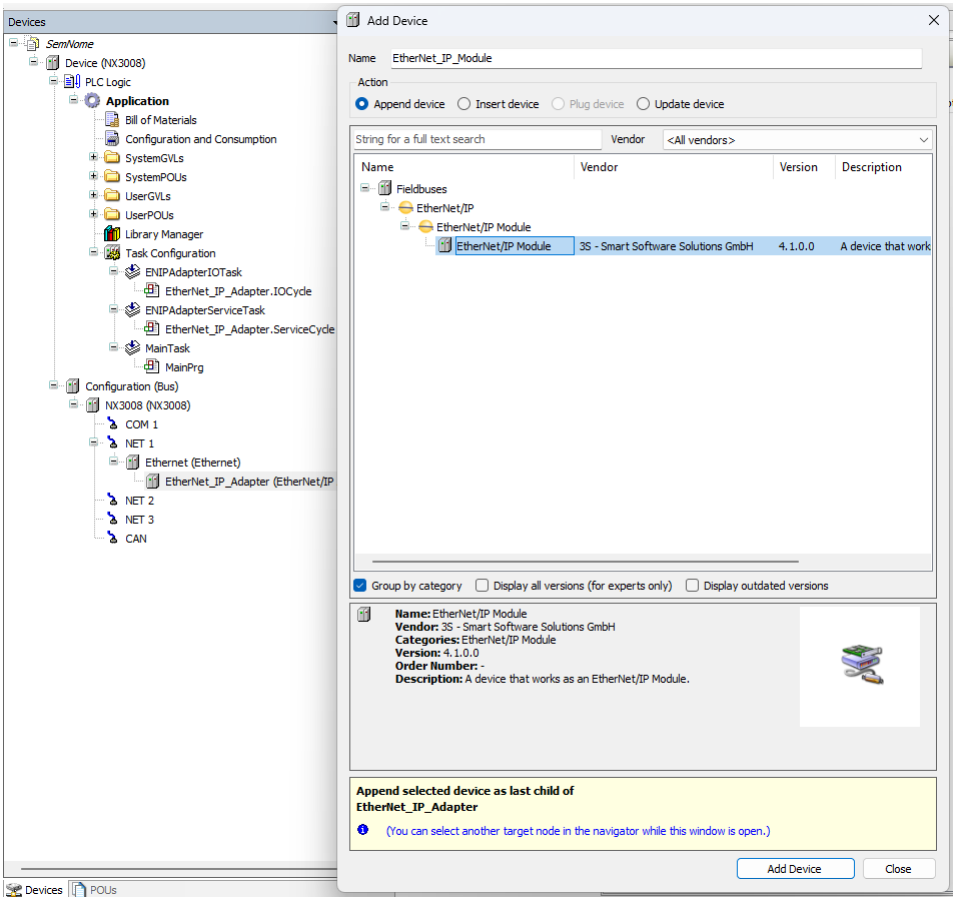



Figure 131: Adding an EtherNet/IP Module under the Adapter

#### 5.9.10.4.1. Assemblies

The parameters of the module's *General* tab follow the same rules as described in the 111 and 112 tables.

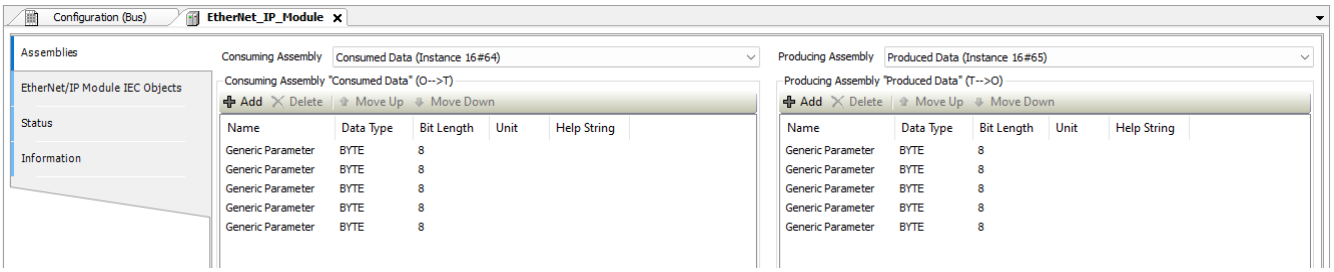



Figure 132: EtherNet/IP Module Assemblies Tab

#### 5.9.10.4.2. EtherNet/IP Module: I/O Mapping

The I/O Mapping tab shows, in the *Variable* column, the name of the automatically generated Adapter instances. In this way, the instance can be accessed by the user application.

### 5.9.11. CANopen Manager

CANopen is a protocol based on CAN bus which provides fast I/O update (around 5 ms for a 1000 kbit/s network with a few slaves) with a simple twisted pair physical bus infrastructure.

The CANopen Manager (master) is responsible for controlling the slave devices, managing their operation state and exchanging I/O and other service data. By default, the CANopen manager protocol activities (bus cycle) are executed on the context of MainTask, keeping it synchrounous with the execution of application code.

The configuration of CANopen network is performed with the support of EDS files, which describes the I/O data and service objects (PDO and SDO) of the slave and must be provided by the device manufacturer.

Additionally, an application library called CiA405 is provided with FunctionBlocks which allows to perform several specific actions like changing the slave state (NMT), receiving emergency object, querying the slave state and performing SDO read/write commands. The complete description of CiA405 library can be found on *Online Help* (F1) of Mastertool.

**ATTENTION**

- Only one CANopen Manager instance per project is allowed
- The CANopen specification allows up to 127 nodes (including Manager).

A special care must be taken considering the physical bus lenght and the selected baudrate. The following table shows the maximum bus lenght that can be used safely with a given baudrate:

| Baudrate | Maximum Bus Length |
|---|---|
| **1000 kbit/s** | 25 m |
| **500 kbit/s** | 100 m |
| **250 kbit/s** | 250 m |
| **125 kbit/s** | 500 m |
| **100 kbit/s** | 700 m |
| **< 50 kbit/s** | 1000 m |

Table 113: Baudrate X Bus Length

#### 5.9.11.1. Installing and inserting CANopen Devices

The configuration of a CANopen network uses the same standard procedure of other fieldbuses configuration on Mastertool.

To add a CANopen Manager, right-click on the *CAN* interface and select *Add Device*. Expand the itens until finding *CANopen_Manager* device and click on the *Add Device* button. The *CANopen Manager* device will appear below the CAN interface as shown on the following picture:

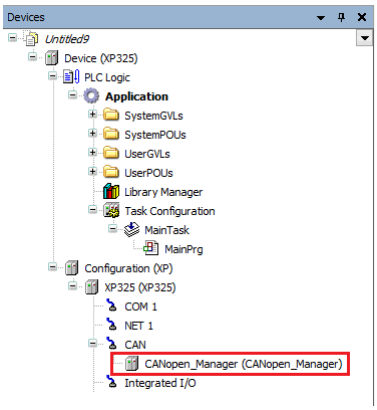



Figure 133: Adding CANopen Manager

To add a CANopen slave device, first you need to install it on the *Device Repository*. To do that, go to *Tools -> Device Repository* and install the device EDS file.

After that, right-click on the *CANopen_Manager* device and click on *Add Device*. Search the devices you desire and click on *Add Device* button like shown on the following picture:

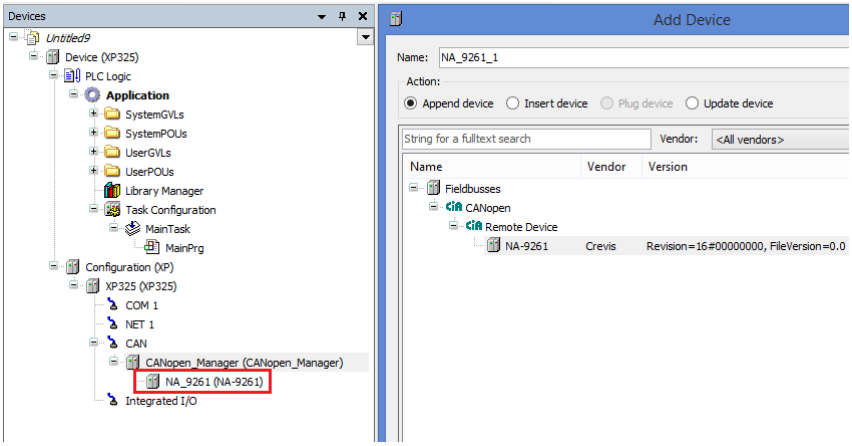



Figure 134: Adding CANopen Slave Device

### 5.9.11.2. CANOpen Manager Configuration

The CANopen Manager comes with a ready-to-use configuration (default values). Typically, it is just needed to set the correct baudrate and slave address to have a network running.

The main parameters of CANopen manager are located at *General* tab:

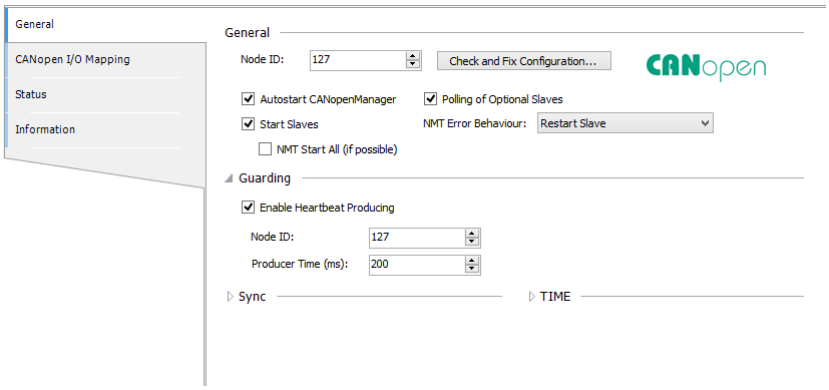



Figure 135: CANopen Manager general parameters

The detailed description of CANopen Manager general parameters can be found on section *Device Editors -> CANopen* of Mastertool's online help (F1).

Additionally, the tab *CANopen I/O Mapping* allows to change the bus cycle task:

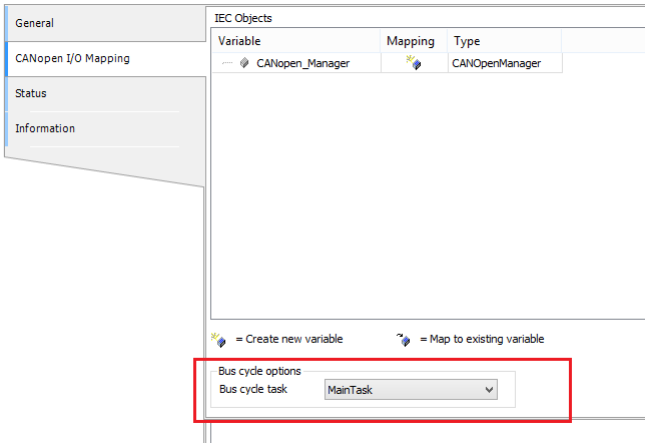



Figure 136: CANopen Manager bus cycle task setting

By default, the bus cycle task is configured to use the MainTask. This is the recommended setting for most of the applications. Changing this setting is only required on a very specific scenario which requires the implementation of a time-critical control loop using CANopen I/O (5ms lets say) that can not be performed on MainTask due to heavy application code.

#### 5.9.11.3. CANopen Slave Configuration

The configuration of CANopen Remote Devices (Slaves) is separated in the first four tabs shown on the following picture:

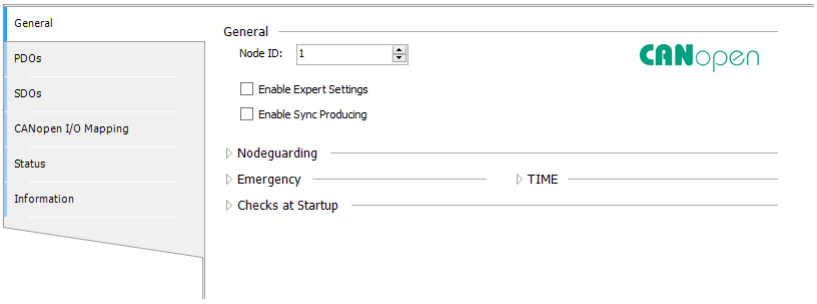



Figure 137: CANopen Slave parameters

The *General* tab contains the slave address (Node ID), Nodeguarding and Emergency object settings.

The *PDO* tab contains the configuration of process data (I/O data) that will be exchanged.

The *SDO* tab contains the SDO objects which can be selected to be accessed by SDO read/write FunctionBlock provided by CiA405 library.

The detailed description of CANopen Slave parameters can be found on section "*Device Editors -> CANopen* of Mastertool Online Help (F1).

#### 5.9.12. PROFINET Controller

For correct use of the PROFINET Controller protocol, it is necessary to consult the manual MU214621 - Nexto Series PROFINET Manual .

## 5.10. Communication Performance

### 5.10.1. MODBUS Server

The MODBUS devices configurable in the Nexto CPU run in the background, with a priority below the user application and cyclically. Thus, their performance varies depending on the remaining time, taking into account the difference between the interval and time that the application takes to run. For example, a MODBUS device in an application that runs every 100 ms, with a running time of 50 ms, will have a lower performance than an application running every 50 ms to 200 ms of interval. It happens because in the latter case, the CPU will have a longer time between each MainTask cycle to perform the tasks with lower priority.

It also has to be taken into account the number of cycles that the device, slave or server takes to respond to a request. To process and transmit a response, a MODBUS RTU Slave will takes two cycles (cycle time of the MODBUS task), where as a MODBUS Ethernet Server task takes only one cycle. But this is the minimum time between receipt of a request and the reply. If the request is sent immediately after the execution of a task MODBUS cycle time may be equal to 2 or 3 times the cycle time for the MODBUS slave and from 1 to 2 times the cycle time for the MODBUS server.

In this case: Maximum Response Time = 3 * (cycle time) + (time of execution of the tasks) + (time interframe chars) + (send delay).

For example, for a MODBUS Ethernet Server task with a cycle of 50 ms, an application that runs for 60 ms every 100 ms, the server is able to run only one cycle between each cycle of the application. On the other hand, using the same application, running for 60 ms, but with an interval of 500 ms, the MODBUS performs better, because while the application is not running, it will be running every 50 ms and only each cycle of MainTask it will take longer to run. For these cases, the worst performance will be the sum of the Execution Time of the user application with the cycle time of the MODBUS task.

For the master and client devices the operating principle is exactly the same, but taking into account the polling time of the MODBUS relation and not the cycle time of the MODBUS task. For these cases, the worst performance of a relationship will be performed after the polling time, along with the user application Execution Time.

It is important to stress that the running MODBUS devices number also changes its performance. In an user application with Execution Time of 60 ms and interval of 100 ms, there are 40 ms left for the CPU to perform all tasks of lower priority. Therefore, a CPU with only one MODBUS Ethernet Server will have a higher performance than a CPU that uses four of these devices.

#### 5.10.1.1. CPU's Local Interfaces

For a device MODBUS Ethernet Server, we can assert that the device is capable to answer a x number of requisitions per second. Or, in other words, the Server is able to transfer n bytes per second, depending on the size of each requisition. As smaller is the cycle time of the MODBUS Server task, higher is the impact of the number of connections in his answer rate. However, for cycle times smaller than 20 ms this impact is not linear and the table below must be viewed for information.

The table below exemplifies the number of requisitions that a MODBUS Server inserted in a integrated Ethernet interface is capable to answer, according to the cycle time configured for the MODBUS task and the number of active connections:

| Number of Active Connections | Answered requisitions per second with the MODBUS task cycle at 5 ms | Answered requisitions per second with the MODBUS task cycle at 10 ms | Answered requisitions per second with the MODBUS task cycle at 20 ms |
|---|---|---|---|
| **1 Connection** | 185 | 99 | 50 |
| **2 Connections** | 367 | 197 | 100 |

| Number of Active Connections | Answered requisitions per second with the MODBUS task cycle at 5 ms | Answered requisitions per second with the MODBUS task cycle at 10 ms | Answered requisitions per second with the MODBUS task cycle at 20 ms |
|---|---|---|---|
| **4 Connections** | 760 | 395 | 200 |
| **7 Connections** | 1354 | 695 | 350 |
| **10 Connections** | 1933 | 976 | 500 |

Table 114: Communication Rate of a MODBUS Server at Integrated Interface

**ATTENTION**

The communication performances mentioned in this section are just examples, using a CPU with only one device MODBUS TCP Server, with no logic to be executed inside the application that could delay the communication. Therefore, these performances must be taken as the maximum rates.

For cycle times equal or greater than 20 ms, the increase of the answer rate is linear, and may be calculated using an equation:

N = C x (1 / T )

Where:

N is the medium number of answers per second;

C is the number of active connections;

T is the MODBUS task interval in seconds.

As an example a MODBUS Server, with only one active connection and a cycle time of 50 ms we get:

C = 1; T = 0,05 s;

N = 1 x (1 / (0,05))

N = 20

That is, in this configuration the MODBUS Server answers, on average, 20 requisitions per second.

In case the obtained value is multiplied by the number of bytes in each requisition, we will obtain a transfer rate of n bytes per second.

### 5.10.2. OPC UA Server

The OPC UA Server MU214609 analyzes the performance of OPC UA communication in greater detail, including addressing the consumption of Ethernet communication bandwidth. This manual also discusses concepts about the operation of the OPC UA protocol.

## 5.11. User Web Pages

Also called *Web Visualization*, or simply *Webvisu*, this feature allows to implement a simplified SCADA embedded into the PLC. The Visualization screens are developed on the same enviroment of the PLC application using the programming system. Once the application is downloaded, the PLC starts a web server hosting this special web page.

The complete information about this functionality can be found in the Help of the programming system.

## 5.12. SNMP

### 5.12.1. Introduction

SNMP (*Simple Network Management Protocol*) is a protocol widely used by network administrators to provide important information and diagnostic equipment present in a given Ethernet network.

This protocol uses the concept of agent and manager, in which the manager sends read requests or write certain objects to the agent. Through a MIB (*Management Information Base*) the manager is aware of existing objects in the agent, and thus can make requests of these objects, respecting the read permissions or writing the same. MIB is a collection of information organized hierarchically with each object of this tree is called OID (*Object Identifier*).

For all equipment with SNMP, it is mandatory to support MIB-II. In this MIB are described key information for managing Ethernet networks.

### 5.12.2. SNMP in Nexto XF Controllers

The CPUs of the Nexto Series behave as agents in SNMP communication. The information made available through SNMP cannot be manipulated or accessed through the user application, requiring an external SNMP manager to perform access. The table below contains the objects available in the Nexto CPUs. The supported protocol versions are: SNMPv1, SNMPv2c and SNMPv3 (MIB-II supported objects are described in RFC-1213).

| OID | Name | Description |
|---|---|---|
| 1.3.6.1.2.1.1 | System | Contains name, description, location and other equipment identification information. |
| 1.3.6.1.2.1.2 | Interfaces | Contains information of the machine's network interfaces. The ifTable (OID 1.3.6.1.2.1.2.2) has the indexes 5 and 6 available, which can be viewed by the network interfaces statistics NET 1 and NET 2, respectively, of the Nexto XF CPUs. |
| 1.3.6.1.2.1.3 | At | Contains information of the last required connections to the agent. |
| 1.3.6.1.2.1.4 | IP | Contains statistical connections using IP protocol. |
| 1.3.6.1.2.1.5 | ICMP | Contains statistical connections using ICMP protocol. |
| 1.3.6.1.2.1.6 | TCP | Contains statistical connections using TCP protocol. |
| 1.3.6.1.2.1.7 | UDP | Contains statistical connections using UDP protocol. |
| 1.3.6.1.2.1.11 | SNMP | Contains statistical connections using SNMP protocol. |

Table 115: MIB II Objects – Nexto XF SNMP Agent

By default, the SNMP agent is activated, i.e., the service is initialized at the time the CPU is started. The access to the agent information is via the Ethernet interfaces of the Nexto Series CPUs on UDP port 161. So when the service is active, the agent information can be accessed through any one of the Ethernet interfaces available. In figure below, an example is shown where a SNMP manager presents some read values.

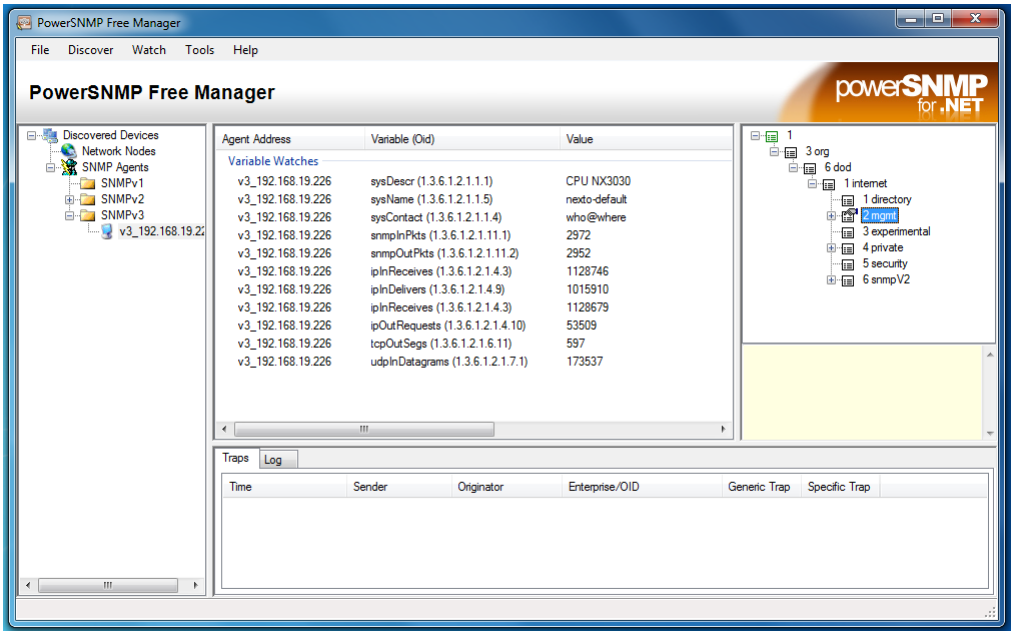



Figure 138: SNMP Manager Example

For SNMPv3, in which there is user authentication and password to requests via SNMP protocol, is provided a standard user described in the User and SNMP Communities section.

If you want to disable the service, change the SNMPv3 user or communities for SNMPv1 / v2c predefined, you must access the System Web Page of the CPU. For details, see the Configuration SNMP section.

### 5.12.3. Configuration SNMP

SNMP settings can be changed through the System Web Page, in the *CPU Management* tab, under the *SNMP* menu. After successful login, the current state of the service (enabled or disabled) as well as the user information SNMPv3 and communities for SNMPv1 / v2c can be viewed.

The user can enable or disable the service via a checkbox at the top of the screen.

It's also possible to change the SNMPv3 information by clicking the *Change* button just below the user information. This action will open a form where the user must fill the old username and old password, also the new username and the new password. Any other information can not be changed.

To change the information from the SNMPv1/v2c *Communities* group data, the process is similar, just click the *Change* button below the group information. In opened screen, new data can be filled in the *rocommunity* and *rwcommunity* fields. If any field is left blank, its correspondent *community* will be disabled. If both fields are left blank, the access to the SNMP agent will only be possible through SNMPv3.

If the user wants to return to the default settings, it must be manually reconfigured according to the available information in the User and SNMP Communities section. Therefore, all current SNMP configurations will be kept in the firmware update process. These options are shown in figure below.

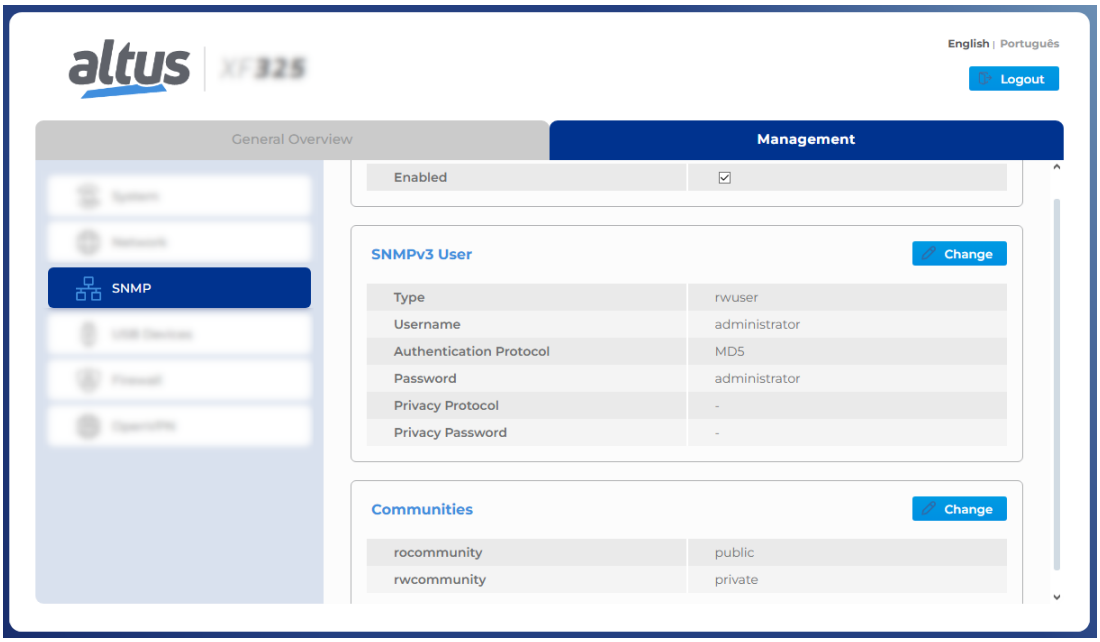



Figure 139: SNMP status configuration screen

### 5.12.4. User and SNMP Communities

Prior to access the SNMPv1 / v2c protocol informations, there are two *communities* settings to be configured, according to table below.

| Communities | Default String | Type |
|---|---|---|
| rocommunity | Public | Only read |
| rwcommunity | Private | Read and Write |

Table 116: SNMPv1/v2c Default Communities info

It's possible to access SNMPv3 using default user, see table below:

| Username | Type | Authentication Protocol | Password | Privacy Protocol | Privacy Password |
|---|---|---|---|---|---|
| administrator | rwuser | MD5 | administrator | - | - |

Table 117: SNMPv3 Default User info

For *communities* settings, user and password, some limits must be respected, as described on the following table:

| Configurable item | Minimum Size | Max Size | Allowed Characters |
|---|---|---|---|
| rocommunity | - | 30 | [0-9][a-z][A-Z]@$*_ |
| rwcommunity | - | 30 | [0-9][a-z][A-Z]@$*_ |
| V3 User | 1 | 30 | [0-9][a-z][A-Z]@$*_ |
| V3 Password | 8 | 30 | [0-9][a-z][A-Z]@$*_ |

Table 118: SNMP settings limits

## 5.13. System Performance

In cases where the application has only one MainTask user task responsible for the execution of a single Program type programming unit called MainPrg (as in Single Profile), the PLC consumes a certain amount of time for the task to be processed. At that time we call it as *Execution Time*.

In an application the average application *Execution Time* can be known using Mastertool in the *Device* item of its *Devices Tree* as follows:

*PLC Logic-> Application-> Task Configuration* in the *Monitor* tab, *Average Cycle Time* column.

The user must pay attention to the *Cycle Time* so that it does not exceed 80% of the interval set in the MainTask user task. For example, in an application where the interval is 100 ms, an appropriate *Cycle Time* is up to 80 ms. This is due to the fact that the CPU needs time to perform other tasks such as communication processing, processing of the display and memory card, and these tasks take place within the range (the remaining 20% of *Cycle Time*).

### 5.13.1. Memory Card

Data transfers involving the memory card is performed by the CPU in the background, as this gives priority to the execution of user application and communication processing. Thus, the transfer of files to the card may suffer an additional significant time, depending on the Cycle Time of the user application.

The time required to read/write files on the card will be directly affected by the Cycle Time of the user application since this application has priority in execution.

Further information about the use of the memory card see Memory Card section.

## 5.14. RTC Clock

The CPUs have an internal clock that can be used through the *NextoStandard.lib* library. This library is automatically loaded during the creation of a new project (to perform the library insertion procedure, see Libraries section). The figure below shows how to include the blocks in the project:

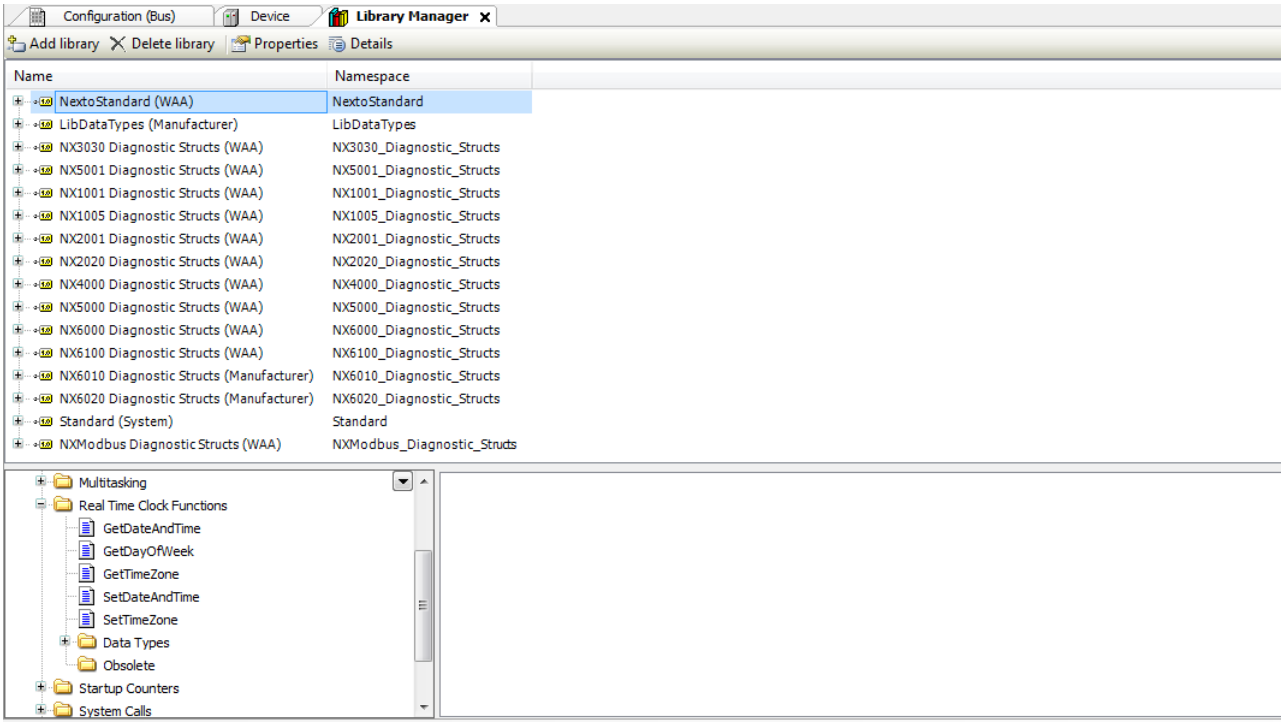



Figure 140: Clock Reading and Writing Blocks

### 5.14.1. Function Blocks for RTC Reading and Writing

Among other function blocks, there are some very important used for clock reading (*GetDateAndTime*, *GetDayOfWeek* and *GetTimeZone*) and for date and time new data configuring (*SetDateAndTime* and *SetTimeZone*). These functions always use the local time, that is, take into account the value defined by the *Time Zone*.

The proceedings to configure these two blocks are described below.

#### 5.14.1.1. Function Blocks for RTC Reading

The clock reading can be made through the following functions:

#### 5.14.1.1.1. GetDateAndTime

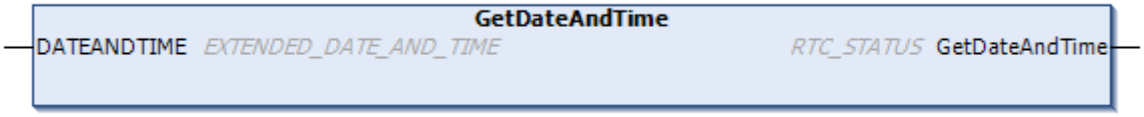



Figure 141: Date and Hour Reading

| Input Parameters | Type | Description |
|---|---|---|
| **DATEANDTIME** | EXTENDED_DATE _AND_TIME | This variable returns the value of date and hour of RTC in the format shown at Table 128. |

Table 119: Input Parameters of GetDateAndTime

| Output Parameters | Type | Description |
|---|---|---|
| **GETDATEANDTIME** | RTC_STATUS | Returns the function error state, see Table 130. |

Table 120: Output Parameters of GetDateAndTime

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Result : RTC_STATUS;
DATEANDTIME : EXTENDED_DATE_AND_TIME;
xEnable : BOOL;
END_VAR
-------------------------------------------------------------------------
IF xEnable = TRUE THEN
Result := GetDateAndTime(DATEANDTIME);
xEnable := FALSE;
END_IF
```

#### 5.14.1.1.2. GetTimeZone

The following function reads the Time Zone configuration, this function is directly related with time in Time Zone at SNTP synchronism service:

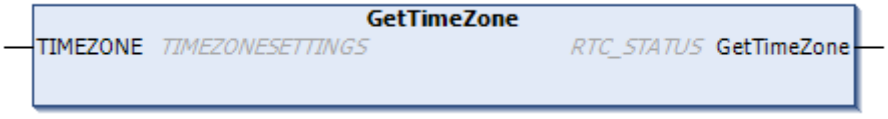



Figure 142: Configuration Reading of Time Zone

| Input Parameters | Type | Description |
|---|---|---|
| **TIMEZONE** | TIMEZONESETTINGS | This variable presents the reading of Time Zone configuration. |

Table 121: Input Parameters of GetTimeZone

| Output Parameters | Type | Description |
|---|---|---|
| **GetTimeZone** | RTC_STATUS | Returns the function error state, see Table 130. |

Table 122: Output Parameters of GetTimeZone

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
GetTimeZone_Status   : RTC_STATUS;
TimeZone       : TIMEZONESETTINGS;
xEnable : BOOL;
END_VAR
-------------------------------------------------------------------------
IF xEnable = TRUE THEN
GetTimeZone_Status := GetTimeZone(TimeZone);
xEnable := FALSE;
END_IF
```

#### 5.14.1.1.3. *GetDayOfWeek*

*GetDayOfWeek* function is used to read the day of the week.


Figure 143: Day of Week Reading

| Output Parameters | Type | Description |
|---|---|---|
| **GetDayOfWeek** | DAYS_OF_WEEK | Returns the day of the week. See Section 129. |

Table 123: Output Parameters of GetDayOfWeek

When called, the function will read the day of the week and fill the structure *DAYS_OF_WEEK*.
Utilization example in ST language:

```
PROGRAM UserPrg
VAR
DayOfWeek : DAYS_OF_WEEK;
END_VAR
-------------------------------------------------------------------------
DayOfWeek := GetDayOfWeek();
```

#### 5.14.1.2. RTC Writing Functions

The clock settings are made through function and function blocks as follows:

##### *5.14.1.2.1. SetDateAndTime*

*SetDateAndTime* function is used to write the settings on the clock. Typically the precision is on the order of hundreds of milliseconds.

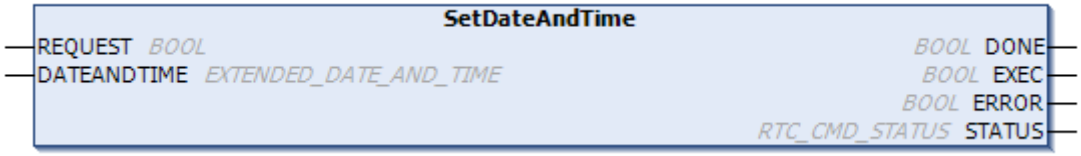



Figure 144: Set Date And Time

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when receives a rising edge, enables the clock writing. |
| **DATEANDTIME** | EXTENDED_DATE_AND_TIME | Receives the values of date and hour with milliseconds. See section 128. |

Table 124: Input Parameters of SetDateAndTime

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable, when true, indicates that the action was successfully completed. |
| **EXEC** | BOOL | This variable, when true, indicates that the function is processing the values. |
| **ERROR** | BOOL | This variable, when true, indicates an error during the Writing. |
| **STATUS** | RTC_STATUS | Returns the error occurred during the configuration. See Table 130. |

Table 125: Output Parameters of SetDateAndTime

When a rising edge occurs at the *REQUEST* input, the function block will write the new *DATEANDTIME* values on the clock. If the writing is successfully done, the *DONE* output will be equal to *TRUE*. Otherwise, the *ERROR* output will be equal to *TRUE* and the error will appear in the *STATUS* variable.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
SetDateAndTime : SetDateAndTime;
xRequest : BOOL;
DateAndTime : EXTENDED_DATE_AND_TIME;
xDone : BOOL;
xExec : BOOL;
xError : BOOL;
xStatus : RTC_STATUS;
END_VAR
-------------------------------------------------------------------------
IF xRequest THEN
  SetDateAndTime.REQUEST:=TRUE;
  SetDateAndTime.DATEANDTIME:=DateAndTime;
  xRequest:= FALSE;
END_IF
SetDateAndTime();
SetDateAndTime.REQUEST:=FALSE;
IF SetDateAndTime.DONE THEN
  xExec:=SetDateAndTime.EXEC;
  xError:=SetDateAndTime.ERROR;
  xStatus:=SetDateAndTime.STATUS;
END_IF
```

**ATTENTION**

If the user attempts to write time values outside the RTC range, the values will be converted to valid ones, as long as they do not exceed the range described in section RTC Operating Limits. For example, if the user tries to write the value 2000 ms, it will be converted to 2 seconds; if 100 seconds are written, it will be converted to 1 minute and 40 seconds; if 30 hours are written, it will be converted to 1 day and 6 hours, and so on.

#### 5.14.1.2.2. *SetTimeZone*

The following function block makes the writing of the time zone settings:

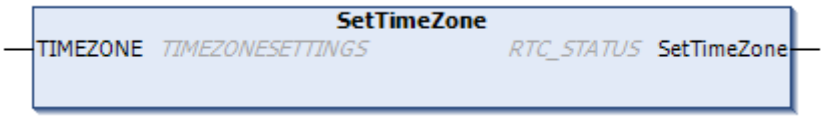



Figure 145: Writing of the Time zone Settings

| Input parameters | Type | Description |
|---|---|---|
| **TIMEZONE** | TIMEZONESETTINGS | Structure with time zone to be configured. See Table 131. |

Table 126: SetTimeZone Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **SetTimeZone** | RTC_STATUS | Returns the error occurred during the reading/setting. See Table 130. |

Table 127: SetTimeZone Output Parameters

When called, the function will configure the *TIMEZONE* with the new system time zone configuration. The configuration results is returned by the function.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Status : RTC_STATUS;
TimeZone : TIMEZONESETTINGS;
xWrite : BOOL;
END_VAR
--------------------------------------------------------------------------------
//FB SetTimeZone
IF (xWrite = TRUE) THEN
Status := SetTimeZone(TimeZone);
  IF Status = RTC_STATUS.NO_ERROR THEN
    xWrite := FALSE;
  END_IF
END_IF
```

**ATTENTION**

To adjust the clock, time and date values must be within the range described in section RTC Operating Limits; otherwise, an error will be reported through the *STATUS* output parameter. For more details about the *STATUS* output parameter, refer to section RTC_STATUS.

### 5.14.2. RTC Data Structures

The reading and setting function blocks of the Nexto Series CPUs RTC use the following data structures in its configuration:

#### 5.14.2.1. EXTENDED_DATE_AND_TIME

This structure is used to store the RTC date when used the function blocks for date reading/setting within milliseconds of accuracy. It is described in the table below:

| Structure | Type | Variable | Description |
|---|---|---|---|
| **EXTENDED_DATE_AND_TIME** | BYTE | byDayOfMonth | Stores the day of the set date. |
| | BYTE | ByMonth | Stores the month of the set date. |
| | WORD | wYear | Stores the year of the set date. |
| | BYTE | byHours | Stores the hour of the set date. |
| | BYTE | byMinutes | Stores the minutes of the set date. |
| | BYTE | bySeconds | Stores the seconds of the set date. |
| | WORD | wMilliseconds | Stores the milliseconds of the set date. |

Table 128: EXTENDED_DATE_AND_TIME

#### 5.14.2.2. DAYS_OF_WEEK

This structure is used to store the day of week:

| Enumerable | Value | Description |
|---|---|---|
| **DAYS_OF_WEEK** | 0 | INVALID_DAY |
| | 1 | SUNDAY |
| | 2 | MONDAY |
| | 3 | TUESDAY |
| | 4 | WEDNESDAY |
| | 5 | THURSDAY |
| | 6 | FRIDAY |
| | 7 | SATURDAY |

Table 129: DAYS_OF_WEEK Structure

#### 5.14.2.3. RTC_STATUS

This enumerator is used to return the type of error in the RTC setting or reading and it is described in the table below:

| Enumerator | Value | Description |
|---|---|---|
| **RTC_STATUS** | NO_ERROR (0) | There is no error. |
| | UNKNOWN_COMMAND (1) | Unknown command. |
| | DEVICE_BUSY (2) | Device is busy. |
| | DEVICE_ERROR (3) | Device with error. |
| | ERROR_READING_OSF (4) | Error in the reading of the valid date and hour flag. |
| | ERROR_READING_RTC (5) | Error in the date and hour reading. |
| | ERROR_WRITING_RTC (6) | Error in the date and hour writing. |
| | ERROR_UPDATING_SYSTEM_TIME (7) | Error in the update of the system's date and hour. |
| | INTERNAL_ERROR (8) | Internal error. |
| | INVALID_TIME (9) | Invalid date and hour. |
| | INPUT_OUT_OF_RANGE (10) | Out of the limit of valid date and hour for the system. |

| Enumerator | Value | Description |
|---|---|---|
| | SNTP_NOT_ENABLE (11) | Error generated when the SNTP service is not enabled and it is done an attempt for modifying the time zone. |

Table 130: RTC_STATUS

#### 5.14.2.4. TIMEZONESETTINGS

This structure is used to store the time zone value in the reading/setting requests of the RTC's function blocks and it is described in table below:

| Structure | Type | Variable | Description |
|---|---|---|---|
| **TIMEZONESETTINGS** | INT | iHour | Set time zone hour. |
| | INT | iMinutes | Set time zone minute. |

Table 131: TIMEZONESETTINGS

**Note:**

**Function Blocks of Writing and Reading of Date and Hour:** different libraries of *NextoStandard*, which have function blocks or functions that may perform access of reading and writing of date and hour in the system, are not indicated. The *NextoStandard* library has the appropriate interfaces for writing and reading the system's date and hour accordingly and for informing the correct diagnostics.

### 5.14.3. RTC Operating Limits

The values presented in Table 132 indicate the earliest and latest time instants that the RTC clock can represent. The minimum value corresponds to the earliest date and time that can be set, while the maximum value represents the latest valid future date and time. These limits must be respected when setting or reading the product's clock, ensuring that all date and time functions operate correctly.

| Parameter | Minimum | Maximum |
|---|---|---|
| Day | 1 | 31 |
| Month | 1 | 12 |
| Year | 2000 | 2099 |
| Hour | 0 | 23 |
| Minute | 0 | 59 |
| Second | 0 | 59 |
| Millisecond | 0 | 999 |

Table 132: Valid value range for the RTC Clock

## 5.15. User Files Memory

Nexto Series CPUs have a memory area destined to the general data storage, in other words, the user can store several project files of any format in the CPU memory. This memory area varies according to the CPU model used (check Memory section).

In order to use this area, the user must access a project in Mastertool and click on the *Devices Tree*, placed at the program left. Double click on the *Device* item and, after selecting the CPU in the *Communication Settings* tab which will be open, select the *Files* tab and click on *Refresh*, both in the computer files column (left) and in the CPU files column (right) as shown on figure below.

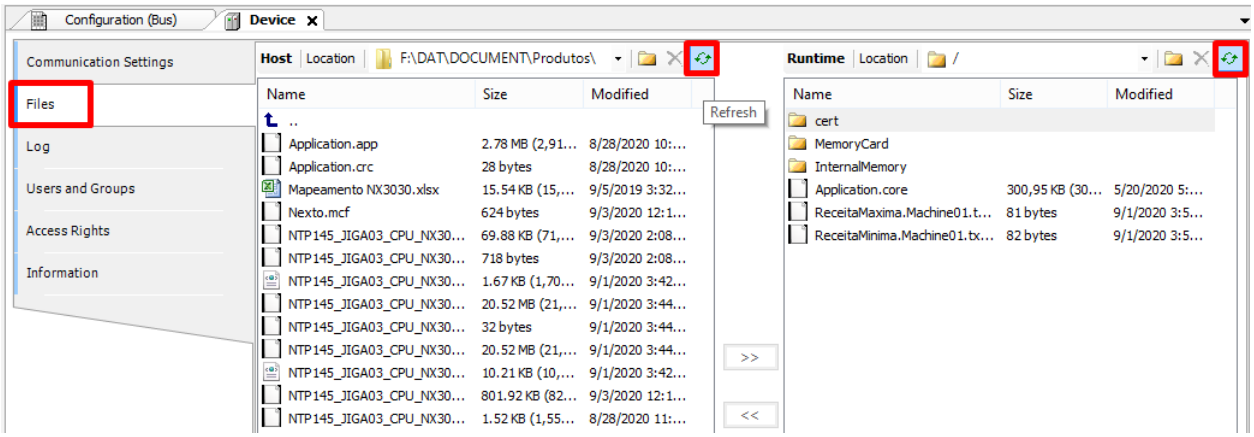



Figure 146: User Files Access

After updating the CPU column of files, the root directory of files stored in the CPU will be shown. Then it will be possible to select the folder where the files will be transferred to. The *"InternalMemory"* folder is a default folder to be used to store files in the CPU's internal memory, since it is not possible to transfer files to the root directory. If necessary, the user can create other folders in the root directory or subfolders inside the *"InternalMemory"* folder.

The *"MemoryCard"* folder is the directory where the memory card is mounted, if it is inserted into the CPU. Files which are transferred to the *"MemoryCard"* are being transferred directly into the memory card. As new features are being added to the product, some folders may appear and which should be ignored by the user.

In order to perform a file transfer from the microcomputer to the CPU just select the desired file in the left column and press the "»" key located in the center of the screen, as shown in figure below. The download time will vary depending on file size and cycle time (execution) of the current application of the CPU and may take several minutes.

The user does not need to be in *Run* Mode or connected to the CPU to perform the transfers, since it has the ability to connect automatically when the user performs the transfer.

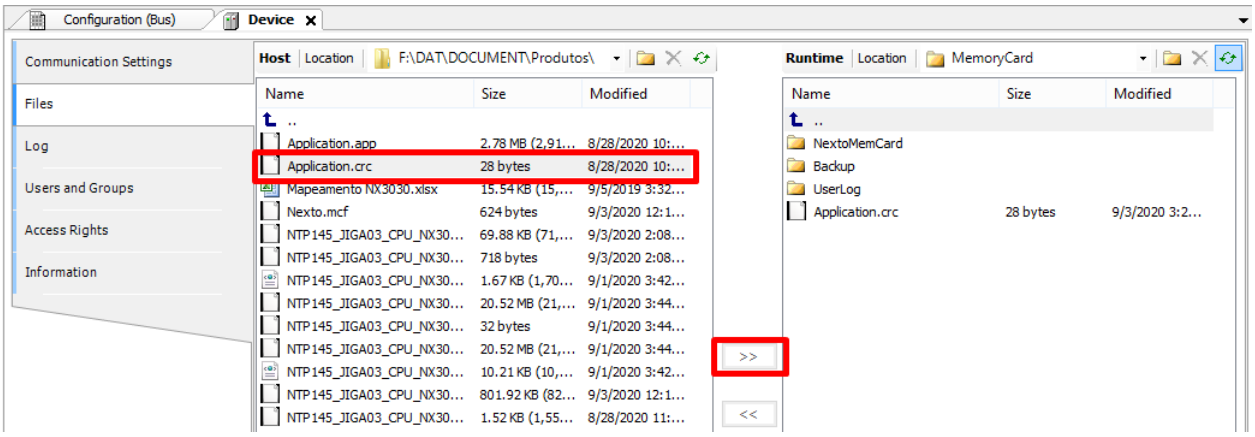



Figure 147: Files Transference

**ATTENTION**

The files contained in the folder of a project created by Mastertool have special names reserved by the system in this way cannot be transferred through the *Files* tab. If the user wishes to transfer a project to the user memory, you must compact the folder and then download the compressed file (*.zip for example).

In case it is necessary to transfer documents from the CPU to the PC in which Mastertool is installed, the user must follow a very similar procedure to the previously described, as the file must be selected from the right column and the button "«" pressed, placed on the center of the screen.

Furthermore, the user has some operation options in the storing files area, which are the following:

- New directory : allows the creation of a new folder in the user memory area.
- Delete item : allows the files excluding in the folders in the user memory area.
- Refresh : allows the file updating, on Mastertool screen, of the files in the user memory area and in the computer.

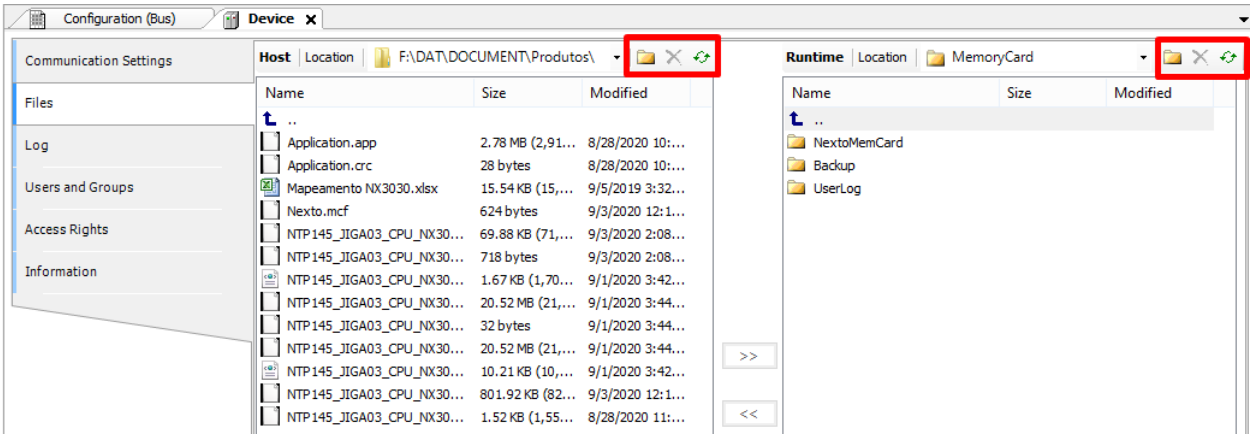



Figure 148: Utilization Options

**ATTENTION**

For a CPU in Stop Mode or with no application, the transfer rate to the internal memory is approximately 150 Kbytes/s.

## 5.16. Function Blocks and Functions

### 5.16.1. Special Function Blocks for Serial Interfaces

The special function blocks for serial interfaces make possible the local access (COM 1 and COM 2 - if available) and also access to remote serial ports (expansion modules). Therefore, the user can create his own protocols and handle the serial ports as he wishes, following the IEC 61131-3 languages available in Mastertool. The blocks are available inside the *NextoSerial* library which must be added to the project so it's possible to use them (to execute the library insertion procedure, see programming manual in section Library).

The special function blocks for serial interfaces can take several cycles (consecutive calls) to complete the task execution. Sometimes a block can be completed in a single cycle, but in the general case, needs several cycles. The task execution associated to a block can have many steps which some depend on external events, that can be significantly delayed. The function block cannot implement routines to occupy the time while waits for these events, because it would make the CPU busy. The solution could be the creation of blocking function blocks, but this is not advisable because it would increase the user application complexity, as normally, the multitask programming is not available. Therefore, when an external event is waited, the serial function blocks are finished and the control is returned to the main program. The task treatment continues in the next cycle, in other words, on the next time the block is called.

Before describing the special function blocks for serial interfaces, it is important to know the *Data types*, it means, the data type used by the blocks.

| Data type | Options | Description |
|---|---|---|
| SERIAL_BAUDRATE | BAUD200 | Lists all baud rate possibilities (bits per second) |
| | BAUD300 | |
| | BAUD600 | |
| | BAUD1200 | |
| | BAUD1800 | |
| | BAUD2400 | |

| Data type | Options | Description |
|---|---|---|
| | BAUD4800 | |
| | BAUD9600 | |
| | BAUD19200 | |
| | BAUD38400 | |
| | BAUD57600 | |
| | BAUD115200 | |
| SERIAL_DATABITS | DATABITS_5 | Lists all data bits possibilities. |
| | DATABITS_6 | |
| | DATABITS_7 | |
| | DATABITS_8 | |
| SERIAL_HANDSHAKE | Defines all modem signal possibilities for the configurations: | |
| | RS232_RTS | Controls the Nexto CPU RS-232C port. The transmitter is enabled to start the transmission and disabled as soon as possible after the transmission is finished. For example, can be used to control a RS-232/RS-485 external converter. |
| | RS232_RTS_OFF | Controls the RS-232C port of the Nexto CPU. The RTS signal is always off. |
| | RS232_RTS_ON | Controls the RS-232C port of the Nexto CPU. The RTS signal is always on. |
| | RS232_RTS_CTS | Controls the RS-232C port of the Nexto CPU. In case the CTS is disabled, the RTS is enabled. Then waits for the CTS to be enabled to get the transmission and RTS restarts as soon as possible, at the end of transmission. Ex: Controlling radio modems with the same modem signal. |
| | RS232_MANUAL | Controls the RS-232C port of the Nexto CPU. The user is responsible to control all the signals (RTS, DTR, CTS, DSR, DCD). |
| SERIAL_MODE | NORMAL_MODE | Serial Communication Normal Operation mode. |
| | EXTENDED_MODE | Serial Communication Extended Operation mode in which are provided information about the received data frame. |
| SERIAL_PARAMETERS | Defines all configuration parameters of the serial port: | |
| | BAUDRATE | Defined in SERIAL_BAUDRATE. |
| | DATABITS | Defined in SERIAL_DATABITS. |
| | STOPBITS | Defined in SERIAL_STOPBITS. |
| | PARITY | Defined in SERIAL_PARITY. |
| | HANDSHAKE | Defined in SERIAL_HANDSHAKE. |

| Data type | Options | Description |
|---|---|---|
| | UART_RX_THRESHOLD | Byte quantity which must be received to generate a new UART interruption. Lower values make the TIMESTAMP more precise when the EXTENDED MODE is used and minimizes the overrun errors. However, values too low may cause too many interruptions and delay the CPU. |
| | MODE | Defined in SERIAL_MODE. |
| | ENABLE_RX_ON_TX | When true, all the received byte during the transmission will be discharged instead going to the RX line. Used to disable the full-duplex operation in the RS-422 interface. |
| | ENABLE_DCD_EVENT | When true, generates an external event when the DCD is modified. |
| | ENABLE_CTS_EVENT | When true, generates an external event when the CTS is modified. |
| SERIAL_PARITY | PARITY_NONE | List all parity possibilities. |
| | PARITY_ODD | |
| | PARITY_EVEN | |
| | PARITY_MARK | |
| | PARITY_SPACE | |
| SERIAL_PORT | COM 1 | List all available serial ports (COM 10, COM 11, COM 12, COM 13, COM 14, COM 15, COM 16, COM 17, COM 18 and COM 19 – expansion modules). |
| | COM 2 (if available) | |
| SERIAL_RX_CHAR_ EXTENDED | Defines a character in the RX queue in extended mode. | |
| | RX_CHAR | Data byte. |
| | RX_ERROR | Error code. |
| | RX_TIMESTAMP | Silence due to the previous character or due to another event which has happen before this character (serial port configuration, transmission ending). |
| | It has some fields which deliver information regarding RX queue status/error, used when the normal format is utilized (no error and timestamp information): | |
| | RX_FRAMING_ERRORS | Frame errors counter: character incorrect formation – no stop bit, incorrect baud rate, among other – since the serial port configuration. Returns to zero when it reaches the maximum value (65535). |
| | RX_PARITY_ERRORS | Parity errors counter, since the serial port configuration. Returns to zero when it reaches the maximum value (65535). |

| Data type | Options | Description |
|---|---|---|
| SERIAL_RX_QUEUE_ STATUS | RX_BREAK_ERRORS | Interruption errors counter, since the serial port configuration, in other words, active line higher than the character time. Returns to zero when it reaches the maximum value (65535). |
| | RX_FIFO_OVERRUN_ ERRORS | FIFO RX overrun errors counter, since the serial port configuration, in other words, error in the FIFO RX configured threshold. Returns to zero when it reaches the maximum value (65535). |
| | RX_QUEUE_OVERRUN_ ERRORS | RX queue overrun errors counter, in other words, the maximum characters number (1024) was overflowed and the data are being overwritten. Returns to zero when it reaches the maximum value (65535). |
| | RX_ANY_ERRORS | Sum the last 5 error counters (frame, parity, interruption, RX FIFO overrun, RX queue overrun). |
| | RX_REMAINING | Number of characters in the RX queue. |
| | List of critic error codes that can be returned by the serial function block. Each block returns specific errors, which will be described below: | |
| | NO_ERROR | No errors. |
| | ILLEGAL_* | Return the parameters with invalid values or out of range:<br>- SERIAL_PORT<br>- SERIAL_MODE<br>- BAUDRATE<br>- DATA_BITS<br>- PARITY<br>- STOP_BITS<br>- HANDSHAKE<br>- UART_RX_THRESHOLD<br>- TIMEOUT<br>- TX_BUFF_LENGTH<br>- HANDSHAKE_METHOD<br>- RX_BUFF_LENGTH |
| | PORT_BUSY | Indicates the serial port is being used by another instance |
| | HW_ERROR_UART | Hardware error detected in the UART. |
| | HW_ERROR_REMOTE | Hardware error at communicating with the remote serial port. |
| | CTS_TIMEOUT_ON | Time-out while waiting for the CTS enabling, in the RS-232 RTS/CTS handshake, in the SERIAL_TX block. |

| Data type | Options | Description |
|---|---|---|
| SERIAL_STATUS | CTS_TIMEOUT_OFF | Time-out while waiting for the CTS disabling, in the RS-232 RTS/CTS handshake, in the SERIAL_TX block. |
| | TX_TIMEOUT_ERROR | Time-out while waiting for the transmission ending in the SERIAL_TX. |
| | RX_TIMEOUT_ERROR | Time-out while waiting for all characters in the SERIAL_RX block or the SERIAL_RX_EXTENDED block. |
| | FB_SET_CTRL_ NOT_ALLOWED | The SET_CTRL block can't be used in case the handshake is different from RS232_MANUAL. |
| | FB_GET_CTRL_ NOT_ALLOWED | The GET_CTRL block can't be used in case the handshake is different from RS232_MANUAL. |
| | FB_SERIAL_RX_ NOT_ALLOWED | The SERIAL_RX isn't available for the RX queue, extended mode. |
| | FB_SERIAL_RX_ EXTENDED_NOT_ALLOWED | The SERIAL_RX_EXTENDED isn't available for the RX queue, normal mode. |
| | DCD_INTERRUPT_ NOT_ALLOWED | The interruption by the DCD signal can't be enabled in case the serial port doesn't have the respective pin. |
| | CTS_INTERRUPT_ NOT_ALLOWED | The interruption by the CTS signal can't be enabled in case the handshake is different from RS232_MANUAL or in case the serial port doesn't have the respective pin. |
| | DSR_INTERRUPT_ NOT_ALLOWED | The interruption by the DSR signal can't be enabled in case the serial port doesn't have the respective pin. (Nexto CPUs don't have this signal in integrated ports) |
| | NOT_CONFIGURED | The function block can't be used before the serial port configuration. |
| | INTERNAL_ERROR | Indicates that an internal problem has ocurred in the serial port. |
| SERIAL_STOPBITS | STOPBITS_1 | List all Stop Bits possibilities. |
| | STOPBITS_2 | |
| | STOPBITS_1_5 | |

Table 133: Serial Function Blocks Data types

#### 5.16.1.1. SERIAL_CFG

This function block is used to configure and initialize the desired serial port. After the block is called, every RX and TX queue associated to the serial ports and the RX and TX FIFO are restarted.


Figure 149: Serial Configuration Block

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |
| **PARAMETERS** | SERIAL_PARAMETERS | This structure defines the serial port configuration parameters, as described in the SERIAL_PARAMETERS data type. |

Table 134: SERIAL_CFG Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- ILLEGAL_SERIAL_MODE<br>- ILLEGAL_BAUDRATE<br>- ILLEGAL_DATA_BITS<br>- ILLEGAL_PARITY<br>- ILLEGAL_STOP_BITS<br>- ILLEGAL_HANDSHAKE<br>- ILLEGAL_UART_RX_THRESHOLD<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- DCD_INTERRUPT_NOT_ALLOWED<br>- CTS_INTERRUPT_NOT_ALLOWED<br>- DSR_INTERRUPT_NOT_ALLOWED |

Table 135: SERIAL_CFG Output Parameters

Utilization example in ST language, after the library Nexto Serial is inserted in the project:

```
PROGRAM UserPrg
VAR
Config: SERIAL_CFG;
Port: SERIAL_PORT := COM1;
Parameters: SERIAL_PARAMETERS := (BAUDRATE := BAUD9600,
DATABITS := DATABITS_8,
STOPBITS := STOPBITS_1,
PARITY := PARITY_NONE,
HANDSHAKE := RTS,
ENABLE_DCD_EVENT := FALSE,
ENABLE_CTS_EVENT := FALSE,
UART_RX_THRESHOLD := 8,
MODE :=NORMAL_MODE,
ENABLE_RX_ON_TX := FALSE);
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Config.REQUEST := TRUE;
Config.PORT := Port;
```

```
Config.PARAMETERS := Parameters;
//FUNCTION:
Config();
//OUTPUTS:
Config.DONE;
Config.EXEC;
Config.ERROR;
Status := Config.STATUS;    //If it is necessary to treat the error.
```

#### 5.16.1.2. SERIAL_GET_CFG

The function block is used to capture the desired serial port configuration.


Figure 150: Block to Capture the Serial Configuration

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |

Table 136: SERIAL_GET_CFG Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |

| Output parameters | Type | Description |
|---|---|---|
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- NOT_CONFIGURED |
| **PARAMETERS** | SERIAL_PARAMETERS | This structure receives the serial port configuration parameters, as described in the SERIAL_PARAMETERS data type. |

Table 137: SERIAL_GET_CFG Output Parameters

Utilization example in ST language, after the library is inserted in the project:

```
PROGRAM UserPrg
VAR
GetConfig: SERIAL_GET_CFG;
Port: SERIAL_PORT := COM1;
Parameters: SERIAL_PARAMETERS;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
GetConfig.REQUEST := TRUE;
GetConfig.PORT := Port;
//FUNCTION:
GetConfig();
//OUTPUTS:
GetConfig.DONE;
GetConfig.EXEC;
GetConfig.ERROR;
Status := GetConfig.STATUS; //If it is necessary to treat the error.
Parameters := GetConfig.PARAMETERS; //Receive the parameters of desired serial
   port.
```

#### 5.16.1.3. SERIAL_GET_CTRL

This function block is used to read the CTS, DSR and DCD control signals, in case they are available in the serial port. A false value will be returned when there are not control signals.


Figure 151: Block Used to Visualize the Control Signals

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |

Table 138: SERIAL_GET_CTRL Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- FB_GET_CTRL_NOT_ALLOWED<br>- NOT_CONFIGURED |
| **CTS_VALUE** | BOOL | Value read in the CTS control signal. |
| **DSR_VALUE** | BOOL | Value read in the DSR control signal. |
| **DCD_VALUE** | BOOL | Value read in the DCD control signal. |

Table 139: SERIAL_GET_CTRL Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Get_Control: SERIAL_GET_CTRL;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Get_Control.REQUEST := TRUE;
Get_Control.PORT := Port;
//FUNCTION:
Get_Control();
//OUTPUTS:
Get_Control.DONE;
Get_Control.EXEC;
Get_Control.ERROR;
Status := Get_Control.STATUS;   //If it is necessary to treat the error.
Get_Control.CTS_VALUE;
Get_Control.DSR_VALUE;
Get_Control.DCD_VALUE;
```

#### 5.16.1.4. SERIAL_GET_RX_QUEUE_STATUS

This block is used to read some status information regarding the RX queue, specially developed for the normal mode, but it can also be used in the extended mode.


Figure 152: Block Used to Visualize the RX Queue Status

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |

Table 140: SERIAL_GET_RX_QUEUE_STATUS Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- NOT_CONFIGURED |
| **RXQ_STATUS** | SERIAL_RX_ QUEUE_STATUS | Returns the RX queue status/error, as described in the SERIAL_RX_QUEUE_STATUS data type. |

Table 141: SERIAL_GET_RX_QUEUE_STATUS Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Get_Status: SERIAL_GET_RX_QUEUE_STATUS;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
Status_RX: SERIAL_RX_QUEUE_STATUS;
END_VAR
//INPUTS:
Get_Status.REQUEST := TRUE;
Get_Status.PORT := Port;
//FUNCTION:
Get_Status();
//OUTPUTS:
Get_Status.DONE;
Get_Status.EXEC;
Get_Status.ERROR;
Status := Get_Status.STATUS;    //If it is necessary to treat the error.
Status_RX := Get_Status.RXQ_STATUS; //If it is necessary to treat the error of
   the RX.
```

#### 5.16.1.5. SERIAL_PURGE_RX_QUEUE

This function block is used to clean the serial port RX queue, local and remote. The UART RX FIFO is restarted too.

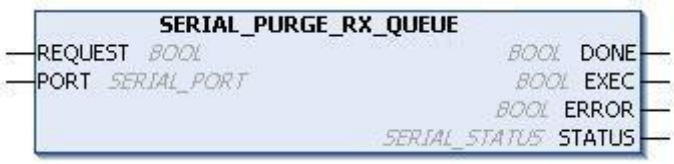



Figure 153: Block Used to Clean the RX Queue

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |

Table 142: SERIAL_PURGE_RX_QUEUE Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It's false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It's false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It's false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- NOT_CONFIGURED |

Table 143: SERIAL_PURGE_RX_QUEUE Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Purge_Queue: SERIAL_PURGE_RX_QUEUE;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Purge_Queue.REQUEST := TRUE;
Purge_Queue.PORT := Port;
//FUNCTION:
Purge_Queue();
//OUTPUTS:
Purge_Queue.DONE;
Purge_Queue.EXEC;
Purge_Queue.ERROR;
Status := Purge_Queue.STATUS; //If it is necessary to treat the error.
```

#### 5.16.1.6. SERIAL_RX

This function block is used to receive a serial port buffer, using the RX queue normal mode. In this mode, each character in the RX queue occupy a single byte which has the received data, storing 5, 6, 7 or 8 bits, according to the serial interface configuration.


Figure 154: Block Used to Read the Reception Buffer Values

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |
| **RX_BUFFER_ POINTER** | POINTER TO BYTE | Pointer of a byte array to receive the buffer values. |
| **RX_BUFFER_ LENGTH** | UINT | Specify the expected character number in the byte array. In case more than the expected bytes are available, only the expected quantity will be read from the byte array, the rest will be leaved in the RX queue (maximum size equal to 1024 characters). |

| Input parameters | Type | Description |
|---|---|---|
| **RX_TIMEOUT** | UINT | Specify the time-out to receive the expected character quantity. In case it is smaller than the necessary to receive the characters, the RX_TIME-OUT_ERROR output from the STATUS parameter will be indicated. When the specified value, in ms, is equal to zero, the function will return the data within the buffer. |

Table 144: SERIAL_RX Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- ILLEGAL_RX_BUFF_LENGTH<br>- RX_TIMEOUT_ERROR<br>- FB_SERIAL_RX_NOT_ALLOWED<br>- NOT_CONFIGURED |
| **RX_RECEIVED** | UINT | Returns the received characters number. This number can be within zero and the configured value in RX_BUFFER_LENGTH. In case it is smaller, an error will be indicated by the function block. |
| **RX_REMAINING** | UINT | Returns the number of characters which are still in the RX queue after the function block execution. |

Table 145: SERIAL_RX Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Receive: SERIAL_RX;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..1023] OF BYTE;    //Max size.
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Receive.REQUEST := TRUE;
Receive.PORT := Port;
Receive.RX_BUFFER_POINTER := ADR(Buffer_Pointer);
Receive.RX_BUFFER_LENGTH := 1024;     //Max size.
Receive.RX_TIMEOUT := 10000;
//FUNCTION:
Receive();
//OUTPUTS:
Receive.DONE;
Receive.EXEC;
Receive.ERROR;
Status := Receive.STATUS;   //If it is necessary to treat the error.
Receive.RX_RECEIVED;
Receive.RX_REMAINING;
```

#### 5.16.1.7. SERIAL_RX_EXTENDED

This function block is used to receive a serial port buffer using the RX queue extended mode as shown in the Serial Interface section.

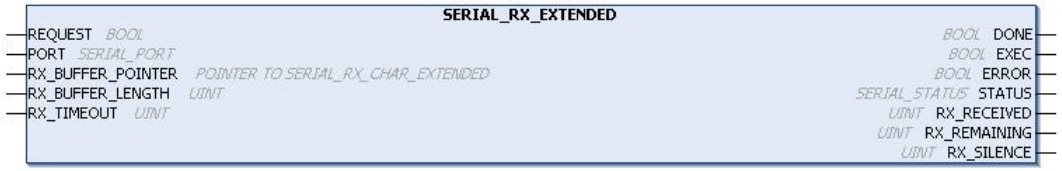



Figure 155: Block Used for Reception Buffer Reading

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |
| **RX_BUFFER_POINTER** | POINTER TO SERIAL_RX_CHAR_EXTENDED | Pointer of a SERIAL_RX_CHAR_EXTENDED array to receive the buffer values. |

| Input parameters | Type | Description |
|---|---|---|
| **RX_BUFFER_ LENGTH** | UINT | Specify the expected character number in the SERIAL_RX_CHAR_EXTENDED array. In case more than the expected bytes are available, only the expected quantity will be read from the byte array, the rest will be leaved in the RX queue (maximum size equal to 1024 characters). |
| **RX_TIMEOUT** | UINT | Specify the time-out to receive the expected character quantity. In case it is smaller than the necessary to receive the characters, the RX_TIMEOUT_ERROR output from the STATUS parameter will be indicated. When the specified value, in ms, is equal to zero, the function will return the data within the buffer. |

Table 146: SERIAL_RX_EXTENDED Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- ILLEGAL_RX_BUFF_LENGTH<br>- RX_TIMEOUT_ERROR<br>- FB_SERIAL_RX_EXTENDED_NOT _ALLOWED<br>- NOT_CONFIGURED |
| **RX_RECEIVED** | UINT | Returns the received characters number. This number can be within zero and the configured value in RX_BUFFER_LENGTH. In case it is smaller, an error will be indicated by the function block. |

| Output parameters | Type | Description |
|---|---|---|
| **RX_REMAINING** | UINT | Returns the number of characters which are still in the RX queue after the function block execution. |
| **RX_SILENCE** | UINT | Returns the silence time in the RX queue, measured since the last received character is finished. The time unit is 10 $\mu$s. This output parameter type is important to detect the silence time in protocols as MODBUS RTU. It might not be the silence time after the last received character by this function block, as it is only true if RX_REMAINING = 0. |

Table 147: SERIAL_RX_EXTENDED Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Receive_Ex: SERIAL_RX_EXTENDED;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..1023] OF SERIAL_RX_CHAR_EXTENDED;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Receive_Ex.REQUEST := TRUE;
Receive_Ex.PORT := Port;
Receive_Ex.RX_BUFFER_POINTER := ADR(Buffer_Pointer);
Receive_Ex.RX_BUFFER_LENGTH := 1024;      //Max size.
Receive_Ex.RX_TIMEOUT := 10000;
//FUNCTION:
Receive_Ex();
//OUTPUTS:
Receive_Ex.DONE;
Receive_Ex.EXEC;
Receive_Ex.ERROR;
Status := Receive_Ex.STATUS;  //If it is necessary to treat the error.
Receive_Ex.RX_RECEIVED;
Receive_Ex.RX_REMAINING;
Receive_Ex.RX_SILENCE;
```

#### 5.16.1.8. SERIAL_SET_CTRL

This block is used to write on the control signals (RTS and DTR), when they are available in the serial port. It can also set a busy condition for the transmission, through BREAK parameter and it can only be used if the modem signal is configured for RS232_MANUAL.


Figure 156: Block for Control Signals Writing

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |
| **RTS_VALUE** | BOOL | Value to be written on RTS signal. |
| **RTS_EN** | BOOL | Enables the RTS_VALUE parameter writing. |
| **DTR_VALUE** | BOOL | Value to be written on DTR signal. |
| **DTR_EN** | BOOL | Enables the DTR_VALUE parameter writing. |
| **BREAK** | BOOL | In case it's true, enables logic 0 (busy) in the transmission line. |

Table 148: SERIAL_SET_CTRL Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- FB_SET_CTRL_NOT_ALLOWED<br>- NOT_CONFIGURED |

Table 149: SERIAL_SET_CTRL Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Set_Control: SERIAL_SET_CTRL;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR

//INPUTS:
Set_Control.REQUEST := TRUE;
Set_Control.PORT := Port;
Set_Control.RTS_VALUE := FALSE;
Set_Control.RTS_EN := FALSE;
Set_Control.DTR_VALUE := FALSE;
Set_Control.DTR_EN := FALSE;
Set_Control.BREAK := FALSE;
//FUNCTION:
Set_Control();
//OUTPUTS:
Set_Control.DONE;
Set_Control.EXEC;
Set_Control.ERROR;
Status := Set_Control.STATUS;    //If it is necessary to treat the error.
```

#### 5.16.1.9. SERIAL_TX

This function block is used to transmit a data buffer through serial port and it is only finalized after all bytes were transmitted or after time-out (generating errors).


Figure 157: Block for Values Transmission by the Serial

| Input parameters | Type | Description |
|---|---|---|
| **REQUEST** | BOOL | This variable, when true, enables the function block use. |
| **PORT** | SERIAL_PORT | Select the serial port, as described in the SERIAL_PORT data type. |
| **TX_BUFFER_ POINTER** | POINTER TO BYTE | Pointer of a byte array to transmit the buffer values. |
| **TX_BUFFER_ LENGTH** | UINT | Specify the expected character number in the byte array to be transmitted (TX queue maximum size is 1024 characters). |

| Input parameters | Type | Description |
|---|---|---|
| **TX_TIMEOUT** | UINT | Specify the time-out to complete the transmission including the handshake phase. The specified value, in ms, must be positive and different than zero. |
| **DELAY_BEFORE_TX** | UINT | Specify the delay, in ms, between the function block call and the transmission beginning. This variable can be used in communications with some modems. |
| **CLEAR_RX_BEFORE_TX** | BOOL | When true, the RX queue and the UART FIFO RX are erased before the transmission beginning. This behavior is typical in half-duplex master/slave protocols. |

Table 150: SERIAL_TX Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **DONE** | BOOL | This variable is true when the block is completely executed. It is false otherwise. |
| **EXEC** | BOOL | This variable is true while the block is being executed. It is false otherwise. |
| **ERROR** | BOOL | This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion. |
| **STATUS** | SERIAL_STATUS | In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are:<br>- NO_ERROR<br>- ILLEGAL_SERIAL_PORT<br>- PORT_BUSY<br>- HW_ERROR_UART<br>- HW_ERROR_REMOTE<br>- ILLEGAL_TX_BUFF_LENGTH<br>- ILLEGAL_TIMEOUT<br>- CTS_TIMEOUT_ON<br>- CTS_TIMEOUT_OFF<br>- TX_TIMEOUT_ERROR<br>- NOT_CONFIGURED |
| **TX_TRANSMITTED** | UINT | Returns the transmitted byte number which must be equal to TX_BUFFER_LENGTH, but can be smaller in case some error has occurred during transmission. |

Table 151: SERIAL_TX Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Transmit: SERIAL_TX;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..9] OF BYTE := [0,1,2,3,4,5,6,7,8,9];
Status: SERIAL_STATUS;
END_VAR

//INPUTS:
Transmit.REQUEST := TRUE;
Transmit.PORT := Port;
Transmit.TX_BUFFER_POINTER := ADR(Buffer_Pointer);
Transmit.TX_BUFFER_LENGTH := 10;
Transmit.TX_TIMEOUT := 10000;
Transmit.DELAY_BEFORE_TX := 1000;
Transmit.CLEAR_RX_BEFORE_TX := TRUE;
//FUNCTION:
Transmit();
//OUTPUTS:
Transmit.DONE;
Transmit.EXEC;
Transmit.ERROR;
Status := Transmit.STATUS;    //If it is necessary to treat the error.
Transmit.TX_TRANSMITTED;
```

### 5.16.2. Timer Retain

The timer retain is a function block developed for applications as production line clocks, that need to store its value and restart the counting from the same point in case of power supply failure. The values stored by the function block, are only zero in case of a *Reset Cold*, *Reset Origin* or a new application *Download* (see Mastertool's user manual), when the counters keep working, even when the application is stopped (Stop Mode).

**ATTENTION**

It is important to stress that, for the correct functioning of the Timer Retain blocks, the variables must be declared as Retain (*VAR RETAIN*). It's also important to notice that in simulation mode, the Timer Retain function blocks do not run properly due to need the Nexto CPU for correct behavior.

The three blocks already available in Mastertool *NextoStandard* library are described below (for the library insertion proceeding, see programming manual, section Library).

#### 5.16.2.1. TOF_RET

The function block *TOF_RET* implements a time delay to disable an output. When the input *IN* has its state changed from (TRUE) to (FALSE), or a falling edge, the specified time *PT* will be counted and the *Q* output will be driven to (FALSE) at the end of it. When the input *IN* is in logic level 1 (TRUE), the output *Q* remain in the same state (TRUE), even if this happened in the middle of the counting process. The *PT* time can be changed during the counting as the block assumes the new value if the counting hasn't finished. Figure 158 depicts the *TOF_RET* block and Figure 159 shows its graphic behavior.


Figure 158: TOF_RET Block

| Input parameters | Type | Description |
|---|---|---|
| **IN** | BOOL | This variable, when receives a falling edge, enables the block counting. |
| **PT** | TIME | This variable specifies the block counting limit (time delay). |

Table 152: TOF_RET Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **Q** | BOOL | This variable executes a falling edge as the PT variable (time delay) reaches its maximum value. |
| **ET** | TIME | This variable shows the current time delay. |

Table 153: TOF_RET Output Parameters

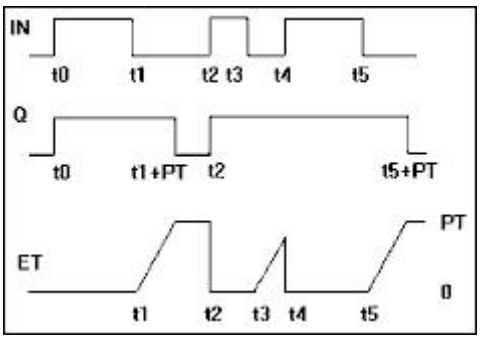



Figure 159: TOF_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL := TRUE;
TOF_RET : TOF_RET;
END_VAR

// When bStart=FALSE starts counting
TOF_RET( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TOF_RET.Q = FALSE) THEN
bStart := TRUE;
END_IF
```

#### 5.16.2.2. TON_RET

The *TON_RET* implements a time delay to enable an output. When the input *IN* has its state changed from (FALSE) to (TRUE), or a rising edge, the specified time *PT* will be counted and the *Q* output will be driven to (TRUE) at the end of it. When the input *IN* is in logic level 0 (FALSE), the output *Q* remain in the same state (FALSE), even if it happens in the middle of the counting process. The *PT* time can be changed during the counting as the block assumes the new value if the counting hasn't finished. Figure 160 depicts the *TON_RET* block and Figure 161 shows its graphic behavior.


Figure 160: TON_RET Function Block

| Input parameters | Type | Description |
|---|---|---|
| **IN** | BOOL | This variable, when receives a rising edge, enables the function block counting. |
| **PT** | TIME | This variable specifies the block counting limit (time delay). |

Table 154: TON_RET Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **Q** | BOOL | This variable executes a rising edge as the PT variable (time delay) reaches its maximum value. |
| **ET** | TIME | This variable shows the current time delay. |

Table 155: TON_RET Output Parameters

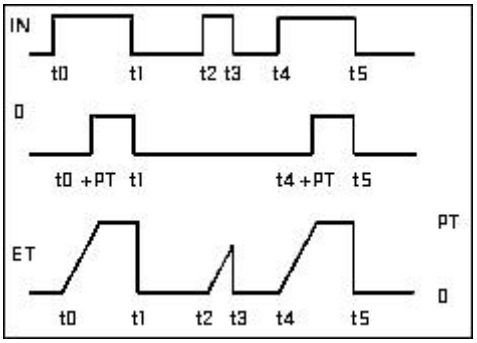



Figure 161: TON_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL;
TON_RET : TON_RET;
END_VAR

// Quando bStart=TRUE starts counting
TON_RET( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TON_RET.Q = TRUE) THEN
bStart := FALSE;
END_IF
```

#### 5.16.2.3. TP_RET

The *TP_RET* function block works as a trigger. The timer which starts when the *IN* input has its state changed from (FALSE) to (TRUE), that is, a rising edge, it is increased until the *PT* time limit is reached. During the counting, the *Q* output is (TRUE), otherwise it is (FALSE). The *PT* time can be changed during the counting as the block assumes the new value if the counting has not finished. Figure 162 depicts the *TP_RET* and Figure 163 shows its graphic behavior.


Figure 162: TP_RET Function Block

| Input parameters | Type | Description |
|---|---|---|
| **IN** | BOOL | This variable, when receives a rising edge, enables the function block counting. |
| **PT** | TIME | This variable specifies the function block counting limit (time delay). |

Table 156: TP_RET Input Parameters

| Output parameters | Type | Description |
|---|---|---|
| **Q** | BOOL | This variable is true during the counting, otherwise is false. |
| **ET** | TIME | This variable shows the current time delay. |

Table 157: TP_RET Output Parameters

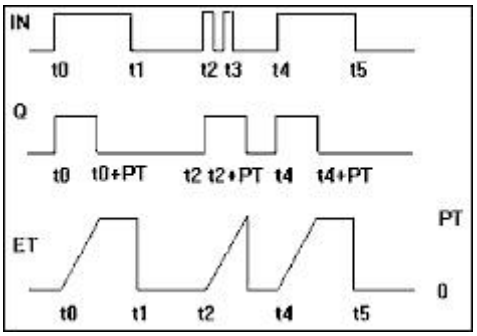



Figure 163: TP_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL;
TP_RET : TP_RET;
END_VAR

// Configure TP_RET
TP_RET( IN := bStart,
PT := T#20S);

bStart := FALSE;

// Actions executed during the counting
IF (TP_RET.Q = TRUE) THEN
// Executes while the counter is activated
ELSE
// Executes when the counter is deactivated
END_IF
```

## 5.17. FTP Server

FTP (File Transfer Protocol) is a protocol that allows files to be transferred between devices. It operates in the client-server mode, where the CPU becomes a FTP Server, storing files that FTP Clients can access for transfer, download, and upload.

The controlling FTP connection is a TCP connection established through port 21 of the FTP Server. Through port 21, the Client and Server exchange commands and responses to manage the file transfer session.

Through the FTP protocol, the FTP Client can read and write files that are stored either in the internal memory of the CPU or in external memory (memory card, for example) if present in the architecture. The maximum file size that can be transferred varies according to the amount of memory available between internal and external memory. When the memory limit is reached, the transfer will stop, and if the file has not been transferred completely, it will become a corrupted file.

**ATTENTION**

Downloading and uploading large files through FTP, both from internal and external memory, can affect the performance of the CPU considerably.

**ATTENTION**

The FTP server does not support communication through Windows File Explorer, so an FTP Client software is required to access it.

### 5.17.1. Configuration

The FTP configuration is done through a dedicated section located in the *Management* tab of the controller's System Web Page, as shown below:

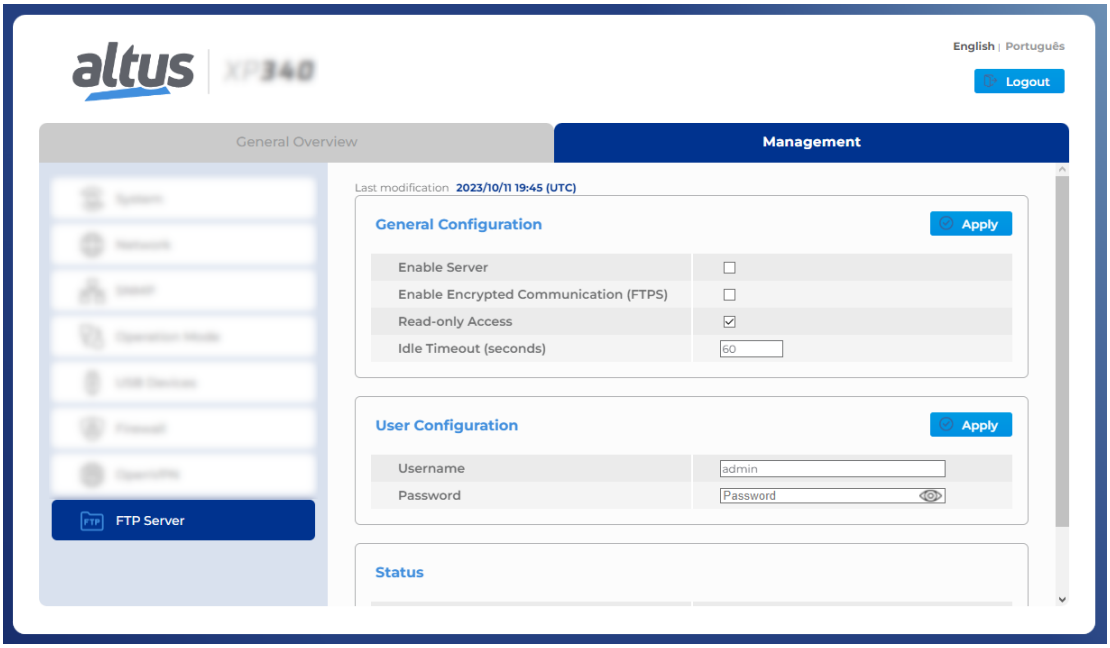



Figure 164: FTP Server Configuration Screen

FTP is a separate feature from Mastertool, meaning it does not require any interaction with Mastertool. The settings applied on the *FTP Server* section take effect when confirmed through the *Apply* button and are automatically saved in the controller. As long as the functionality is enabled, it will resume operation even after the device is restarted.

#### 5.17.1.1. General Configuration

##### *5.17.1.1.1. Enable Server*

Allows to activate or deactivate the functionality. When the FTP server is enabled, the settings made through the System Web Page are applied to the configuration files. That means the server is available according to the configuration made. If FTP is disabled, the configuration is still stored, but the service cannot be used.

##### *5.17.1.1.2. Enable Security*

Enables and disables encrypted communication. This communication is done through Explicit FTPS, also known as FTPES. This is a secure extension of the FTP protocol, which adds a layer of encryption to the transfer.

In this type of communication, when a client-server connection is established, instead of immediately starting the data transfer, the FTP Client sends an AUTH SSL command to request a secure connection. Then, when enabling encrypted communication, the UCP generates a self-signed certificate to guarantee communication using the SSL (Secure Sockets Layer) protocol.

When the FTP Client performs the authenticated connection, the certificate and its respective information will be displayed, allowing Explicit FTPS communication (FTP over SSL) to be established through port 21.

**ATTENTION**

The FTPES certificate is valid for one year from the date it was created. To generate a new certificate, simply reapply the configuration.

##### *5.17.1.1.3. Read-only Access*

Enables and disables limiting write and read access to the Server. By default, the parameter is enabled.

When the parameter is enabled, the FTP Client has read-only access without the possibility to add or remove any files from the FTP Server. In this case, the only operation allowed is uploading files, that is, transferring them from the Server to the Client. When the parameter is disabled, all operations can be performed.

#### 5.17.1.1.4. *Idle Timeout (Seconds)*

Sets the maximum time the connection will be maintained before the FTP Server closes it due to inactivity. In other words, once the connection to a FTP Client is opened, if there is no activity after the Idle Timeout, the connection to the Client is closed by the Server.

The parameter can be configured with values between 10 to 60 seconds, the default being 60 seconds.

### 5.17.1.2. User Configuration

#### 5.17.1.2.1. *Username*

*User* or *Username* required for the Client to connect to the Server.

If a new configuration is made, the previous one is removed. In other words, there is only one FTP user, but this one can be used for multiple connections.

#### 5.17.1.2.2. *Password*

Manages the authentication key so the FTP Client can connect to the FTP Server.

There is no password recovery so, if the password has been lost, it is necessary to add a new user configuration.

| Configurable Item | Minimum Size | Maximum Size | Allowed Characters | Default |
|---|---|---|---|---|
| **Username** | 4 | 30 | [a-z][A-Z][0-9]@$*_ | admin |
| **Password** | 4 | 30 | [a-z][A-Z][0-9]@$*_ | admin |

Table 158: FTP User Settings

### 5.17.1.3. Status

#### 5.17.1.3.1. *Current State*

Displays the current status of the FTP Server. The possible states are *"Running"*, *"Not Running"*, and *"Restarting Service"*. Each configuration applied will restart the service.

#### 5.17.1.3.2. *Connected Clients*

Shows the user the current number of active connections.

The FTP Server accepts a maximum of two active connections simultaneously. Some FTP Clients, such as *Filezilla*, use a feature called Multithread File Transfer to improve the efficiency and speed of the transfer. This feature allows the FTP Client to open more than one connection for the transfer of a file. Consequently, when using an FTP Client of this type, just one Client already counts as two active connections on the Server.

Other FTP Clients, such as the command terminal, only use one active connection, allowing two FTP Clients to connect to the Server simultaneously.

## 5.18. Memory Card

Among different storage modes, this Nexto Series CPU model allows the user to use a memory card, defined according to the characteristics of the Memory Card Interface section, which serves to store files.

When the card is inserted in the CPU and it presents a file system different from FAT32, it starts blinking the SD Card LED. Once the memory card is mounted, the CPU will read its data, leaving access to the memory card slower in the first few minutes. This procedure is done only once when the card is inserted or in case of the CPU reset.

### 5.18.1. General Storage

Memory Card devices can be used to expand the controller's flash memory to store a large amount of data, such as in data logger applications. To use a memory card device, simply connect it to the SD Card slot. After a few seconds, when the device is properly detected and mounted, the SD Card LED will turn on and the device information will appear in the section *Memory Card Devices* located at the tab *PLC Management* of the controller's diagnostics web page as shown below:

The information shown on the status section of this page is also available in the symbolic variables diagnostics structure (see Section Diagnostics via Variables).

**ATTENTION**

The Memory Card device must be formatted as a FAT32 volume. Other file system formats are not supported.

The device can be ejected through a long press in the PLC button or via web page.

After the device is properly detected and mounted, a new folder called *MemoryCard* will appear on the controller's memory as shown on the picture below:

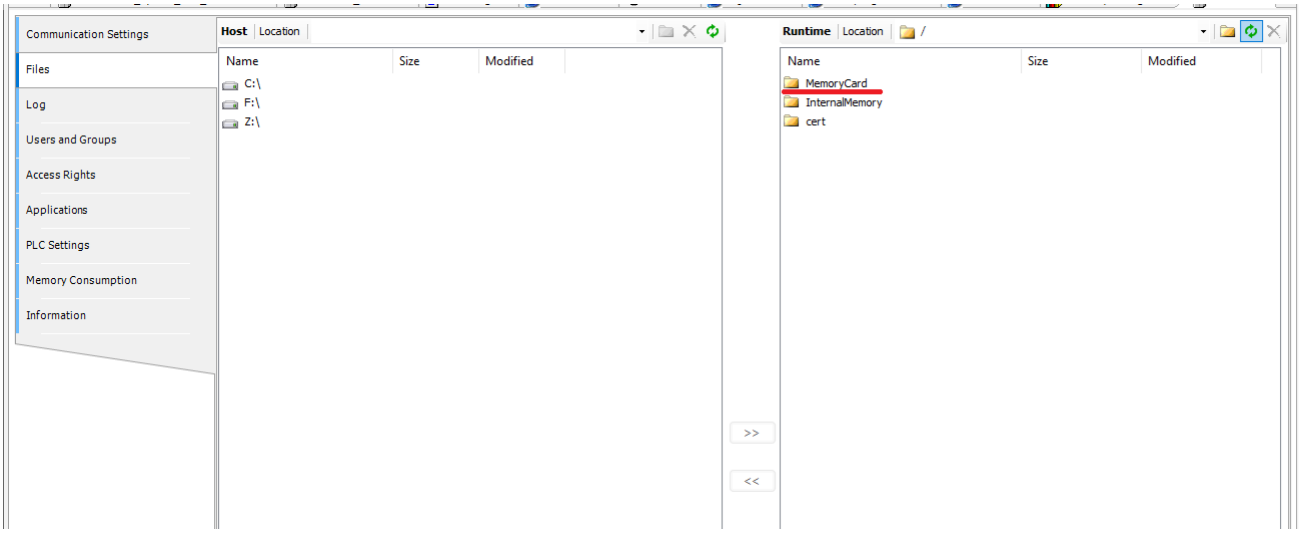



Figure 165: Memory Card Storage Folder

### 5.18.2. Memory Card Configuration

A web page was developed for managing the memory card. Among the features offered are formatting, the option to unmount and remove the card, and the ability to enable and disable the card interface. These settings were developed in the "Memory Card" section of the CPU webpage, under the "Management" tab. In addition to the settings, information about the device's current status, total and free storage space, both measured in *kB*, are also displayed. The image below shows the home page, with the interface enabled and no devices connected.

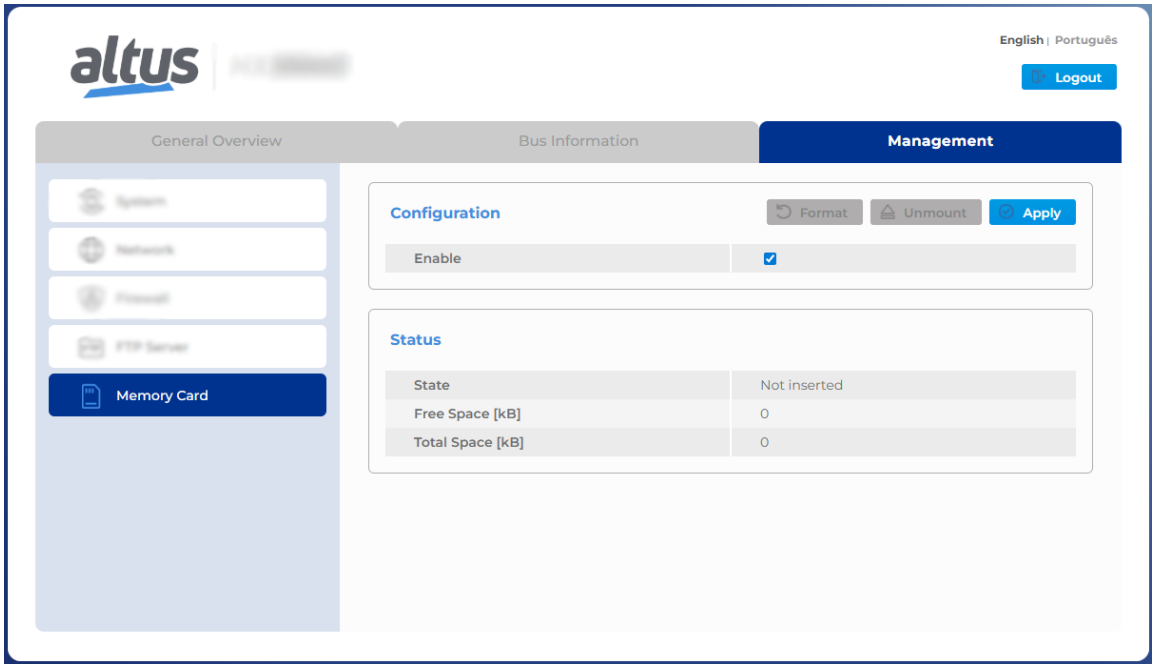



Figure 166: Memory Card Home Page

As long as there is no memory card inserted in the CPU, the *Format* and *Unmount* buttons remain locked for use. The *Status* table indicates the *"Not Inserted"* state. When a memory card is inserted into the CPU, the *Format* button is enabled for use. After the card is mounted, the *Unmount* button also becomes available for use. When inserting a memory card into the CPU, it may take a few moments for the card to be mounted and the information updated on the web page. The image below shows the web page when a card is connected and mounted in the CPU.

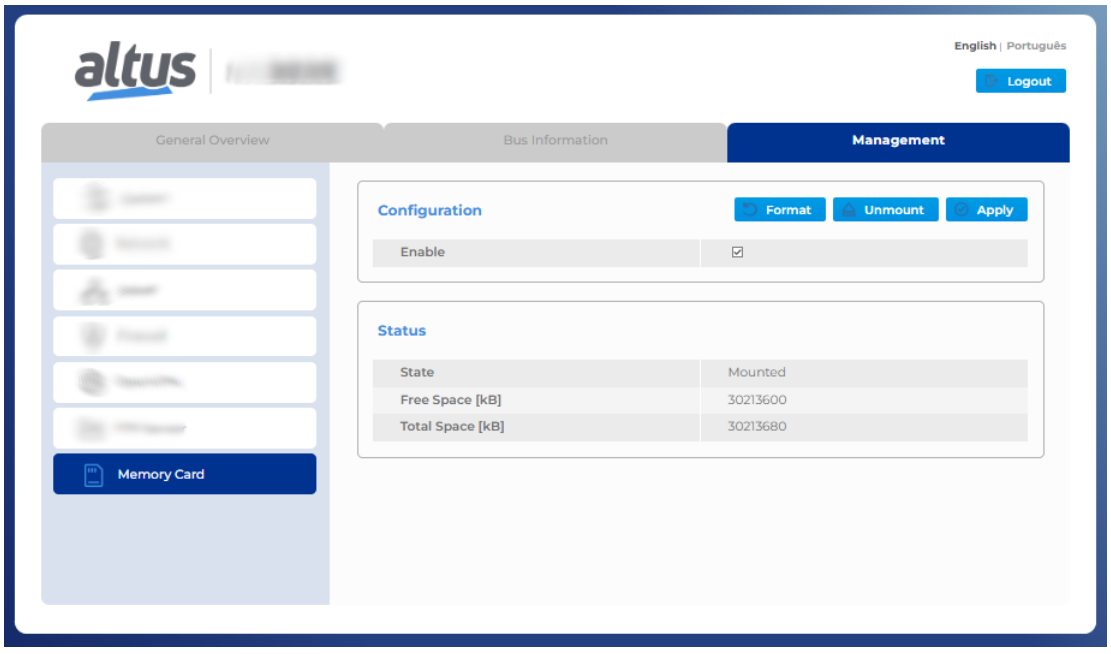



Figure 167: Memory Card Device Mounted

**ATTENTION**

For the memory card to properly mount, there must be at least *1Kb* of free space available on the CPU. If the card is inserted and there is not enough free space, the card will not mount and the web page will display the message *"Unmounted"*

#### 5.18.2.1. Format Not Supported

If the device has a file system that is not supported by the CPU, the state *"Format not supported"* will appear when the device is inserted, as shown in the figure below. In this case, the device cannot be used until it is formatted with a supported file system.

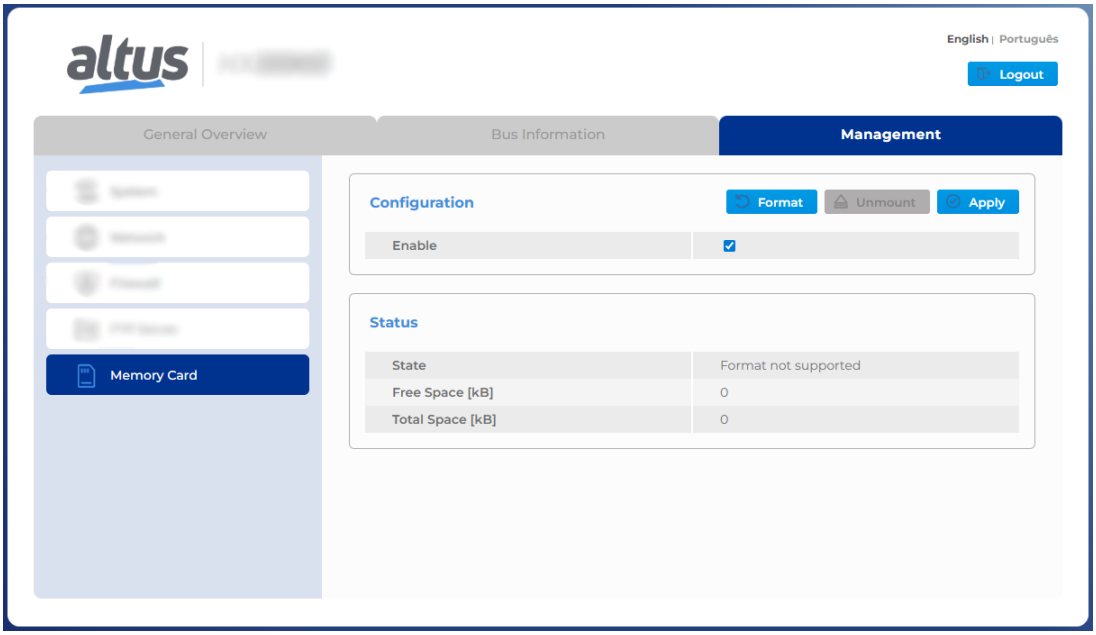



Figure 168: Format Not Supported Message

To format the device, the *Format* button may be used, as shown in Formatting the Memory Card.

#### 5.18.2.2. Formatting the Memory Card

To format the device, use the *Format* button. When you click, a *pop-up* style message will appear, asking you to confirm the operation. The image below presents this message.

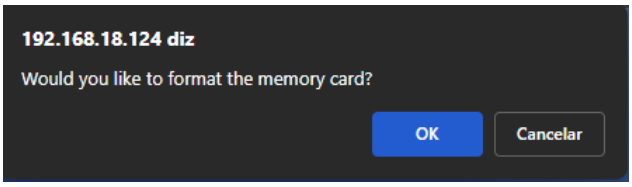



Figure 169: Format Confirmation Message

Upon confirming with the *OK* button, the operation begins, after which all configurations are blocked. The *Format*, *Unmount* and *Apply* buttons, as well as the checkbox, become unavailable during formatting. The formatting process is indicated in the *Status* table, with the *State* value changed to *"Formatting..."*, as shown in the figure below.

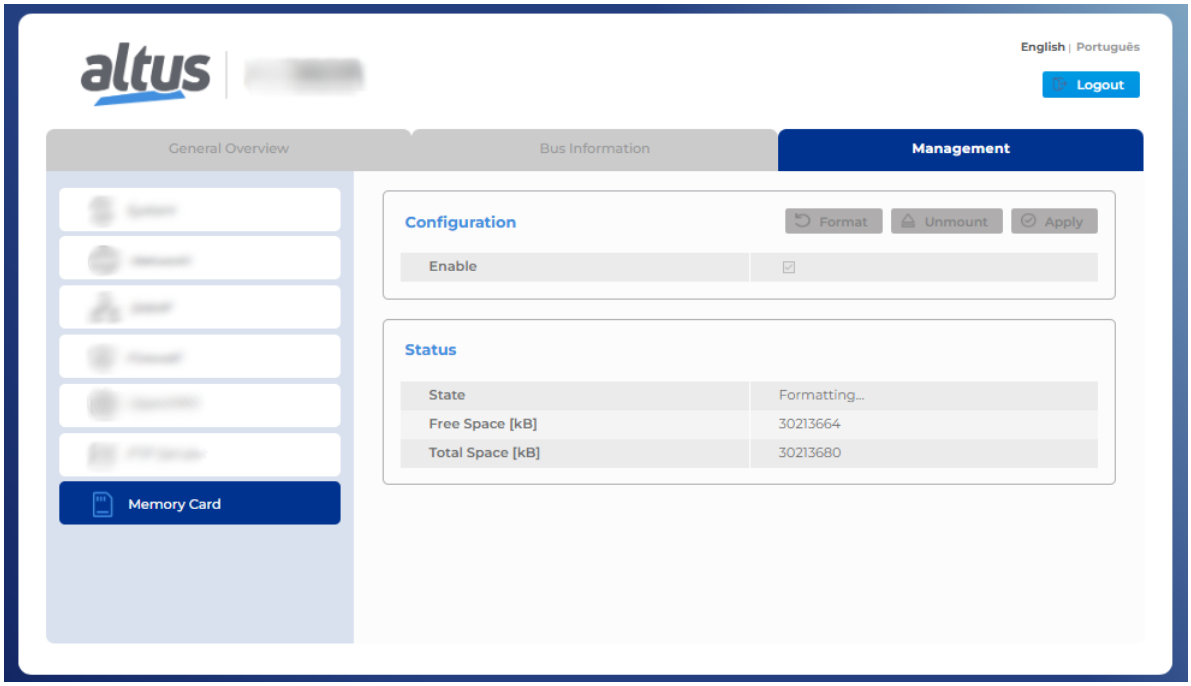



Figure 170: Formatting Memory Card

At the end of the formatting process, a message is displayed indicating that the operation has been completed on the device. The following figure shows this message.

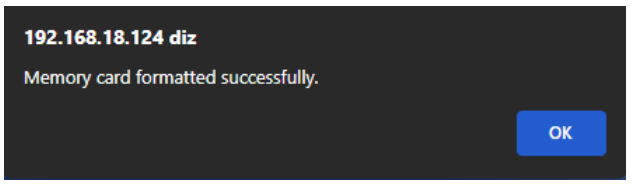



Figure 171: Formatting Complete Message

Upon completion of the operation, the web page returns to its initial state, unlocking all buttons, as well as the checkbox, as shown in figure Memory Card Device Mounted.

#### 5.18.2.3. Unmounting the Memory Card

To unmount and remove the device, click the *Unmount* button. A confirmation *pop-up* will appear asking you to confirm the operation. The image below shows this message.

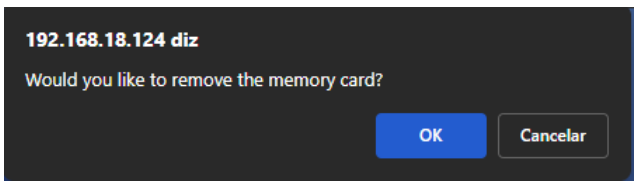



Figure 172: Confirmation Message to Unmount

Upon confirming with the *OK* button, the operation begins, after which all configurations are blocked. The *Format*, *Unmount* and *Apply* buttons, as well as the checkbox, become unavailable during unmounting operation. The unmounting process is indicated in the *Status* table, with the *State* value changed to *"Unmounting..."*, as shown in the figure below.

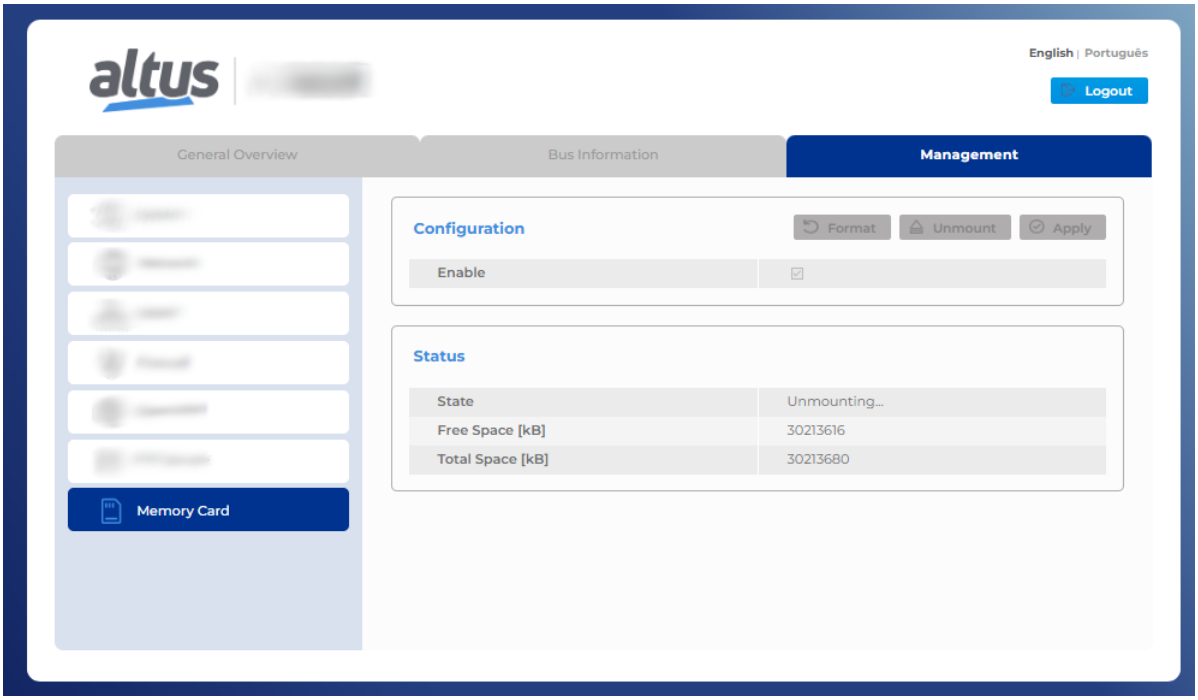



Figure 173: Unmounting Memory Card

At the end of the unmounting process, a message is displayed indicating that the operation on the device has been completed. The following figure shows this message.

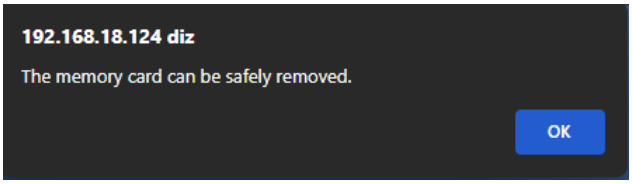



Figure 174: Unmount Complete Message

Upon completion of the operation, the *Format* and *Apply* buttons, as well as the checkbox, become available for use. The *Unmount* button remains locked because the card has already been unmounted. The *Status* table displays the data for an unmounted card, as can be seen in the image below:

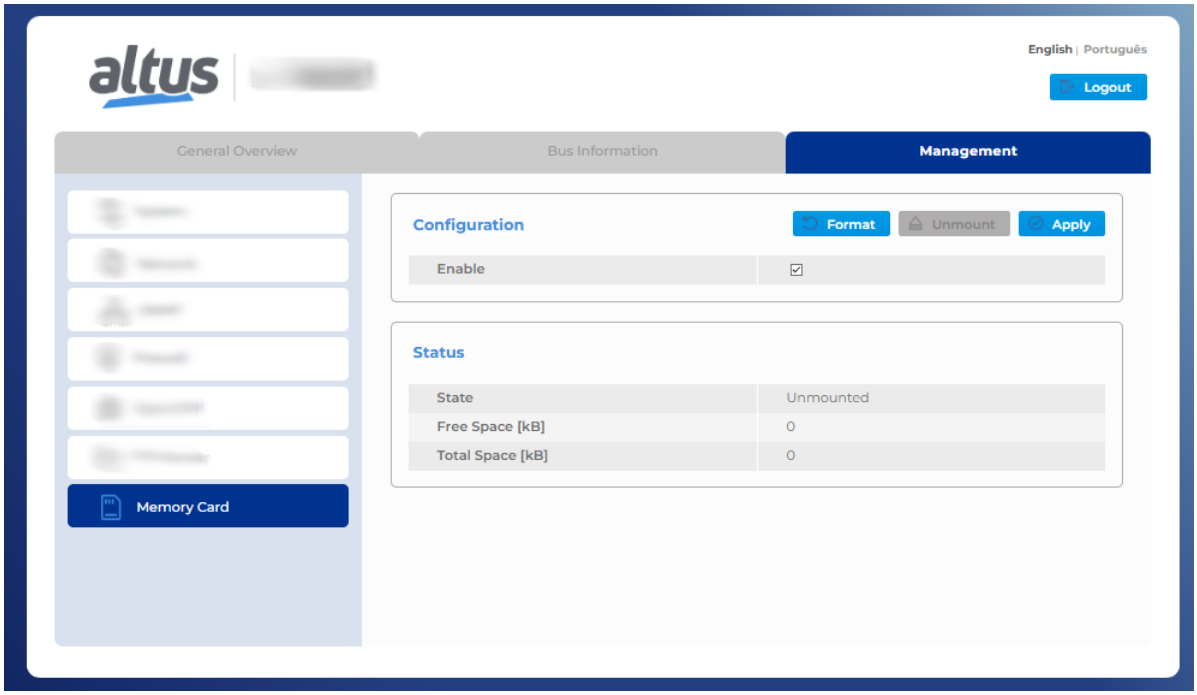



Figure 175: Memory Card Unmounted

#### 5.18.2.4. Memory Card Interface Management

A setting was developed on the card's web page to enable and disable the memory card interface, this functionality is part of the level one cybersecurity requirements according to IEC 62443. To enable, check the **Enable** checkbox in the **Configuration** table. Then use the **Apply** button to submit the new configuration. To disable, uncheck the **Enable** checkbox and use the same button to apply the setting. Clicking the **Apply** button will display a *pop-up* message asking you to confirm the operation. The image below shows the message.

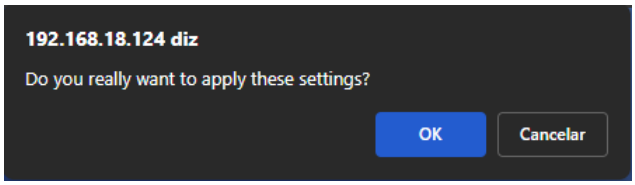



Figure 176: Confirmation Message to Apply Configuration

After confirming by clicking the **OK** button, the new configuration is sent to the CPU. If the interface has been enabled, the information displayed on the web page depends on whether or not a memory card is inserted for mounting. If there is no device, the page will display the information shown in figure Memory Card Home Page. When a device is connected, the information displayed will be as shown in figure Memory Card Device Mounted, after the card has been properly mounted.

If the new configuration disables the interface, the information displayed will be as shown in the image below.

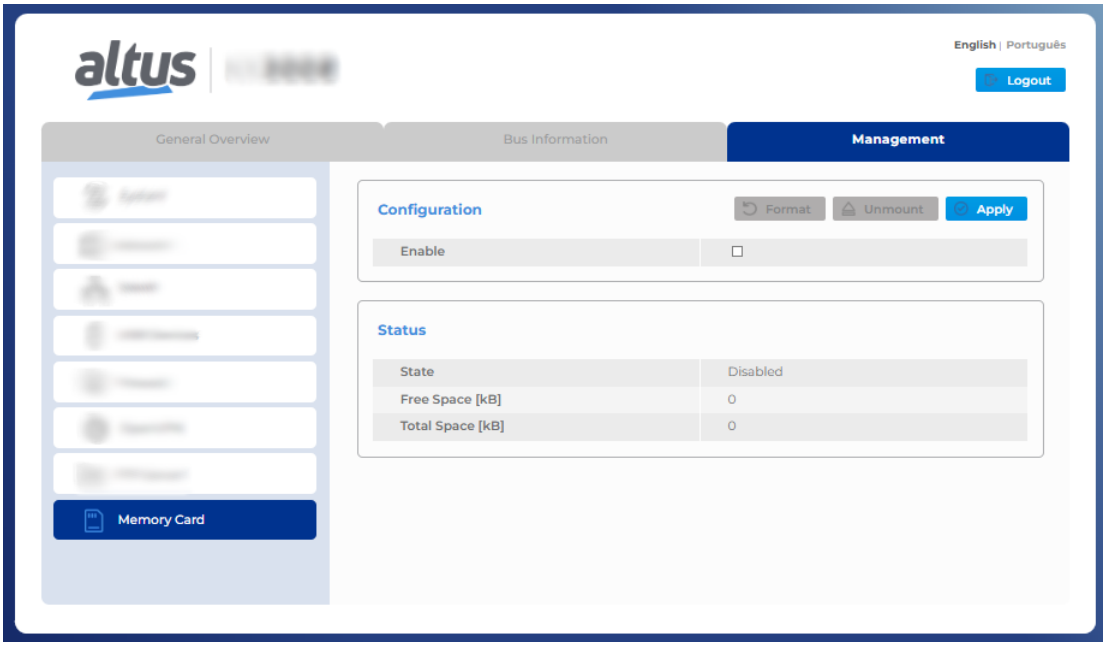



Figure 177: Memory Card Disabled Interface

**ATTENTION**

If the memory card deactivation is in effect, the MemoryCard folder will not be mounted.

#### 5.18.2.5. Memory Card Interface Management by Application

To facilitate the management of the memory card interface, a function was developed that can be called directly by the user's application code. The **SetMemCardState** function was implemented within the **NextoStandard** library. The image below shows the library information, as presented in the *Library Manager*.

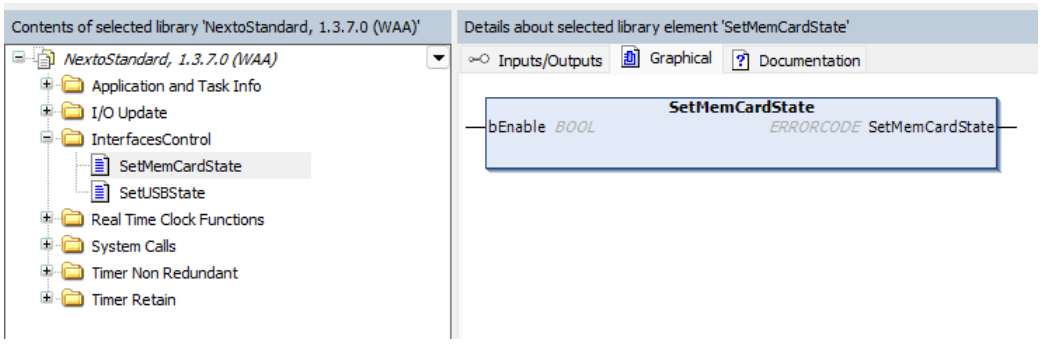



Figure 178: SetMemCardState information in Library Manager

The function has an input variable of type *bool*, **bEnable**, which receives the value to enable or disable the card interface. The function has three return values: *NoError* on success, *SetMemCardStateFail* on failure, or *ImportFunctionNotFound* if the function is not supported. The image below shows a basic example of variable declaration and function call.

```
PROGRAM UserPrg
VAR
  bSetMemoryCardInterfaceState : BOOL;
  stErrorCode : NextoStandard.ERRORCODE;
END_VAR
-------------------------------------------------------------------------
```

```
// Function call example to configurate memory card's interface
stErrorCode:= NextoStandard.SetMemCardState(bEnable :=
    bSetMemoryCardInterfaceState);
```

**ATTENTION**

The function executes the command to set the desired value for the memory card interface. It is neither necessary nor recommended that the function be called cyclically.

## 5.19. Firewall

### 5.19.1. Introduction

The Firewall is designed to increase the security of the device while it is in use. The main function of the Firewall is to filter data packets coming into and leaving the device. The implemented filter uses information from each data packet to decide whether that packet is allowed. The main parameters used are the input/output interfaces, the port, the transport layer protocol, and the source and destination addresses.

**ATTENTION**

When the device is turned on with its button pressed, the Firewall is disabled and its rules are not applied.

### 5.19.2. Configuration

Firewall configuration is done through a dedicated section located in the *Management* tab of the controller's System Web Page, as shown below:

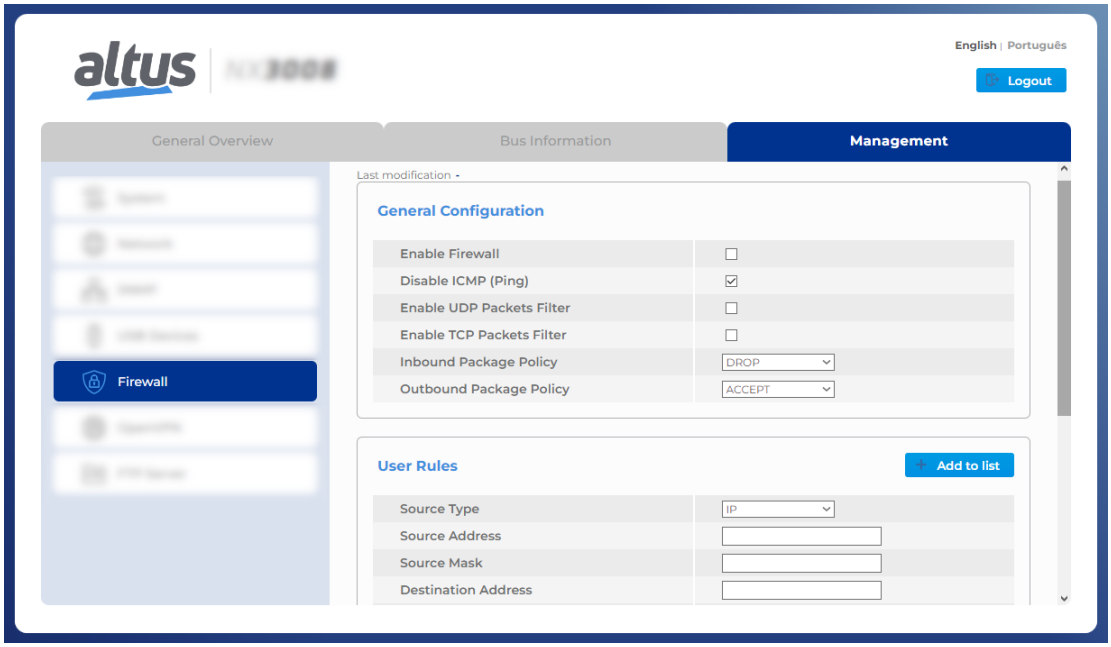



Figure 179: Firewall Configuration Screen

The Firewall is a separate feature from Mastertool, that is it doesn't require any interaction with Mastertool. Settings applied on the *Firewall* section take effect when confirmed with the *Apply* button, and are automatically saved in the controller. If the feature is enabled, it will operate again even after rebooting the device.

The following sections describe the possible settings for the Firewall, divided according to the tables of the *Firewall* section.

### 5.19.3. General Configuration

The image below shows all the settings in the *General Configuration* table:

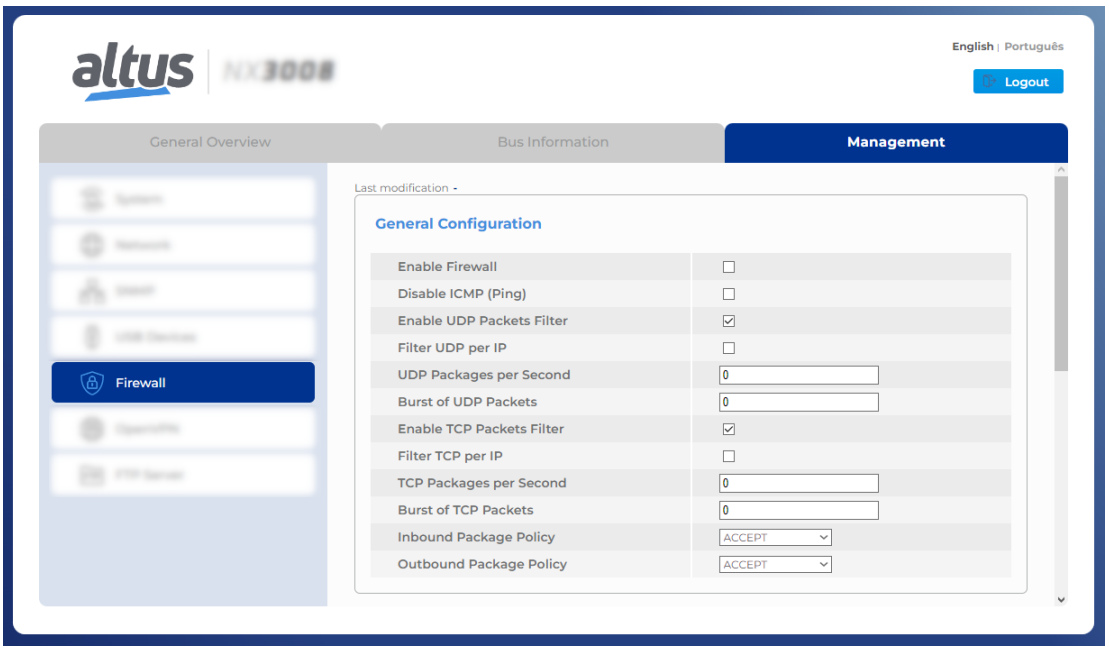



Figure 180: Firewall General Settings Table

This table expands dynamically by selecting the options to enable UDP and TCP packet filters, revealing all the items that can be configured. The first item in this table, *Enable Firewall*, is used to enable and disable this functionality. When the Firewall is enabled, the web page settings, when submitted to the device, will be applied to the configuration files, and then the Firewall will filter what has been configured. If the Firewall is disabled, the configuration that was made is stored, but the rules are not applied in the controller.

The field *Disable ICMP (Ping)* enables or disables protection against the ICMP protocol. When protection is enabled, the controller will not respond to *Ping* requests, since it will drop packets that use the ICMP protocol. When disabled, the operation of the device for *Ping* responses maintains its normal behavior.

When enabled, the fields that enable UDP and TCP packet filtering, filter these protocols according to the limits configured in their respective fields. The packet filtering rule works like this: for a packet to be accepted, there must be "credits" available, and one credit is used to accept a data packet.

The setting of the field *Burst of XXX Packets* sets the initial value of packages (credits), which will be accepted. In this way, it is possible to set an overflow limit for these packets, where if there is a large flow of packets, only the configured amount will be accepted. The *XXX Packages per Second* field sets how many credits that rule will earn per second. For example, if the value is 5, each second, the rule will receive five new credits, so it will be able to accept five more packages. The limitation for this increment in the number of credits is the configuration of *Burst of XXX Packets* itself, and the limit set here is not exceeded, even with the increment of packets every second. These settings are applied as a *stock*, where upon receiving a data packet, it is first checked if there is any credit available in the stock, and then a decision is made whether or not to accept the package. If the packet is accepted in this quantity filter, it is forwarded to the filter of the other firewall rules.

The setting *Filter XXX per IP* causes the rule to differentiate the source addresses of each packet and apply the packet per second and packet overflow filters individually to each IP address. So, going back to the previous example, it can be considered that each source address has its *stock* of credits, and one address cannot use the credits that are in the *stock* reserved for others.

**ATTENTION**

Negative values are not allowed for the *XXX Packages per Second* and *Burst of XXX Packets* fields. If negative values are set, when applying the settings an error message will appear on the screen indicating the field that had a conflict. If the filter is enabled, but the values in these fields are left at 0, the filter is not applied.

The settings in this table are applied with the *Apply* button that appears in figure 182.

The fields for selecting both incoming and outgoing policies have options to accept and drop. If the Firewall is active, when data packets arrive, all the rules that have been configured are checked, and then the policy configured for these packets is applied, whether *Accept* or *Drop*. So if an accept policy is set, *Accept*, all packets that do not match any configured rule will be accepted by the firewall, and if a reject policy is set, *Drop*, they would all be dropped.

At the top of the table, the *Export* and *Import* buttons are available. Using the *Export* button allows you to download a file with the extension *.firewall* containing all applied Firewall configurations. The *Import* button allows importing a *.firewall* file previously exported from this or another CPU of the same model, containing the Firewall configurations. When importing a file, the rules contained within it will be validated and configured in the *General Settings* fields and in the *User Rules* table. After importing, the rules will be applied by clicking the *Apply* button.

**ATTENTION**

The exported file contains only the rules that have been effectively applied to the device. Any configuration or rule added in the web page list but not applied to the CPU will not be included in the generated file.

**ATTENTION**

Any modification to the *.firewall* file, such as manual editing or changing the extension, may prevent some rules from being applied correctly. Always use files generated by the *Export* button of the same CPU, or from another of the same model, when importing Firewall configurations.

### 5.19.4. User Rules

The *User Rules* table was created to allow greater control over the firewall's rule settings. With it, you can configure different rules dynamically and with more precise filters.

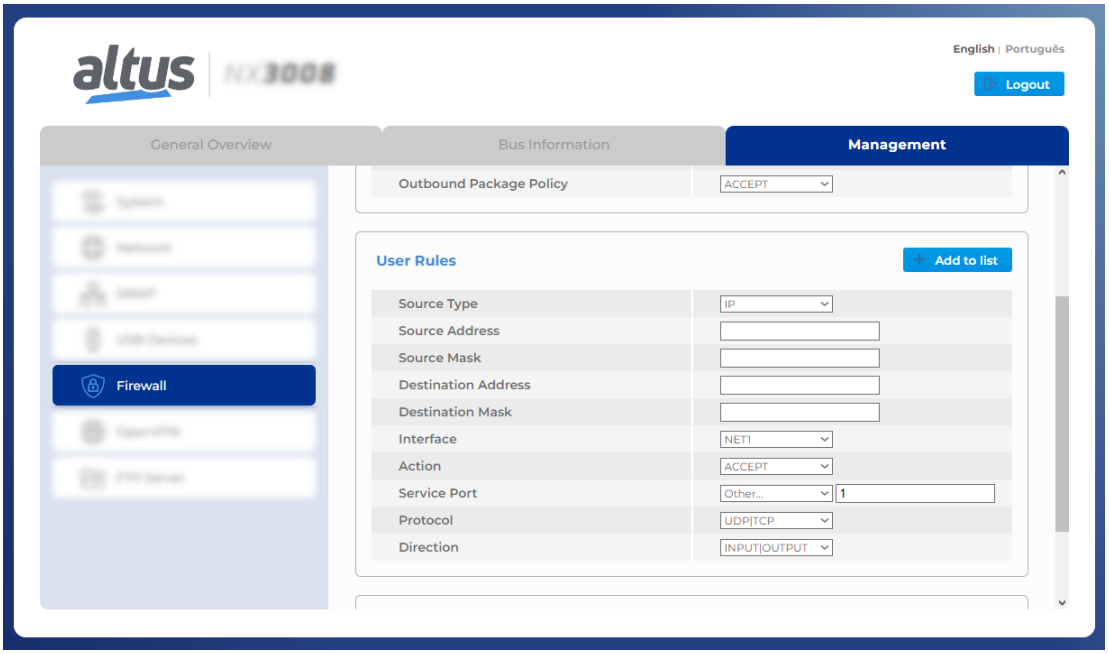



Figure 181: Firewall User Rules Configuration Table

This table changes its format according to the selected *Source Type*, which can be IP or MAC. When the type is *IP*, the table has the items shown in figure above, but when the type is selected as *MAC*, the source and destination mask fields disappear, as well as the *Destination Address* field. The item *Source Address* now accepts a MAC address as input in a format of six groups of two hexadecimal digits separated by colons, e.g. "1A:2B:3C:4D:5E:6F". Also, an address-based *MAC* rule can only be configured as an input rule. In other words, the *Direction* field will be forced with the value *INPUT*.

With the *Source Address* and *Destination Address* fields, you can enter the addresses that will be configured for that specific rule, and using the *Source Mask* and *Destination Mask* fields, you can configure a network range for this rule. If you only configure the address, only the address will be assigned to the rule but with different netmask configurations, you can get IP groups of various sizes to be applied to the rule.

Interface configuration makes it possible to individually select each physical or virtual interface available to the controller. Based on which interface you select for a given rule, only data packets entering or leaving the interface will be filtered by the Firewall. If you use the option *Any*, this rule will have no interface filter. So the filtering rule will be valid for all available interfaces.

The *Action* field has three configuration options: *ACCEPT*, *DROP*, and *REJECT*. The action sets up what should be done with the package whose characteristics match the rule applied. If the chosen action is *ACCEPT*, the data packet having characteristics according to the rule will be accepted. If it is *DROP*, the packet will be dropped, and no reply will be sent to the sender of the package. Finally, if it is set to *REJECT*, the packet will be rejected, and a reply will be sent to the sender, stating that the requested *host* is inaccessible.

The *Service Port* field, is used to indicate which ports will be configured in this rule. All service ports that have a certain protocol or communication *standard* for the controller, such as the MODBUS protocol that has the standard port 502, are available with the service name and port used next to it. Thus, if you configure the rule for the MODBUS protocol, port 502 will be applied if you configure the rule for the WebVisu service, port 8080 will be applied, and so on for the other protocols listed in the checkbox.

This field also has two other settings, which are *Any* and *Other*. When you select the *Any* option, the rule is applied to all service ports except port 80 then two rules are created using the following port ranges: *1:79* and *81:65535*. If you select the *Other* option, a text box appears in which you can configure the port you want, except for port 80. To configure a port, you can type its number in the text box, but if you want to add more than a single port, you must use the *"&"* separator, and if you want to insert a range of ports, simply enter the start and end port using the separator *":"*.

Example of configuring ports 120, 144, and the range 1300 to 1450 in the same field: *120 & 144 & 1300:1450*.

This field doesn't accept values outside the range 1:65535, port 80, or port repeats.

The HTTP port 80 can only be set by selecting it from the list of known protocols and cannot be applied to the NET 1 interface. So if the HTTP protocol is chosen, the Interface field *NET 1* and *Any* won't be selectable.

In the *Protocol* field, you can select between UDP, TCP, and UDP|TCP protocols. If you select the UDP|TCP option, two rules will be created on the firewall, one for each transport protocol.

In the *Direction* field, you can select between *INPUT*, *OUTPUT*, and *INPUT\OUTPUT*. These options cause the rule to be applied to packets arriving at the device, option *INPUT*, or leaving it, option *OUTPUT*. If the joint option is configured, two rules will be created, one with each direction option.

The figure below demonstrates how a rule is applied:

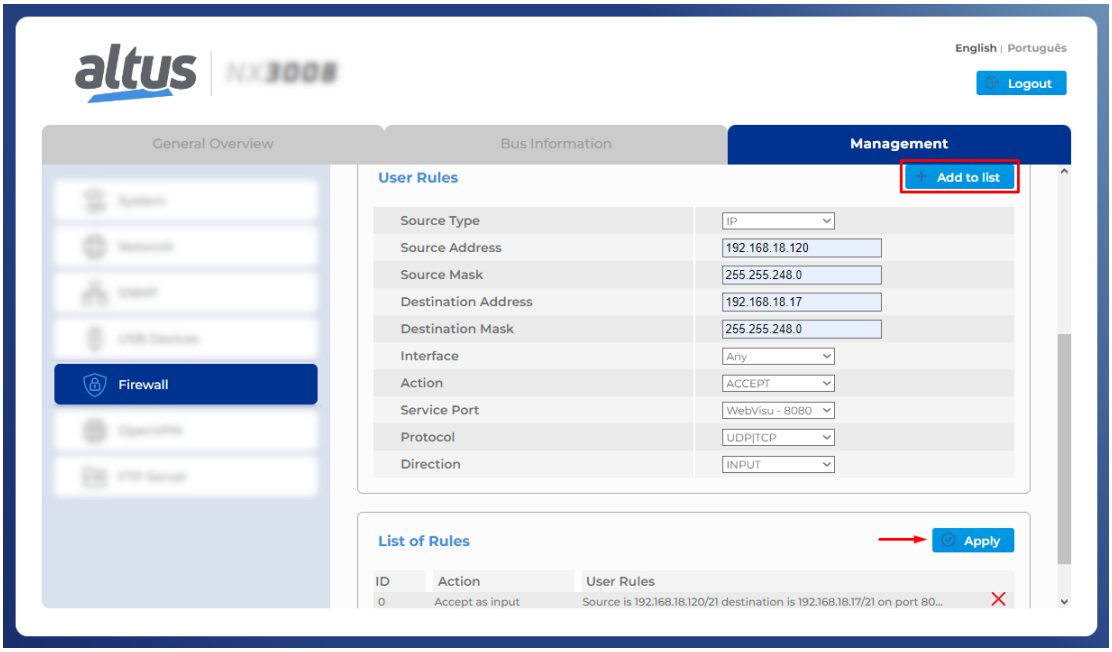



Figure 182: Firewall User Rules Enforcement Table

After filling in the fields as you wish to configure the firewall rule, you must click the *Add to list* button. By doing this, all the settings will be analyzed to check if there are invalid values or if there is any duplicate rule. It's impossible to add two rules with the same address, mask, interface, port, and direction parameters. If a conflict is found, a message will be displayed indicating the field that contains an invalid setting or the ID number of the rule in the table whose settings caused the conflict with the newly configured one.

After all, parameters are checked, the rule will be added to the list below the configuration table. This list expands automatically as rules are added or deleted. If you want to exclude a rule from the list, you can place the mouse over the one you want to exclude. When you do this, a red *X* button will appear on the right, as shown in the previous figure. By clicking it, the rule will be deleted from the table.

When adding new rules, or deleting an existing one, in the rules table, the *Apply* button below must be clicked for the configuration to be applied to the device.

**ATTENTION**

During the application of firewall rules, there may be a momentary instability in Ethernet communication.

## 5.20. OpenVPN

### 5.20.1. Introduction

VPN (Virtual Private Network), used for surfing unsecured networks, transmitting data, or simply accessing the Internet with a high level of security and privacy. The VPN virtual network can be understood as a tunnel in which information travels securely, protected by security certificates and keys. OpenVPN is an *open-source* service, which means that it is free to use and distribute, and its source code is open for modifications if needed.

The main purpose of a VPN is to communicate securely over an unsecured network. To make this possible, data encryption is used based on certificates and keys generated using TLS, Transport Layer Security, a protocol that performs 256-bit encryption, one of the most secure.

To perform the configuration of the OpenVPN client or server, the OpenVPN page was created in the *Management* tab of the CPU's System Web Page. As shown in the figure below.

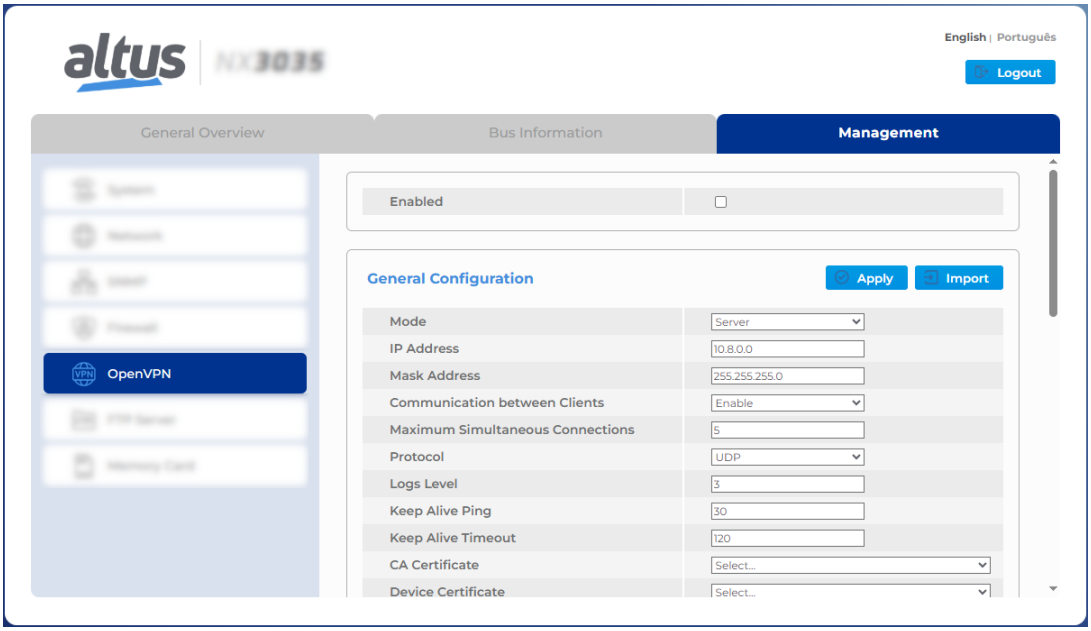



Figure 183: OpenVPN Configuration Screen

Because it is located within the *Management* tab, access to this page is password protected. The following sections describe the settings and functionality of this page.

### 5.20.2. Import Configuration

To quickly and easily configure the VPN on your device, you can use the *Import* button that appears in the picture 183 in the upper right corner of the page. Clicking on this button opens a file explorer window where you can select a configuration file. Files with extension *.conf* or *.ovpn* should be selected. When you select a file, its contents will be read and the configuration parameters present will fill their respective configuration fields on the web page.

For the file's parameters to be interpreted correctly, they must follow standard OpenVPN configuration file syntax.

If there are security files, certificates, or keys, written in the configuration file, along with the other parameters, they will be read and separated into separate files within the controller for use.

**ATTENTION**

Do not use spaces to separate the words in the name of the *".conf"* files. Instead, use *"_"* to separate them.

### 5.20.3. OpenVPN Configuration

Here is an image with all the settings for an OpenVPN server:

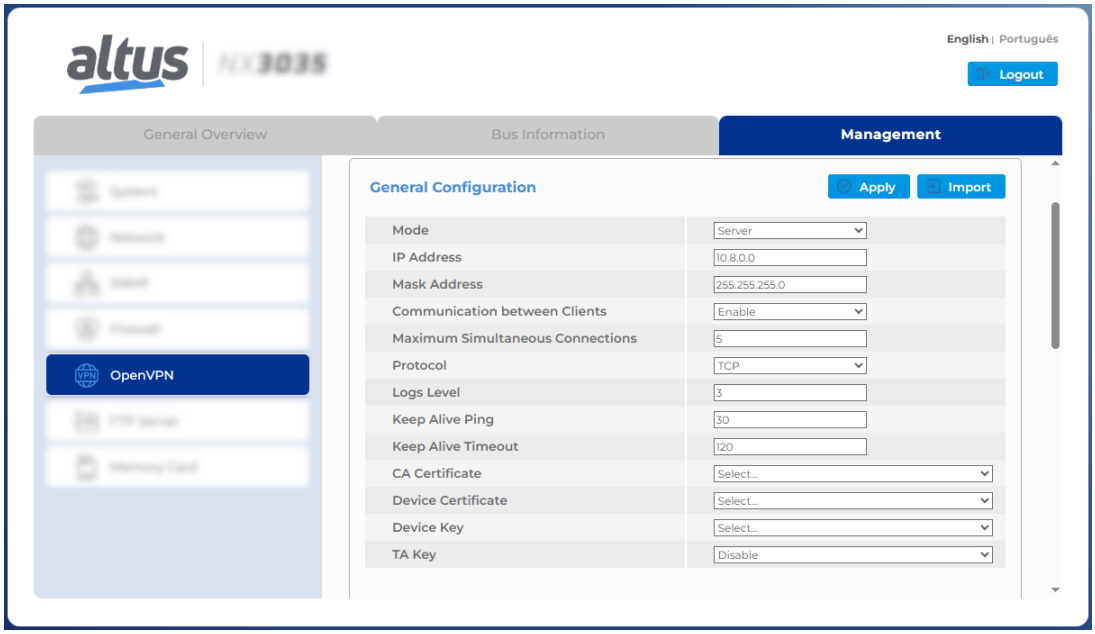



Figure 184: OpenVPN Server Configuration Table

Here is an image with all the settings for an OpenVPN client:

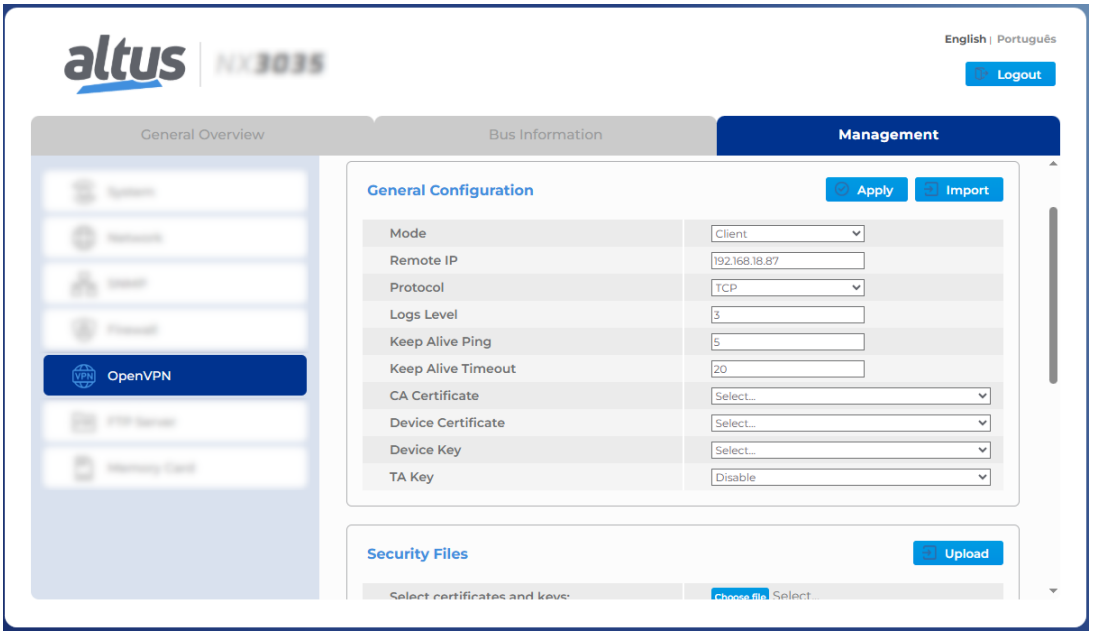



Figure 185: OpenVPN Client Configuration Table

This section shows how OpenVPN configuration is performed. The settings will be divided into three parts: settings common to both operating modes, settings unique to a server, and settings unique to a client.

#### 5.20.3.1. Common Configurations

Looking at the figures with the client configurations, figure 185, and the server configuration, figure 184, you can identify that several parameters are the same for both configurations. These are:

##### *5.20.3.1.1. Mode*

With the configuration of the *Mode*, you can select between two options, client or server. When you select one of the two modes, the settings table changes automatically to allow the configuration of the necessary fields for each mode of operation.

##### *5.20.3.1.2. Protocol*

This field configures which transport protocol will be used for VPN communication. It can be set between UDP and TCP.

**ATTENTION**

The configuration of the server and all its clients must be the same. With a divergent configuration, OpenVPN is not able to perform communication.

##### *5.20.3.1.3. Logs level*

This field sets the level that the log file will receive. The setting ranges from 0 to 5, 0 being the most basic level and 5 being the most advanced.

Level 0 only displays logs about some critical failure in OpenVPN and levels 4 and above are used for debugging as there is a lot of information being written to the log file. For normal operation, it is recommended to use value 3.

This field only accepts numbers as input. You are not allowed to use letters or special characters.

##### *5.20.3.1.4. Keep Alive Ping*

This field sets the time, *in seconds* when the *Ping* request will be forwarded. This request serves to verify the connection between the server and the clients.

This parameter can be set on both the server and the OpenVPN clients, but if this parameter is set on the server, the clients will assume the server's value and not the value set on them. If the server doesn't have such a setting, each client assumes its setting normally. If you want to disable pinging between the server and the clients, set the value to 0.

This field only accepts numbers as input. You are not allowed to use letters or special characters.

#### *5.20.3.1.5. Keep Alive Timeout*

This field sets the time, *in seconds* when the timeout of the *Ping* request will occur. After the expiration of this time, without a response from the other VPN device, it will be considered disconnected.

This parameter can be set on both the server and the OpenVPN clients, but if this parameter is set on the server, the clients will assume half of the server's value and not the value set on them. Clients receive half the amount to ensure that they are disconnected in case the server disconnects. If the Server does not have such a setting, each client assumes its setting normally. If you wish to disable this feature, set the value to 0.

This field only accepts numbers as input. You are not allowed to use letters or special characters.

#### *5.20.3.1.6. Security Files*

In the fields *CA Certificate*, *Device Certificate*, *Device Key* and *TA Key*, you must select which security file, certificate, or key, will be used to establish the OpenVPN communication. The options in each field, *combobox*, are filtered according to the type of key file or certificate, although there is no differentiation between keys and certificates.

To be possible to select a file, it must first have been imported.

All security files are required for correct communication to be established between clients and the VPN server, except for *TAP Key*. This key is optional for communication, but if it is used on the server, it becomes mandatory for all clients on the server.

See the TLS Key and Certificate Management section for further information about generating certificates and security keys based on TLS.

#### *5.20.3.1.7. TA Key*

In the field *TA Key* it is set which type of encryption will be applied to the *TA Key*. This field stays hidden until you select a file for the TLS key because it is only used in conjunction with this key. The default value of this parameter is *SHA1*, but you can select from the following values: *SHA256*, *SHA512*, and *MD5*, in addition to the default SHA1.

**ATTENTION**

This configuration needs to be the same between the clients and the server in the same OpenVPN network. If the value of this field is different between the client and server, the connection will not be established.

### 5.20.3.2. Exclusive Server Configurations

The exclusive server configurations, seen in figure 184, are described below.

#### *5.20.3.2.1. Network Address*

The IP range that will be used to assign the server and client addresses for the VPN network is configured by the server by setting the *IP Address* and *Mask Address* fields. All IPs that will be assigned to the clients and the server will be taken from the specified range.

The server's IP address is always the first available value in the configured range, and for IP assignments to clients, the values still available in the range are used, so the first available value is assigned as clients make their connection. For example, if a network is configured with the addresses 10.8.12.4 and mask 255.255.255.248, the server will assume IP 10.8.12.5 which is the first available address in the configured range. However, if mask 255.255.255.255.0 is set, the server will assume IP 10.8.12.1, which is the first available address in the range.

The IP and Mask address fields only accept settings that have the syntax of an IP address and mask address, respectively. If anything out of the standard is configured, an alert message will be displayed, informing you that an error has occurred.

#### 5.20.3.2.2. *Communication between Clients*

In this field, you can enable or disable communication between clients in the VPN network. When the option is selected as *Disabled*, only client-server communication can be performed directly. If the option selected is *Enabled*, it will allow communication between the clients themselves in addition to the client-server communication.

#### 5.20.3.2.3. *Maximum Connected Clients*

In this field, you can set the maximum number of clients that can connect to the server simultaneously. This field accepts only numeric characters, and the minimum value is 1.

#### 5.20.3.2.4. *Private Networks*

When you select OpenPVN's operating mode as a server, a table will be displayed, normally hidden, which allows the configuration of private networks that can be below the server and each client.

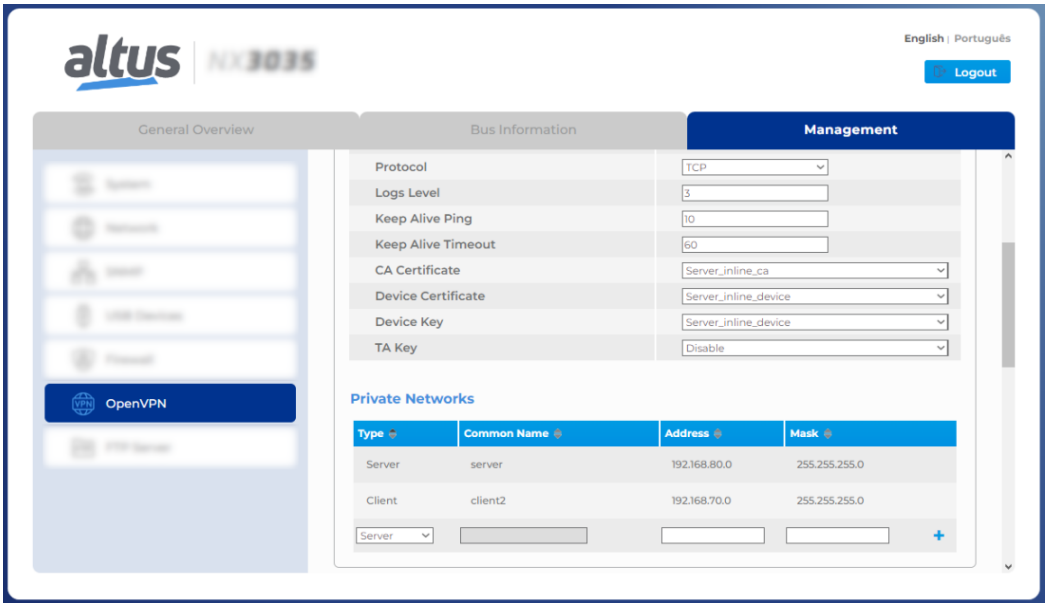



Figure 186: OpenVPN Private Network Configuration Table

To configure a private network that is below the server, simply select the network type as *Server* and configure the network addresses and mask. Configuring a private network for a client requires, in addition to setting the type as *Client*, to enter the *Common Name* of the client that owns the network being configured.

The Common Name of a client is set when generating the *Device Certificate*. This parameter is entered when creating the certificate and is unique for each client and server. The configuration of these private networks creates a routing table that will be checked when receiving or sending packets over the VPN.

Figure above, shows a configuration of a subnet *80* on the OpenVPN server, then a routing rule will be configured that will forward the data packets that will be received by the VPN to the device interface configured on this network. It also creates a rule, internal to the server, that if a data packet has the subnet *70*, this packet will be routed and forwarded through the VPN tunnel. The same behavior occurs with the *client2* client, but with the subnets switched, because below this client is the *70* subnet and it will forward packets with the *80* subnet to the VPN tunnel.

See the following figure for an example of architecture:

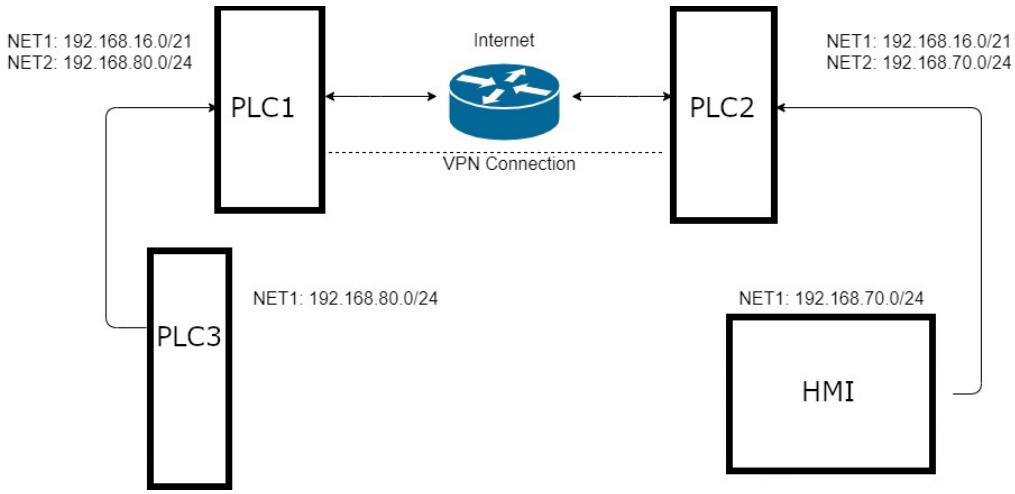



Figure 187: Architecture Example with Private Networks

In Figure 187, the PLC1 on the left has a private network 80 configured on its NET 2, and connected to it is the PLC3 on the same network. The PLC2 on the right has a private network 70 configured on its NET 2 and connected to it is an HMI, on the same network. The example architecture realizes the communication between the PLC3 and HMI devices over VPN by configuring their respective private networks.

After filling the fields, shown in Figure 186, with the desired configuration, you must click on the blue + button that appears on the far right of the configuration fields, so that the rule is added to the table. If you want to delete a rule, drag your mouse over the rule you want to remove, and a red *X* will appear on the right, as shown in Figure 186. By clicking on this *X*, the rule is removed from the table.

For the settings present in the table to be applied to the device you must click the *Apply* button and confirm the operation in the confirmation window that will appear. When the rules are applied, a message will be displayed indicating whether the operation was successful or not.

#### 5.20.3.3. Exclusive Client Configurations

There is only one configuration unique to OpenVPN clients on the page, which you can see in the picture 185. This configuration is the *IP Remote*.

##### *5.20.3.3.1. Remote IP*

The Remote IP field sets the address where the VPN server is expecting communication from the clients. If an OpenVPN server is established on a computer, the remote IP configuration must be done according to the IP address of this computer. This field also accepts *host names* as the remote address, so you can set an IP or a hostname in this parameter.

**ATTENTION**

Because of the need to allow for such different parameters, IPs, and host names, the only check that exists in this field is whether or not data exists. Be careful when performing the configuration.

#### 5.20.3.4. Application Settings

To enable the functionality, the checkbox *Enabled*, shown in the figure above, must be checked. If you just want to apply the settings you have made and not enable OpenVPN, uncheck this checkbox.

After you have made all the desired settings, the settings must be applied to the device, to do this use the *Apply* button. This button is shown in the figure 185 in the lower right corner. When the settings are applied and the VPN is enabled, the web page will perform an automatic *scroll* to the OpenVPN *status* table, displayed in the Status Table section.

### 5.20.4. Security Files

Security files are used to establish OpenVPN's communication securely by performing the role of encrypting and decrypting the data packets that will travel through the VPN tunnel. In the TLS Key and Certificate Management section, it is described

how to generate TLS keys and certificates. Here is a screenshot that shows the section responsible for managing the security files:

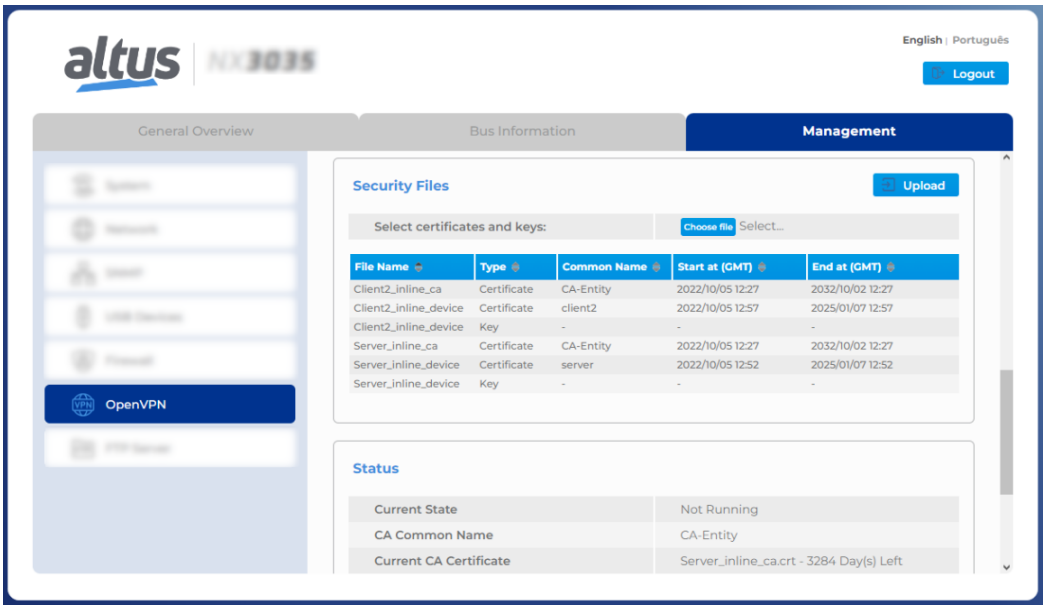



Figure 188: OpenVPN Security Files Table

In this section of the System Web Page, you can manage the security files. You can import files, monitor the validity of certificates, download files uploaded to the device, and delete files that have been uploaded.

By clicking the *Choose files* button, you can import certificates and keys, these files must have the respective extensions *.crt* and *.key*. This button opens a file explorer window and allows the selection of one file, i.e. multiple files.

**ATTENTION**

There is a limit of 12 files that can be imported into the controller.

The control of the files is done in the table, which is shown in the picture 188. This table adds new items, or removes them, as the import or delete operations occur. You can identify whether the file is a key or a certificate by the second item in the list, the *Type*, which indicates what that file is. For the certificates, their *commons names* and their expiry dates, both start and expiry, are also displayed.

You can recover a file that has been imported into the part and also delete it. When you drag the mouse over a file in the table, two buttons appear, one for downloading and one for deleting. The download button is a black arrow pointing downwards, and the delete button is a red *X*.

### 5.20.5. Status Table

Designed to allow for data monitoring, OpenVPN's status table automatically expands as you change settings and displays various data about the connection such as the state of the VPN, the VPN IP assigned to that device, the data being transmitted, and the security files being used for communication.

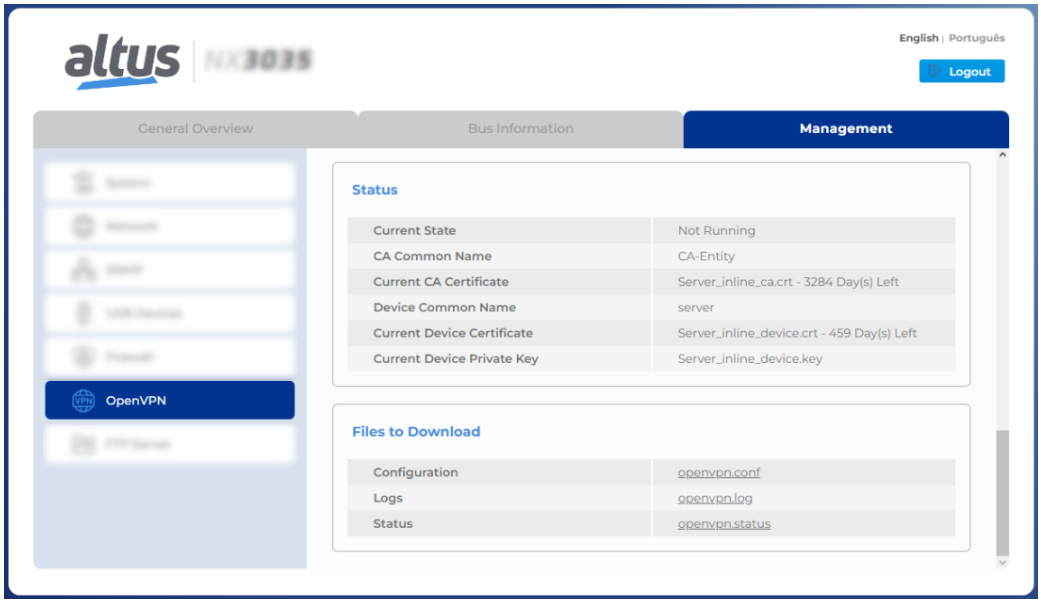



Figure 189: OpenVPN Status Table with Feature Disabled

When VPN is disabled, the table has few parameters. The field *Current State* indicates whether the VPN is enabled or not, and the other fields show which certificates and keys are configured for VPN communication. If one of the security files has not been selected, the character "-" will appear instead of its name, indicating that there is no file configured.

The common name fields for the CA and the device display the names given to the respective certificates, certificate authority, and device.

Next to the file name of each certificate is displayed the remaining time, *in days*, until its expiration date.

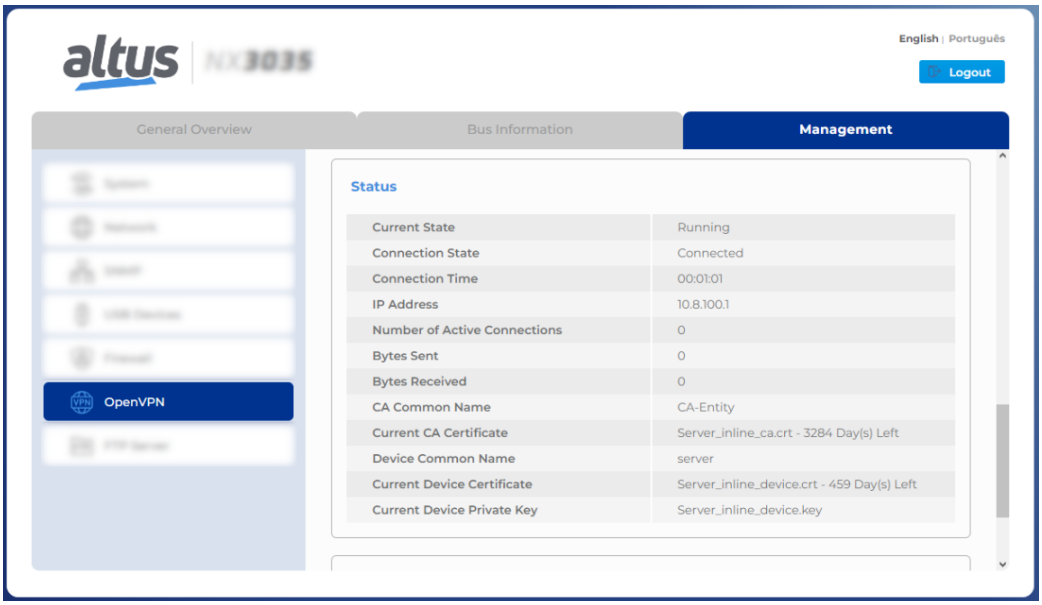



Figure 190: OpenVPN Status Table with Feature Enabled

When the functionality is enabled and the settings are applied on the device, the table has its cells dynamically modified so that the remaining information is displayed. Information about the OpenVPN connection status can be found in the first two topics of the list.

The item *Current State* has the states of *Not Running*, *Starting service...*, and *Running*, which indicate respectively that the VPN is disabled, is starting or is enabled.

The item *Connection State* has the states *Not connected*, *Connecting...*, and *Connected*.

The other information that can be obtained from this table is the total connection time, the device's IP address, and the amount of data sent and received, in bytes. The status of how many clients are currently connected is only displayed when OpenVPN is operating as a server.

### 5.20.6. Download Section

You can check the information generated by OpenVPN through status and log files. The list of files to download is only displayed when there is a file to download. If there is none, the message "No file found in the controller!" is displayed. Clicking on any of the links will download the requested file through the browser.

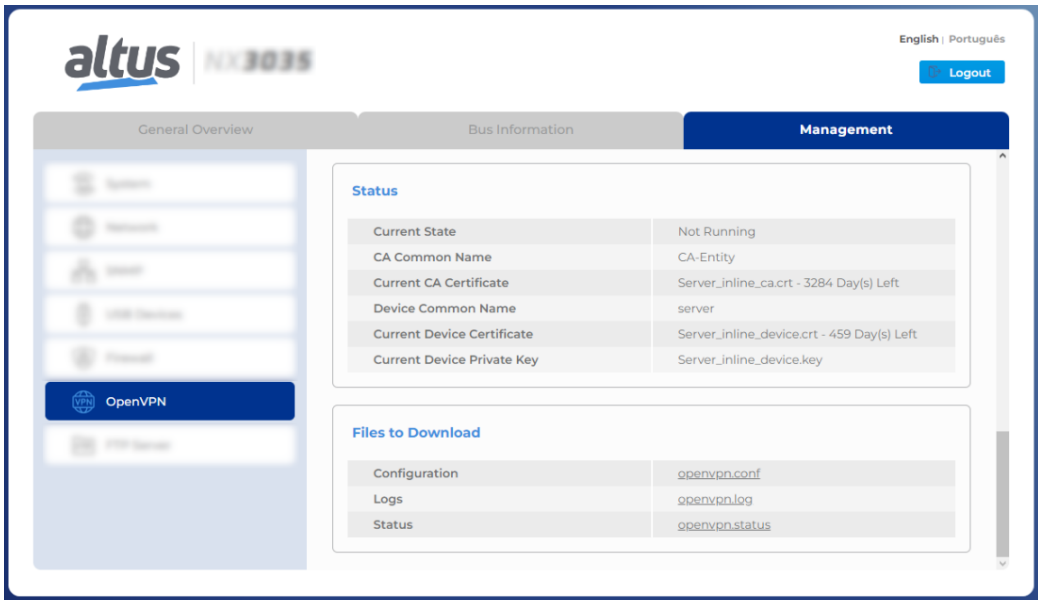



Figure 191: OpenVPN Download Section

The download file list is only displayed when there is a file to download. If there is no file, the list item remains hidden. Clicking on any of the links will download the requested file through the browser.

### 5.20.7. Architectures Configuration

This section will cover some possible configurations for OpenVPN, such as Host-to-Host, Host-to-Site, and Site-to-Site architectures.

#### 5.20.7.1. Host-to-Host Configuration

Below is a picture that represents a Host-to-Host connection:


Figure 192: Example of Host-to-Host architecture

This topology allows the connection between two VPN hosts. Both hosts can be chosen to be configured as the server, then the other should be configured as the client, or both hosts can be configured as clients and have a third host that will be the server for the VPN network.

Setting up this type of architecture doesn't require any specific configuration. In other words, there is no restriction on the settings available on the OpenVPN web page.

#### 5.20.7.2. Host-to-Site Configuration

Below is a picture that represents a Host-to-Site connection:

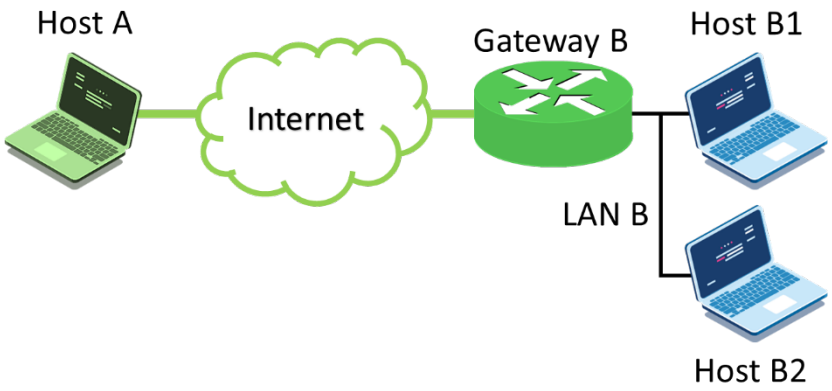



Figure 193: Example of Host-to-Site architecture

This topology allows the connection between two VPN hosts, but one of these hosts also acts as a gateway to the VPN network. Through this gateway, routing is performed to set up communication between hosts A, B1, B2, and Gateway B. In this scenario, either Host A or Gateway B can be the server. When one is the server on the network, the other will be the client.

The hosts, B1 and B2, that are on a private Lan B network below Gateway B, don't need to support OpenVPN to be able to communicate since all communication is handled by the VPN network gateway.

To enable communication between all devices on the network, you need to create routing rules for the VPN tunnel. Please refer to the section Private Networks to see how to create private network rules.

This VPN connection architecture requires some specific configurations. The server must have its topology configuration as a Subnet, this being the default configuration of the controller, to configure the private networks under Gateway B, as seen in the image above.

You also need to enter the address of the private network, Lan B, that will be communicating through the VPN. This configuration is done using the command *push "route Lan_B_IP Lan_B's_Mask"* and is required regardless of whether the private network is located below the client or the OpenVPN server, but if the private network is below the VPN client, you must add, in addition to this command, the following configuration: *route route Lan_B_IP Lan_B's_Mask*. These settings are written to the VPN server's configuration file.

#### 5.20.7.3. Site-to-Site Configuration

Below is a picture that represents a Site-to-Site connection:

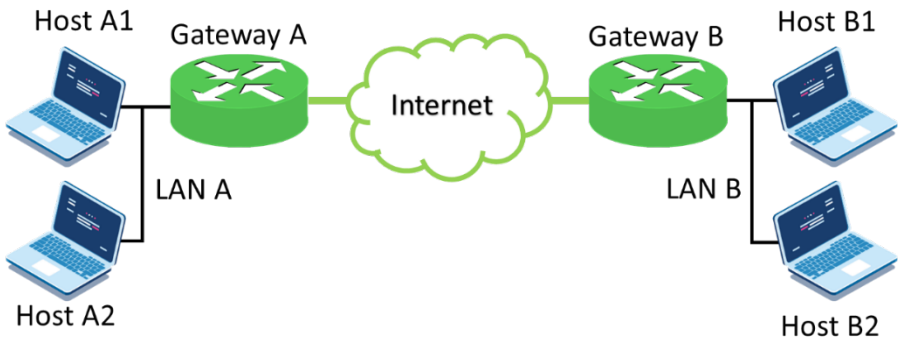



Figure 194: Example of Site-to-Site architecture

This topology allows the connection between two VPN hosts, both of which acts as a gateway to the VPN network. Through these gateways, access is provided to establish communication between hosts A1, A2, B1, B2, Gateway A, and Gateway B. In this scenario, any gateway can assume the role of a server, so the other will be the client.

None of the hosts that are in a private network below one of the two gateways need to support OpenVPN to be able to communicate, since all communication is handled by the VPN network gateways.

To enable communication between all devices on the network, you need to create routing rules for the VPN tunnel. Please refer to the section Private Networks to see how to create private network rules.

The configurations for this architecture need the same specific settings described in section Host-to-Site Configuration, with the difference that now, there are two private networks, and both must follow the configuration that has been demonstrated. Assuming that Gateway A is the server on this connection, you should add the following commands to the configuration file: *push "route Lan_A_IP Lan_A's_Mask"*, *route Lan_B_IP Lan_B's_Mask*, and *push "route Lan_B_IP Lan_B's_Mask"*. If the server is Gateway B, in the configuration file it would be added: *push "route Lan_B_IP Lan_B's_Mask"*, *route Lan_A_IP Lan_A's_Mask*, and *push "route Lan_A_IP Lan_A's_Mask"*.

## 5.21. Docker

The Docker functionality was developed to expand and enhance the integration and operation of the CPU. The main purpose of Docker is to enable the execution and maintenance of Docker *Containers*. The installation, execution and maintenance of the *Container* are carried out through the *Portainer* management platform.

**ATTENTION**

There is a wide range of applications available as Containers and openly distributed on platforms such as *Docker Hub*. Altus does not provide specific technical support for the use and operation of these applications; this should be handled directly by the Container developer or maintainer. Furthermore, due to the processing and memory constraints of the programmable controller's CPU, many of these containers may not run properly. Therefore, we recommend to check the tutorials available on Altus website, which use previously tested containers compatible with the resources available on this platform.

### 5.21.1. How to access a Docker Container

The vast majority of Docker *Container* are accessible through an exposed port configured for the *Container*. The default exposed port can be discovered by searching in the documentation for that *Container*. It is expected that the *Container* will always be linked to the default port listed on those sources.

#### 5.21.1.1. How to access Portainer

Our management platform of Docker Containers, *Portainer*, is a *Container*. Like most *Containers*, it is accessible through the exposed port in which it is linked to. In *Portainer's* case it is located in the CPU IP followed by *":9000"*. For example: *"192.168.15.1:9000"*. After accessing the link, you will be prompted to create an account to access the Portainer platform. This account should have a username and a password of at least 12 characters.

**CAUTION**

This password is not in the *Altus* domain and is **NOT** recoverable! The password should be stored with caution!

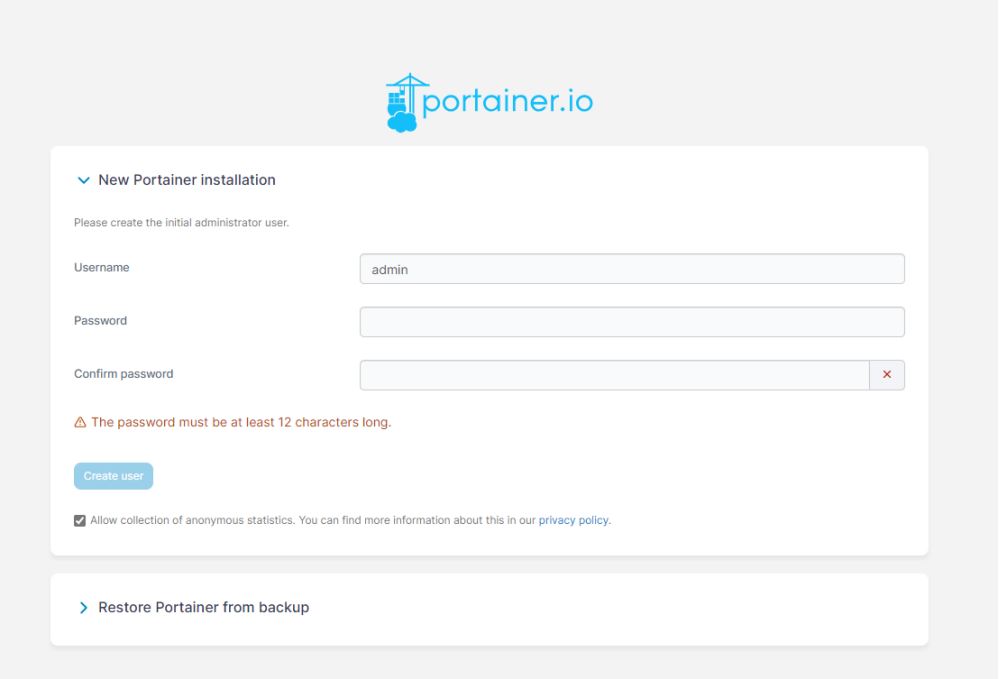



Figure 195: Portainer's Login Screen

It is also possible to use the *"Restore Portainer from backup"* option to restore all configurations using a restoration file.

**ATTENTION**

It is recommended that the backup file be updated regularly.

### 5.21.2. Quick Setup

Once connected, the platform will redirect you to the home screen. This home screen will contain two options: proceed using the local environment in which Portainer is running or add an environment to connect to another available environment.

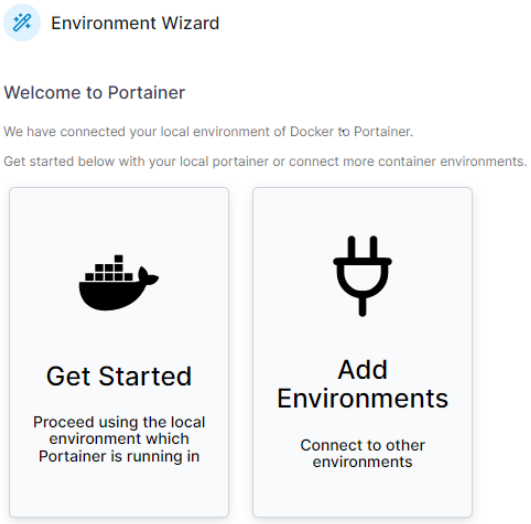



Figure 196: Portainer's Initial Screen

The correct option to be chosen is *"Get Started"*. Once picked, *Portainer* will redirect you to a screen that shows the available environments in your device. It will have only one environment, that is the Docker environment, easily identifiable by the whale logo that it has adopted.

#### 5.21.2.1. Local Environment Informations

You can find information about the Docker environment running within the CPU on the home screen. Among this information are: the complete date, Docker server version, Stacks, and the number of *Containers* followed by their statuses. The number of created Volumes, and Images present in the UCP, the number of usable CPUs, and the maximum amount of memory available for all the *Containers*.


Figure 197: Environment Info Screen

**ATTENTION**

It is important that the date in the environment is correct! It can be configured through the clock setting function on the CPU web page.

### 5.21.3. Environment's Dashboard

After accessing the *Dashboard* by clicking on the desired environment, you will encounter some of the information previously displayed in a table of basic information named *"Environment info"*. these include the available hardware information and the Docker version. It is possible to access the menus listed above by interacting with them. From all the menus listed, the most relevante are the *Container menu* and the *Image menu*.

**ATTENTION**

The PLC does not support containers that require a dedicated graphics processor.

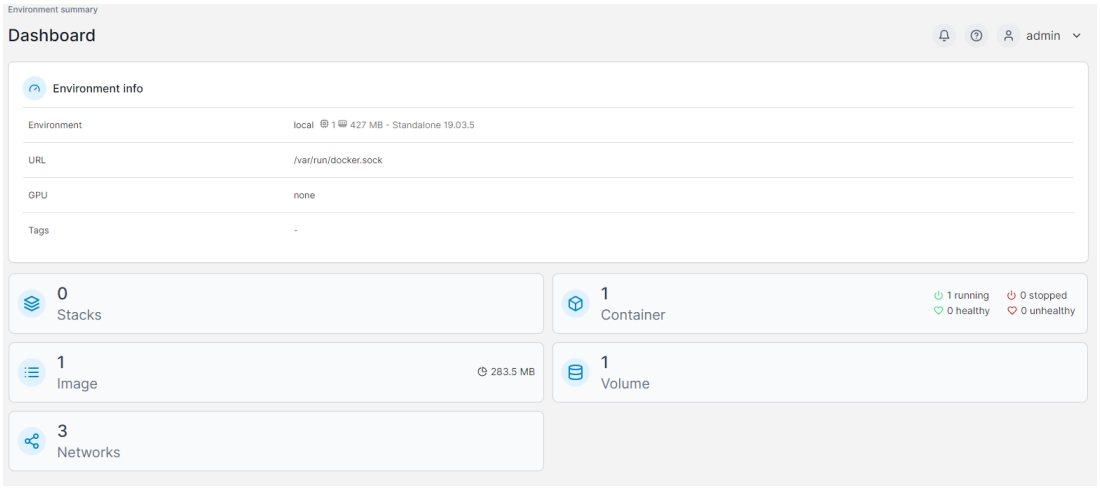



Figure 198: Dashboard

**ATTENTION**

For more information about the menus and/or Docker itself, you can access the *Portainer* documentation, which is available by clicking on the *"(?)"* symbol next to the user in the top right-hand corner of the screen.

#### 5.21.3.1. Containers Menu

The *Container* menu is responsible for the comprehensive administration of *Container*. It allows access to the *Container*, the image, and some relevant quick actions. You can see the creation date, IP address, published ports, when applicable, and the *Container*'s owner profile.

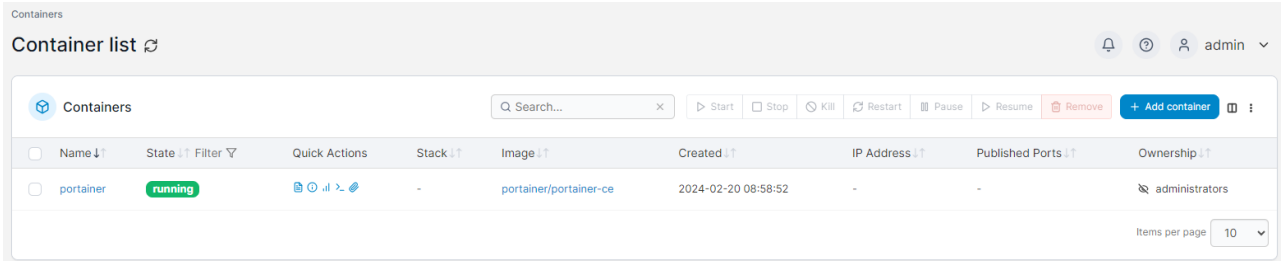



Figure 199: Containers Menu

**CAUTION**

No *Stop, Kill, Restart* or *Pause* commands should be performed on the *Container* of the *Portainer*. If any of these operations are performed, a reboot is required for the *Container* to function again.

#### 5.21.3.2. Images Menu

The *"Images"* menu is responsible for the administration of downloaded images and their statuses. The name and tag of images can be obtained through the *"Search"* button located to the right of the Image input box. The *"Pull the Image"* button is responsible for obtaining the image. As indicated on this screen, there is a limitation on image *"pulls"*.

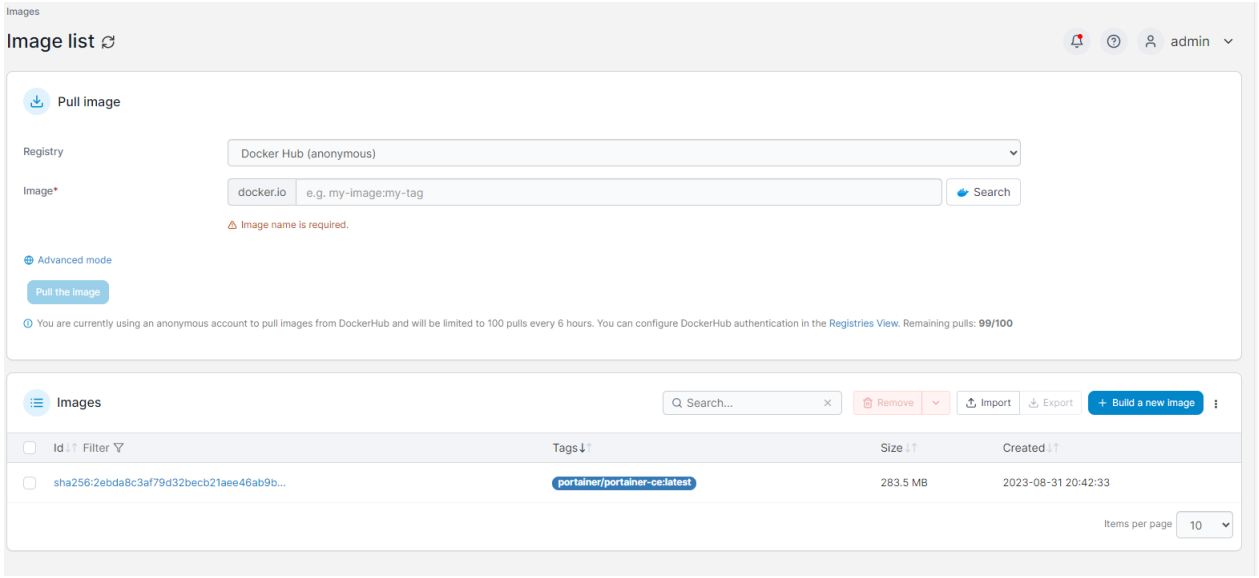



Figure 200: Images Menu

Below, in the *Images* menu, you can identify all the obtained images, their status, whether it is *"Unused"*, their tag, the occupied size, and the creation date. Like the *Stack*, you can add images using a web editor through the *"Import"* and *"Build a new Image"* buttons.

### 5.21.4. Creating and Configuring a Container

A *Container* can be created through the *Container* menu, accessible from the *Dashboard*, or through the quick access menu on the left side of the *Portainer* web page. There is also the option to create a *Container* through *Stacks* and configure them later.

**ATTENTION**

Creating custom images using Portainer's *Build Image* feature is not supported on this platform.

**ATTENTION**

During the process of installation and configuration of a *Container*, it is highly recommended that the controller be stopped, with the application removed using the *Reset Origin* command. This is necessary because, due to the intrinsic characteristics of Docker technology, these operations perform low-level access to the controller's operating system which can temporarily affect the stability of the control function (temporary performance degradation and loss of determinism) and, in extreme cases, even cause the system to reboot.

Using the *"Add Container"* button, you can add a *Container* from images you have already obtained through the *Images* menu or images that will be obtained in the *Container* creation process.

Clicking on *"Add Container"* will take you to the *"Create Container"* screen.

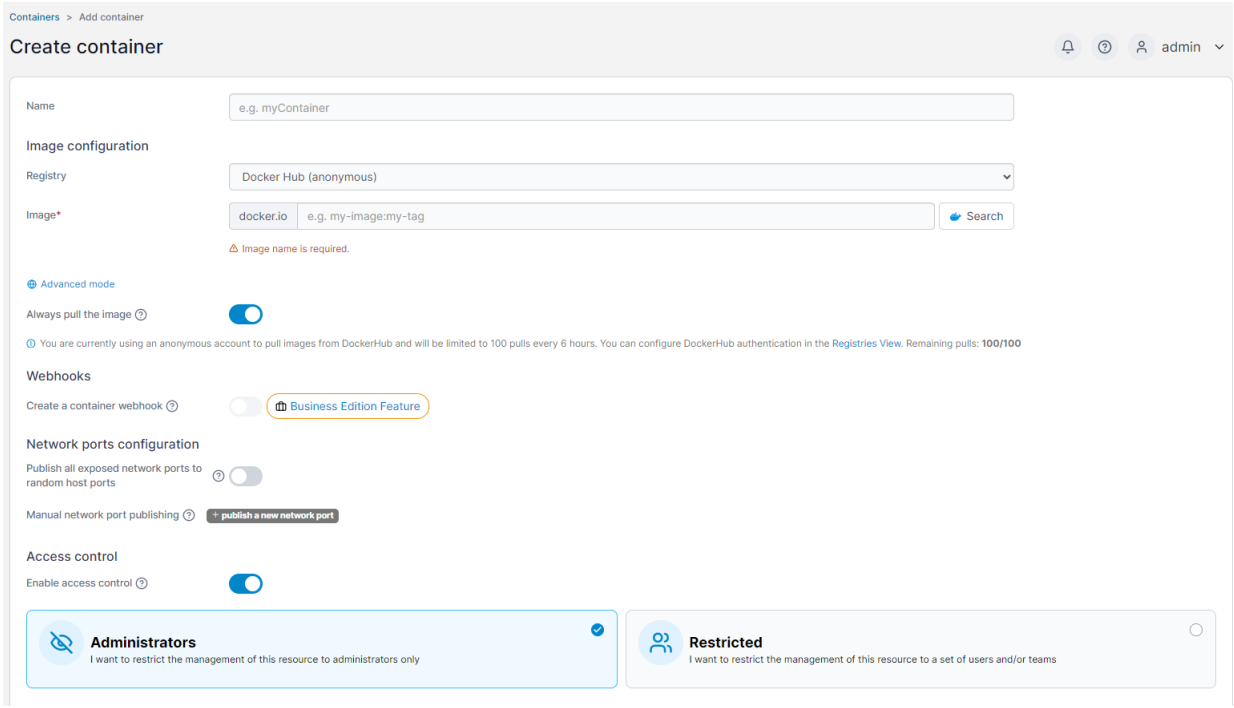



Figure 201: Create Container Screen

On this screen, you can name the *Container* being used and specify the image you want to use.

Like the *Images* menu, you can access Docker Hub to view available images and their respective names. There is an option for *"Always pull the image"*. If activated, shouldn't be needed to add the image through the *Images* menu beforehand. It is **NOT** always the case. So the *Container* may not be created if the image is not available in the device. To add an image, you can use the Images Menu as a guide. The options in this basic *Container* configuration box are case-sensitive and should be evaluated according to the specific requirements.

To create the *Container* definitively, you can use the *"Deploy the Container"* button. However, it is of paramount importance to perform the additional configurations present in the item below.

In case you are creating *Containers* through the *Stacks* method, it is important to noice that:

**CAUTION**

When using the *stacks* option (docker-compose) when passing the parameters for creating a volume, the following path must be used: "/codesys/NextoUser/Files/docker/var-lib/". Otherwise, the startup of the *container* will be aborted.

### 5.21.4.1. Configuring a Container

#### *5.21.4.1.1. Command & Logging Menu*

The first available menu is the *Command & Logging* menu. In this initial menu, there is no need to change any parameters; however, it is recommended to activate the console configuration for *Container* that can take advantage of it.

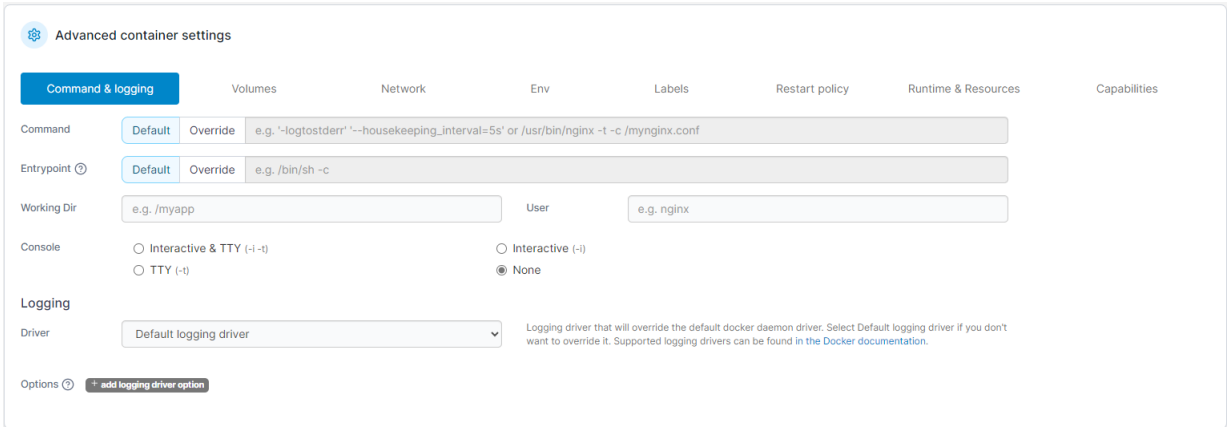



Figure 202: Command & logging Configuration Menu

#### 5.21.4.1.2. *Network Menu*

The Network menu allows the configuration of the network in which this *Container* will operate. The use of network in *"Host"* mode is mandatory for the operation of the *Container*. This happens because our Docker does **not** support any other network's mode.

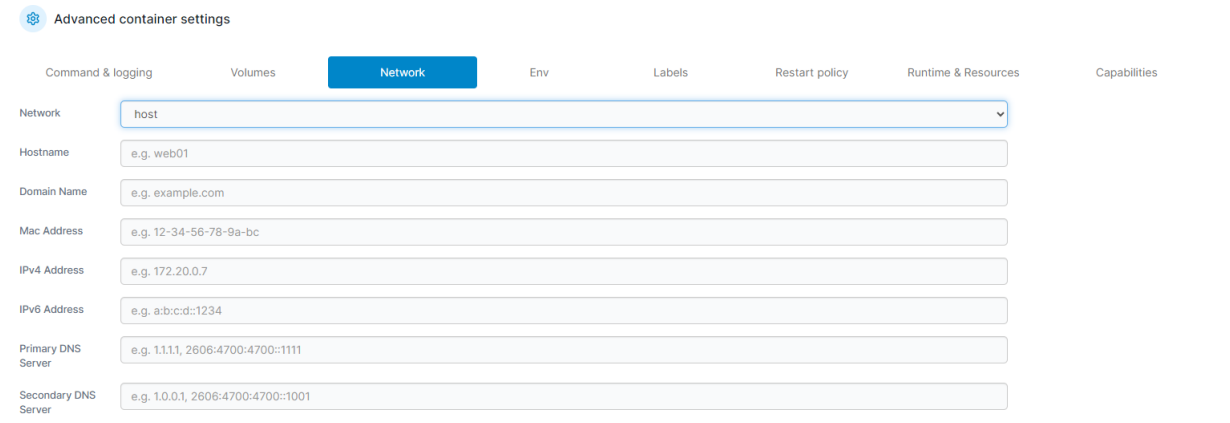



Figure 203: Network Configuration Menu

#### 5.21.4.1.3. *Runtime & Resources Menu*

The *Runtime & Resources* menu is the most crucial among the menus present in a *Container*. In this menu, the user can configure the maximum CPU and RAM that a container can consume.

**CAUTION**

It is recommended that the total sum of RAM memory usage limits of all *Containers* don't surpass the 230 *MegaBytes* mark to ensure proper reserves of RAM for the PLC operation.

The figure below shows the recommended configurations for using just one Container:

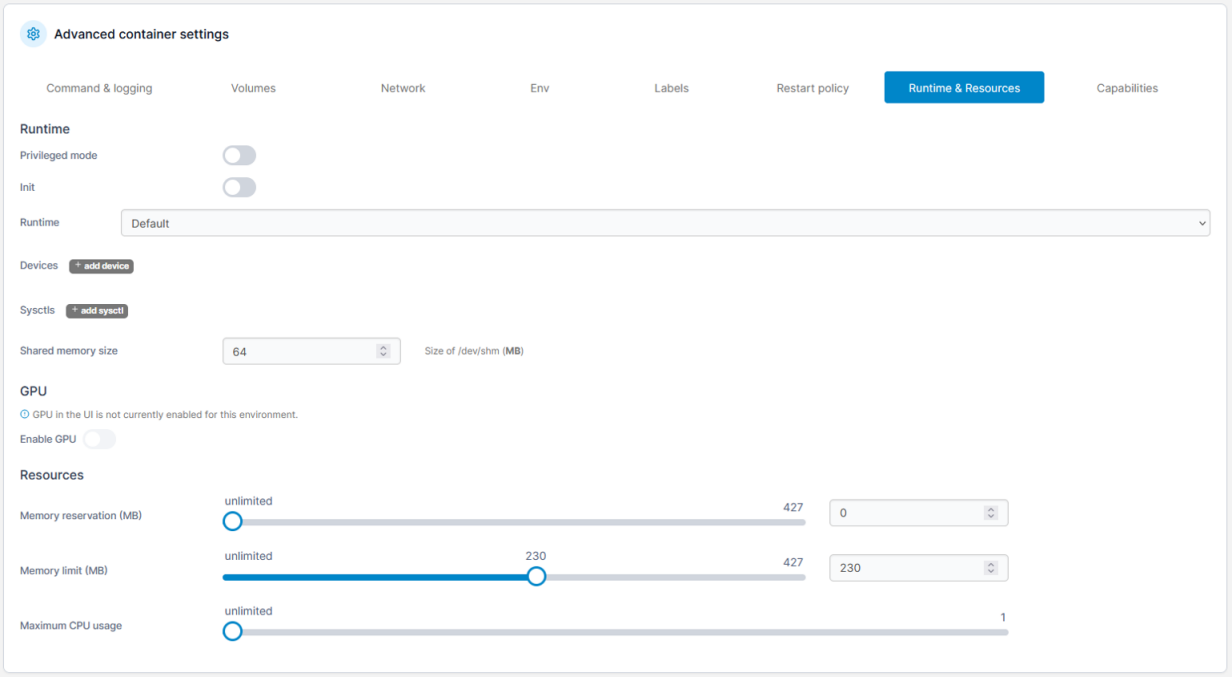



Figure 204: Runtime & Resources Configuration Menu

#### 5.21.4.1.4. *Capabilities Menu*

Finally, there is the *capabilities* menu, which provides a large number of *Container* capabilities. Each one of them comes with a brief explanation of how it works, accessible through the small *"(?)"* symbol next to each configuration.

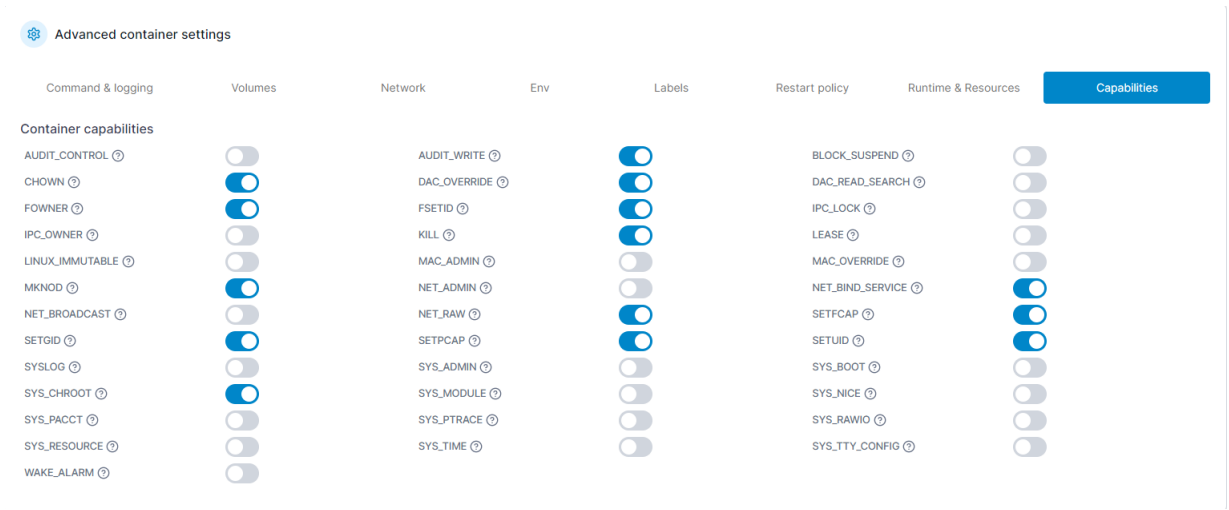



Figure 205: Capabilities Configuration Menu

**CAUTION**

We do not guarantee the desired system functionality in the case of activating certain parameters. Caution should be exercised when enabling or disabling sensitive capabilities.

#### 5.21.4.2. Using a Container

As mentioned in section How to access Portainer, most *Containers* can be accessed using the address *"IP.OF.YOUR.CPU: CONTAINER PORT"*, followed by a port address. One way to access a *Container* is through *Portainer* itself and in its case, the port address is ":9000", meaning that to access it you would use *"IP.OF.YOUR.CPU:9000"*.

Once the *Container* is created, upon accessing it, two sections will be visible: an actions section and a status menu.

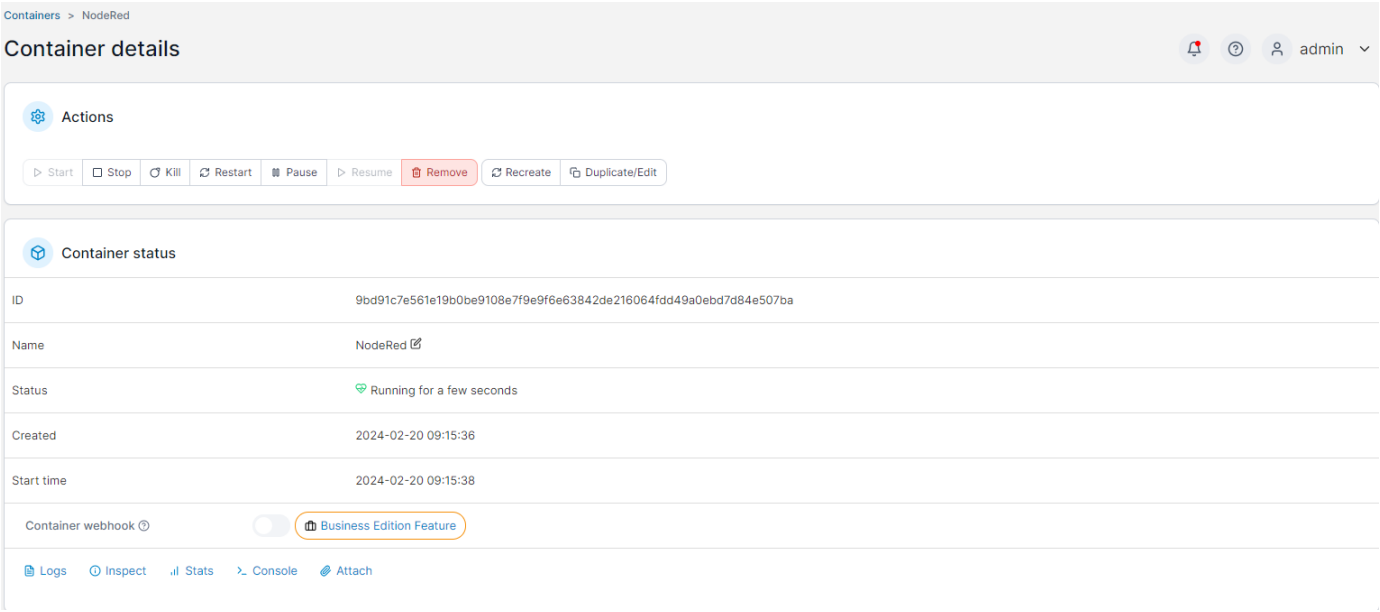



Figure 206: Container Details Table

This actions table enables the execution of some general actions that can be performed with the *Container*. The table below, called *"Container Status"*, presents some general information about the *Container*. At the end of this table, some functionalities can be accessed: The first one is the page of logs for the running *Container*, the second is a deep inspection of parameters and configurations of the *Container*, the third provides relevant statistics of resource consumption generated during the *Container*'s operation, and the fourth is a console for interaction with the *Container* in relevant cases.

**ATTENTION**

You must know the type of console supported by the *Container* you are using. If the selected option is not supported, it will not be able to interact with it.

The next table is the access control table, where the owner of the *Container* can be selected.

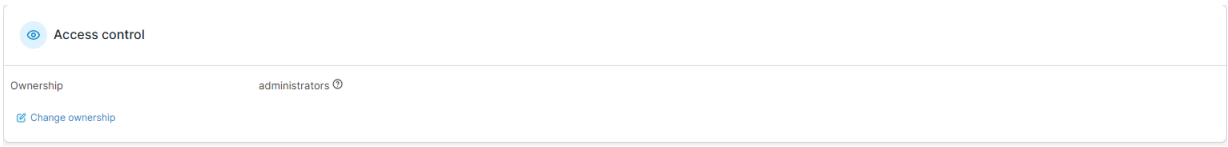



Figure 207: Access Control box

Following that is the *Container* health table, where it informs its state, failure count, and the last error code that was recorded.

The *"Create Image"* table allows the user to create an image of that Docker.

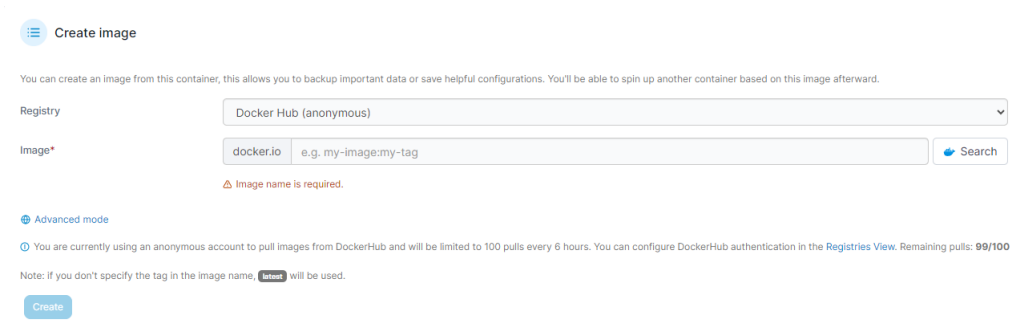



Figure 208: Create Image box Table

#### 5.21.4.3. How to create an image of an existing container to backup its data

On the *Container Details* screen, accessed from the *Containers* menu and clicking on the *Container* name, it is possible to create an image of this container containing all the data and configurations already configured. This feature can be used to facilitate the distribution of examples, to perform backups, or to migrate the Docker application to another controller.

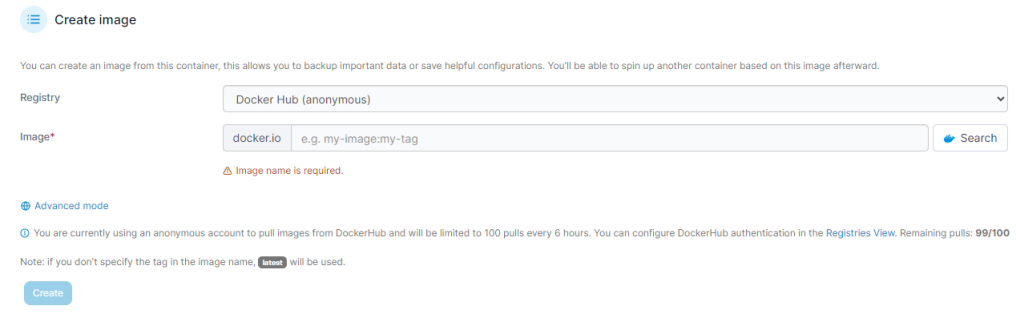



Figure 209: Creating an image of an existing Container

#### 5.21.4.4. Monitoring a Container

To monitor a *Container*, you can use a tool within the *Portainer* itself. This tool is accessible both via the list of *Containers*, in an icon in the *"Quick Actions"* called *"Stats"* and via the *"Container Statistics"* screen. Once accessed, the user will see the menu shown in the figure below. It will be possible to see statistics on memory, CPU, and I/O consumption. The refresh rate of the memory, UCP, and I/O consumption statistics can be configured via the *"Refresh rate"* field. This tool only monitors one *Container*, and it is not possible to monitor the sum of all *Containers* via *Portainer*. However, there are CPU diagnostics that can be monitored using the *Mastertool* and are directly related to the use of the *Container*, such as the *byProcessorLoad* diagnostic, where it is possible to view the percentage of CPU processor use, and the *dwFreeSpacekB* and *dwTotalSizekB* diagnostics, which show the total amount of user memory available.

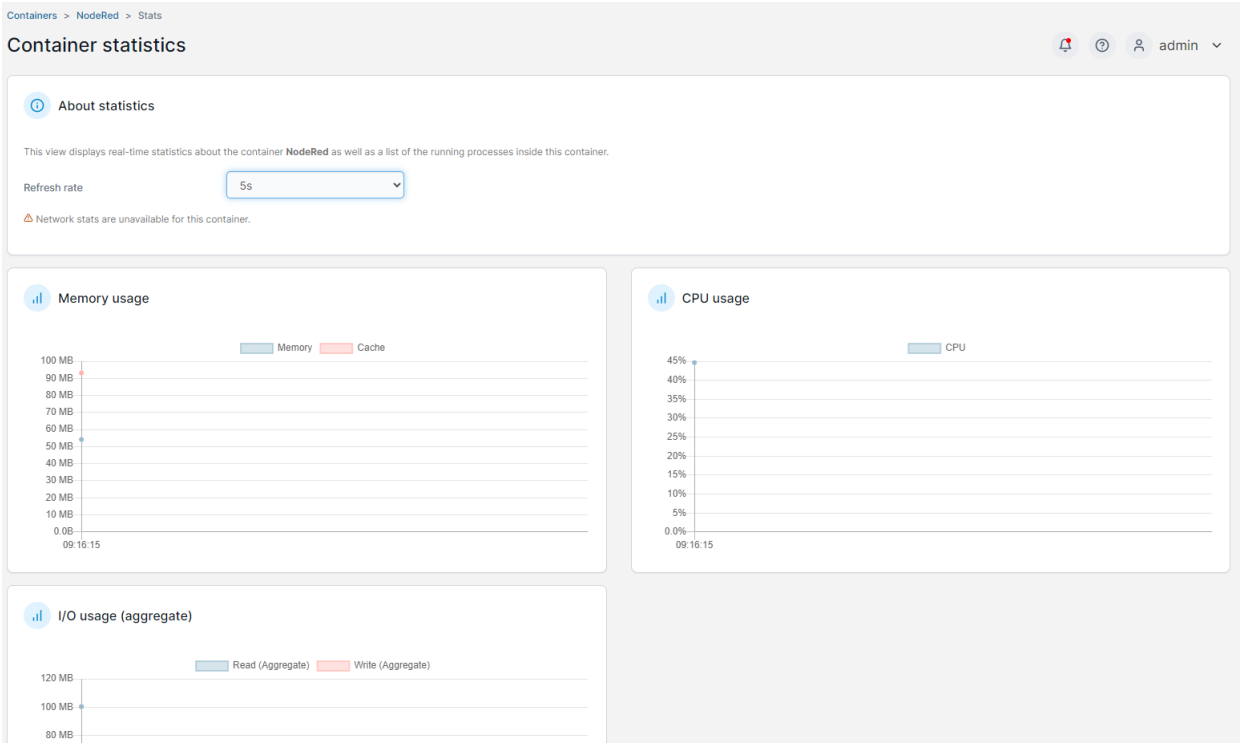



Figure 210: Stats Screen

### 5.21.4.5. Technical Characteristics

#### *5.21.4.5.1. Flash Memory*

The Docker *Containers* are located in the user's storage folder, accessible via the *Files* menu of the *Mastertool*. Because Docker is stored in the user memory, the maximum storage available for both Docker and the Application is the total amount of the device's user memory. *Containers* that use a large amount of storage can, in turn, take a long time to complete the process of obtaining the Image and deploying it.

#### *5.21.4.5.2. RAM Memory*

The Maximum RAM usage, among all Docker *Containers* **MUST** be inferior to 230 *MegaBytes*.

If an attempt is made to set a value higher than the one available, *Portainer* will display an error message. This can also result in the device malfunctioning. The option to use unlimited resources **NOT** should not be used, as this may result in the device malfunctioning.

**DANGER**

The memory is shared among **all** Docker *Containers* and Application. It should be carefully configured to avoid malfunctioning of the device. It is recommended that the sum of the RAM memory usage of all Docker *Containers* don't surpass the mark of 230 *MegaBytes*.

#### *5.21.4.5.3. CPU*

The CPU limitation used by Docker should also be configured through menu mentioned in section Runtime & Resources Menu. The configured value refers to the entire CPU. Therefore, if there is an application and/or *Container* consuming high processing power, this value should be modified accordingly. The device does not have a dedicated, integrated graphics processor or graphics drivers and may present instabilities in *Containers* that require this feature.

#### 5.21.4.6. How to access the Docker's folders

##### *5.21.4.6.1. Mastertool*

Through *Mastertool*, it is possible to check the files of Docker's *Container* using the File access functionality under the device menu. The following image illustrates the menu and the folder that allows access to these files.

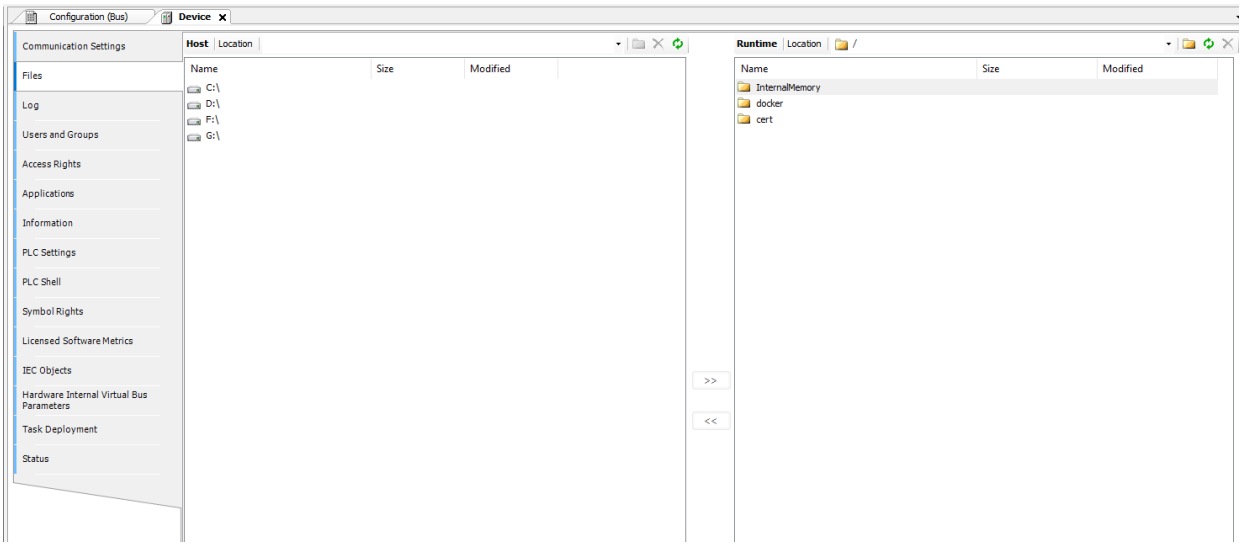



Figure 211: Docker's Folder in Mastertool

#### 5.21.4.7. Using External Memory as Docker Volumes

To expand Docker's storage capacity using external memory devices like SD cards and USB flash drives, you must use volumes to mount the external file systems into the containers. This process is only supported via Portainer Stacks (Docker Compose).

Use the following host paths to configure volumes in your stack file:

| External Device | Host Path |
|---|---|
| **SD Card** | /codesys/NextoUser/Files/MemoryCard |
| **USB Mass Storage** | /codesys/NextoUser/Files/Mass_Storage |

Table 159: Host paths for external memory devices

To create a Docker Stack that uses external memory, you must configure volumes to mount the physical device paths into your container. This allows containerized services to access files on the external storage.

To deploy a new stack, navigate to the Stacks section in Portainer and click Add Stack. The Figure 212 shows an example of a stack file that mounts both an SD card and a USB flash drive to a service:

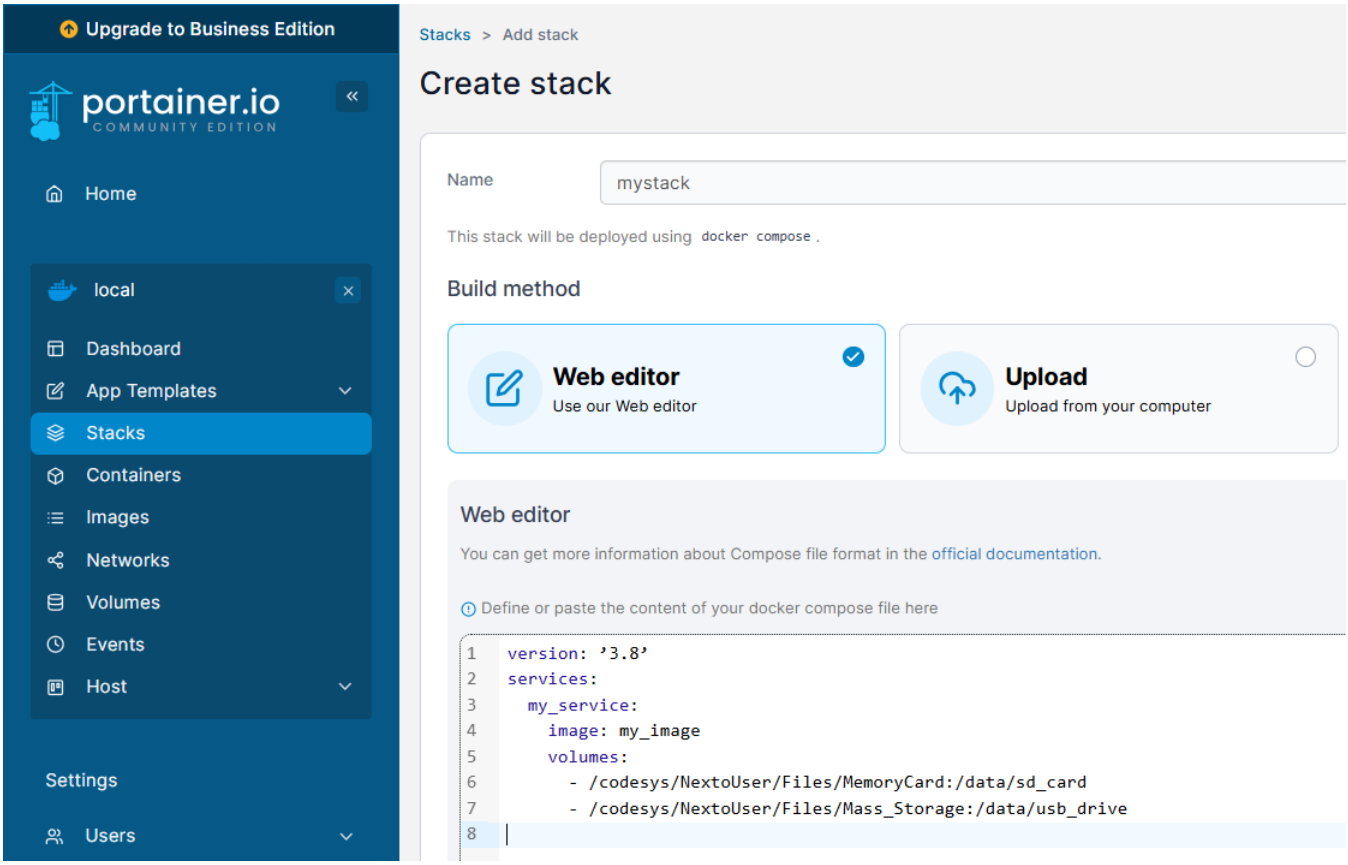



Figure 212: Stack File Example for External Memory Devices

**ATTENTION**

If the external device is not mounted during the container's startup, the container will not be able to access the files on that device. Ensure that the external memory devices are properly mounted before starting the containers. In the same way, if the external device is removed while the container is running, the container will lose access to the files on that device, needing to be restarted to regain access.

#### 5.21.4.8. Troubleshooting

This section aims to describe the most common problems, their causes and solutions.

##### *5.21.4.8.1. Container Installation Failure due to expired or not valid Certificate*

**Cause:** PLC time is invalid.
**Solution:** Adjust the PLC time.

##### *5.21.4.8.2. Portainer is inaccessible, returning after CPU reboot*

**Cause:** Container memory limits were not applied as per section Runtime & Resources Menu
**Solution:** Apply the memory limits as described in this manual.

# 6. Maintenance

## 6.1. Diagnostics

### 6.1.1. Diagnostics via LEDs

Nexto XF family controllers have five LEDs to indicate general states and diagnostics: SYS, RUN, DG, WD and PWR, and four more interface specific LEDs: SD, USB, CAN and RS485 as well as Ethernet connector LEDs. Integrated IO have their own LED indication acording to their function: digital or analog. The following tables show the meaning of each state and its respective description.

#### 6.1.1.1. SYS (System)

| Color | Status | Description |
|---|---|---|
| Green | Off | System off/Corrupted system |
| | Blinking | System booting |
| | On | System ready |

Table 160: System LED Status Description

**Notes:**

**Blinking eternally:** If a long time passes and the LED never stops flashing, the CPU may have lost its signature, check the Description of the Watchdog LED States table.

#### 6.1.1.2. RUN (Application)

| Color | Status | Description |
|---|---|---|
| Red/Green | Off | No application |
| Red | On | App in Stop mode |
| | Blinking 5x | App in breakpoint |
| Green | On | App in Run mode |

Table 161: RUN LED Status Description

#### 6.1.1.3. SD (SD Card)

| Color | Status | Description |
|---|---|---|
| Green | Off | SD Card not mounted / SD Card not connected |
| | On | SD Card mounted |
| | Blink | Unknown file system / Could not mount |

Table 162: SD Card LED Status Description

#### 6.1.1.4. DG (Diagnostic)

| Color | Status | Description | Causes | Priority |
|---|---|---|---|---|
| Green | Off | No active diagnostic | Normal operation | - |
| | Blinking 2x | Bus modules with diagnostic | At least one module, including the CPU, has a diagnostic | 1 |
| | Blinking 3x | Data forcing | Some memory area is being forced by the user via Mastertool | 2 |
| | Blinking 4x | Bus config or hardware error | Bus config or hardware error | 0 (higher) |
| | Blink | Wink identification | Remains blinking for 5 seconds | - |

Table 163: Description of the Diagnostic LEDs States

#### 6.1.1.5. WD (Watchdog)

| Color | Status | Description | Causes | Priority |
|---|---|---|---|---|
| Red | Off | No watchdog indication | Normal operation | - |
| | On | Hardware watchdog | Damaged module and/or corrupted OS | 0 (higher) |
| | Blinking 1x | Software watchdog | User application watchdog | 1 |
| Green | On | Signature missing | Factory signature missing | - |
| | Off | Signature ok | Normal operation | - |

Table 164: Description of the Watchdog LED States

**Notes:**

**Software Watchdog:** In order to remove the watchdog indication, make an application reset or turn off and turn on the CPU again. This watchdog occurs when the user application execution time is higher than the configured watchdog time.

The diagnostics can be checked in the Exception.wExceptionCode variable, see on Table 178.

**Hardware Watchdog:** In order to reset any watchdog indication, as in the WD LED or in the Reset.bWatchdogReset operand, the module must be disconnected from the power supply.

In order to verify the application conditions in the module restart, see configurations on Table 37.

#### 6.1.1.6. PWR (Power Supply)

| Color | Status | Description |
|---|---|---|
| Green | Off | No power supply |
| | On | Connected power supply |

Table 165: PWR LED Status Description

#### 6.1.1.7. CAN

| Color | Status | Description |
|---|---|---|
| Green | Off | CAN is not being used or no data exchange |
| | Blink | CAN transmitting or receiving data |

Table 166: CAN LED Status Description

#### 6.1.1.8. RS485

| Color | Status | Description |
|---|---|---|
| Green | Off | RS485 is not being used or no data exchange |
| | Blink | RS485 transmitting or receiving data |

Table 167: RS485 LED Status Description

#### 6.1.1.9. USB

| Color | Status | Description |
|---|---|---|
| Green | Off | USB not mounted/USB not connected |
| | On | USB mounted |

Table 168: USB LED Status Description

#### 6.1.1.10. RJ45 Connector LEDs

Both LEDs placed in the RJ45 connectors, help the user in the installed physical network problem detection, indicating the network Link speed and the existence of interface communication traffic. The LEDs meaning is presented on table below.

| Yellow | Green | Meaning |
|---|---|---|
| ○ | ○ | Network LINK absent |
| ● | ○ | 10 Mbytes/s network LINK |
| ● | ● | 100 Mbytes/s network LINK |
| X | - | Ethernet network transmission or reception occurrence, for or to this IP address. Blinks on Nexto CPU demand and not every transmission or reception, in other words, it may blink on a lower frequency than the real transmission or reception frequency. |

Table 169: Ethernet LEDs Meaning

#### 6.1.1.11. DO (Digital Output)

| Color | Status | Description |
|---|---|---|
| Green | Off | The correspondent digital output is OFF. |
| | On | The correspondent digital output is ON. |

Table 170: Digital Output LED Status Description

#### 6.1.1.12. DI (Digital Input)

| Color | Status | Description |
|---|---|---|
| Green | Off | The correspondent digital input is OFF |
| | On | The correspondent digital input is ON |

Table 171: Digital Input LED Status Description

#### 6.1.1.13. AO (Analog Output)

| Color | Status | Description |
|---|---|---|
| Green | Off | The correspondent analog output is NOT configured. |
| | On | The correspondent analog output is configured. |

Table 172: Analog Output LED Status Description

#### 6.1.1.14. AI (Analog Input)

| Color | Status | Description |
|---|---|---|
| Green | Off | The correspondent analog input is NOT configured. |
| | On | The correspondent analog input is configured. |

Table 173: Analog Input LED Status Description

### 6.1.2. Diagnostics via Variables

Nexto XF controllers offers a set of global symbolic variables, which provides several diagnostics information related to the hardware and software. These symbolic data structures are automatically created by Mastertool.

#### 6.1.2.1. Summarized Diagnostics

The following table shows the meaning of summarized diagnostics:

| DG_CPU.tSummarized.* | Type | Description |
|---|---|---|
| bHardwareFailure | BOOL | TRUE – Controller has internal hardware failure. |

| DG_CPU.tSummarized.* | Type | Description |
|---|---|---|
| | | FALSE – The hardware is working properly. |
| bSoftwareException | BOOL | TRUE – One or more exceptions generated by the software. |
| | | FALSE – No exceptions generated in the software. |
| bMemoryCardError | BOOL | TRUE – The memory card is inserted in the CPU, but is not working properly. |
| | | FALSE – The memory card is working properly. |
| bCOM1ConfigError | BOOL | TRUE – Error during/after the COM 1 serial interface configuration. |
| | | FALSE – Correct COM 1 serial interface configuration. |
| bNET1ConfigError | BOOL | TRUE – Error during/after the NET 1 Ethernet interface configuration. |
| | | FALSE – Correct NET 1 Ethernet interface configuration. |
| bNET2ConfigError | BOOL | TRUE – Error during/after the NET 2 Ethernet interface configuration. |
| | | FALSE – Correct NET 2 Ethernet interface configuration. |
| bInvalidDateTime | BOOL | TRUE – Invalid date/hour. |
| | | FALSE – Correct date/hour. |
| bRuntimeReset | BOOL | TRUE – The RTS (Runtime System) has been restarted at least once. This diagnostics is only cleared during the system restart. |
| | | FALSE – The RTS (Runtime System) is operating normally. |
| bRetentivityLost | BOOL | TRUE - Error occurred while saving the retentive data. |
| | | FALSE – Valid data in the retentive memory during start up. |
| bIntegratedIODiagnostic | BOOL | TRUE - There is some diagnostic in the Integrated I/O (see detailed) |
| | | FALSE – No diagnostic in the Integrated I/O |
| bUSBDiagnostic | BOOL | TRUE - There is some diagnostic in the USB (see detailed) |
| | | FALSE – No diagnostic in the USB |
| bOTDSwitchError | BOOL | TRUE – True in case the OTD key has been locked for more than 20 s at least once while the CPU was energized. This diagnostic is only cleared in the system restart. |
| | | FALSE – The key is not currently locked or was locked while the CPU was energized. |

| DG_CPU.tSummarized.* | Type | Description |
|---|---|---|

Table 174: Summarized Diagnostics

**Notes:**

**Hardware Failure:** In case the Hardware Failure diagnostic is true, the controller must be sent to Altus Technical Assistance, as it has problems in the RTC or other hardware resources.

**Software Exception:** In case the software exception diagnostic is true, the user must verify his application to guarantee it is not accessing the memory wrongly. If the problem remains, the Altus Technical Support sector must be consulted. The software exception codes are described next in the controller's detailed diagnostics table.

**Retentivity Error:** If Retentive error flag is true, Altus Technical Support must be consulted. *Reset Cold* and *Reset Origin* commands triggered by MasterTool does not cause the indication of this diagnostic.

#### 6.1.2.2. Detailed Diagnostics

The tables below contain Nexto Series' CPUs detailed diagnostics. It is important to have in mind the observations below before consulting them:

- **Visualization of the Diagnostics Structures:** The Diagnostics Structures added to the Project can be seen at the item "*Library Manager*" of Mastertool tree view. There, it is possible to see all data types defined in the structure.
- **Counters:** All CPU diagnostics counters return to zero when their limit value is exceeded.
- *DG_Module*, where the word Module must be replaced by the product being used.

##### *6.1.2.2.1. Target Detailed Diagnostics Group*

| DG_Module.tDetailed.Target.* | Size | Description |
|---|---|---|
| dwCPUModel | DWORD | CPU model. |
| abyCPUVersion | BYTE ARRAY(4) | Firmware version. |
| abyBootloaderVersion | BYTE ARRAY(4) | Bootloader version. |
| abyAnalogprocVersion | BYTE ARRAY(4) | Analog IO coprocessor version. |
| abyBusControllerVersion | BYTE ARRAY(4) | Bus controller version. |
| abyFastOutputVersion | BYTE ARRAY(4) | Fast output version. |

Table 175: Target Detailed Diagnostics Group Description

##### *6.1.2.2.2. Hardware Detailed Diagnostics Group*

| DG_Module.tDetailed.Hardware.* | Size | Description |
|---|---|---|
| bRTCFailure | BIT | The main processor is not enabled to communicate with the RTC (CPU's clock). |
| bIntegratedIoFailure | BIT | Failure in the communication with any Integrated IO processor. |

Table 176: Hardware Detailed Diagnostics Group Description

#### 6.1.2.2.3. Exception Detailed Diagnostics Group

| DG_Module.tDetailed.Exception.* | Size | Description |
|---|---|---|
| wExceptionCode | WORD | Exception code generated by the RTS. See Table 178. |
| byProcessorLoad | BYTE | Level, in percentage (%), of charge in the processor. |

Table 177: Exception Detailed Diagnostics Group Description

**Note:**
**Exception Code:** the code of the exception generated by the RTS (Runtime System) can be consulted below:

| Code | Description | Code | Description |
|---|---|---|---|
| **0x0000** | There is no exception code. | **0x0051** | Access violation. |
| **0x0010** | Watchdog time of the IEC task expired (Software Watchdog). | **0x0052** | Privileged instruction. |
| **0x0012** | I/O configuration error. | **0x0053** | Page failure. |
| **0x0013** | Checksum error after the program download. | **0x0054** | Stack overflow. |
| **0x0014** | Fieldbus error. | **0x0055** | Invalid disposition. |
| **0x0015** | I/O updating error. | **0x0056** | Invalid maneuver. |
| **0x0016** | Cycle time (execution) exceeded. | **0x0057** | Protected page. |
| **0x0017** | Program online updating too long. | **0x0058** | Double failure. |
| **0x0018** | External references not resolved. | **0x0059** | Invalid OpCode. |
| **0x0019** | Download rejected. | **0x0100** | Data type misalignment. |
| **0x001A** | Project not loaded, as the retentive variables cannot be reallocated. | **0x0101** | Arrays limit exceeded. |
| **0x001B** | Project not loaded and deleted. | **0x0102** | Division by zero. |
| **0x001C** | Out of memory stack. | **0x0103** | Overflow. |
| **0x001D** | Retentive memory is corrupted and cannot be mapped. | **0x0104** | Cannot be continued. |
| **0x001E** | Project can be loaded but causes a drop later on. | **0x0105** | Watchdog in the processor load of all IEC task detected. |
| **0x0021** | Target of startup application does not match to the current target. | **0x0150** | FPU: Not specified error. |
| **0x0022** | Scheduled tasks error. | **0x0151** | FPU: Operand is not normal. |
| | | **0x0152** | FPU: Division by zero. |
| **0x0023** | Downloaded file Checksum with error. | **0x0153** | FPU: Inexact result. |
| **0x0024** | Retentive identity is not correspondent to the current identity of the boot project program | **0x0154** | FPU: Invalid operation. |
| **0x0025** | IEC task configuration failure. | **0x0155** | FPU: Overflow. |
| **0x0026** | Application working with wrong target. | **0x0156** | FPU: Stack verification. |
| **0x0050** | Illegal instruction. | **0x0157** | FPU: Underflow. |

Table 178: RTS Exception codes

#### 6.1.2.2.4. WebVisualization Detailed Diagnostics Group

| DG_Module.tDetailed.WebVisualization.* | Size | Description |
|---|---|---|
| byConnectedClients | BYTE | Clients number connected to the WebVisualization. |

Table 179: WebVisualization Detailed Diagnostics Group Description

#### 6.1.2.2.5. RetainInfo Detailed Diagnostics Group

| DG_Module.tDetailed.RetainInfo.* | Size | Description |
|---|---|---|
| byCPUInitStatus | BYTE | CPU Startup Status:<br>01: Hot Start<br>02: Warm Start<br>03: Cold Start<br>Note: These variables are reset in every powerup. |
| wCPUColdStartCounter | WORD | Increments when the CPU starts with loss of retentivity. |
| wCPUWarmStartCounter | WORD | Increments when the CPU starts normally with valid retain data. |
| wRTSResetCounter | WORD | Counter of resets performed by RTS (Runtime System). |
| wWritesCounter | WORD | Retentive memory writes counter. |

Table 180: RetainInfo Group Detailed Diagnostics

#### 6.1.2.2.6. Reset Detailed Diagnostics Group

| DG_Module.tDetailed.Reset.* | Size | Description |
|---|---|---|
| bWatchdogReset | BIT | The CPU was restarted due the active watchdog in the last startup. |

Table 181: Reset Detailed Diagnostics Group Description

#### 6.1.2.2.7. Thermometer Detailed Diagnostics Group

| DG_Module.tDetailed.Thermometer.* | Size | Description |
|---|---|---|
| bOverTemperatureAlarm | BIT | Alarm generated due internal temperature at 85 °C or above it. |
| bUnderTemperatureAlarm | BIT | Alarm generated due internal temperature at 0 °C or under it. |
| diTemperature | DINT | Temperature read in the internal sensor of the CPU. |

Table 182: Thermometer Detailed Diagnostics Group Description

**Note:**

**Temperature:** In order to see the temperature directly in the memory address, a conversion must be made, since the data size is DINT and monitoring is done in 4 bytes. Therefore, it's recommended to use the associated symbolic variable, because it already provides the final temperature value.

#### 6.1.2.2.8. Serial Detailed Diagnostics Group

| DG_Module.tDetailed.Serial.COM1.* | Size | Description |
|---|---|---|
| byProtocol | BYTE | Protocol selected in the COM 1:<br>00: Without protocol<br>01: MODBUS RTU Master<br>02: MODBUS RTU Slave<br>03: Other protocol |
| dwRXBytes | DWORD | Counter of characters received from COM 1 (0 to 4294967295). |
| dwTXBytes | DWORD | Counter of characters transmitted from COM 1 (0 to 4294967295). |
| wRXPendingBytes | WORD | Number of characters left in the reading buffer in COM 1 (0 to 1024). |
| wTXPendingBytes | WORD | Number of characters left in the transmission buffer in COM 1 (0 to 1024). |
| wBreakErrorCounter | WORD | The transmitter is holding the data line at zero for too long, according to the databit configured. |
| wParityErrorCounter | WORD | The received frame has the mismatched parity bit. |
| wFrameErrorCounter | WORD | The received frame has the wrong start point, usually caused by a noise or baud rate mismatch. |
| wRXOverrunCounter | WORD | When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device). |

Table 183: Serial COM 1 Detailed Diagnostics Group Description

**Note:**

**Parity Error Counter:** When the serial COM 1 is configured Without Parity, this error counter won't be incremented when it receives a message with a different parity. In this case, a frame error will be indicated.

#### 6.1.2.2.9. CAN Detailed Diagnostics Group

| DG_Module.tDetailed.CAN.* | Size | Description |
|---|---|---|
| bBusAlarm | BIT | The bus has a critical error and is off. |
| byBusState | BYTE | Reports the device status. |
| udiTxCounter | UDINT | Number of packets (Tx) changed on the CAN bus of the CPU. |
| udiRxCounter | UDINT | Number of packets (Rx) changed on the CAN bus of the CPU. |
| udiTxErrorCounter | UDINT | Number of packets (Tx) with errors on the CPU CAN bus. |
| udiRxErrorCounter | UDINT | Number of packets (Rx) with errors on the CPU CAN bus. |
| udiLostCounter | UDINT | Number of packets lost on the CAN bus of the CPU. |

Table 184: CAN Detailed Diagnostics Group Description

#### 6.1.2.2.10. USB Detailed Diagnostics Group

| DG_Module.tDetailed.USB.* | Size | Description |
|---|---|---|
| byUSBDevice | BYTE | Type of device connected to the USB port. |
| bOvercurrent | BIT | The device connected to the USB port is drawing more current than supported. |
| bEnable | BIT | The USB port is enabled. |
| tMassStorage.byMountState | BYTE | Reports the device status. |
| tMassStorage.dwFreeSpaceKb | DWORD | Reports the free space on the mass storage device. |
| tMassStorage.dwTotalSizeKb | DWORD | Reports the total size of the mass storage device. |
| tSerialConverter.byProtocol | BYTE | Protocol selected in the COM 1:<br>00: Without protocol<br>01: MODBUS RTU Master<br>02: MODBUS RTU Slave<br>03: Other protocol |
| tSerialConverter.dwRXBytes | DWORD | Received character counter on COM 10 (0 to 4294967295). |
| tSerialConverter.dwTXBytes | DWORD | Counter of characters transmitted by COM 10 (0 to 4294967295). |
| tSerialConverter.wRXPendingBytes | WORD | Number of characters remaining in the read buffer on COM 10 (0 to 4095). |
| tSerialConverter.wTXPendingBytes | WORD | Number of characters remaining in the transmit buffer on COM 10 (0 to 1023). |
| tSerialConverter.wBreakErrorCounter | WORD | The transmitter is holding the data line at zero for too long, according to the databit configured. |
| tSerialConverter.wParityErrorCounter | WORD | The received frame has the mismatched parity bit. |
| tSerialConverter.wFrameErrorCounter | WORD | The received frame has the wrong start point, usually caused by a noise or baud rate mismatch. |
| tSerialConverter.wRXOverrunCounter | WORD | When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device). |

Table 185: USB Detailed Diagnostics Group Description

#### 6.1.2.2.11. Ethernet Detailed Diagnostics Group

| DG_Module.tDetailed.Ethernet.* | Size | Description |
|---|---|---|
| NET[x].bLinkDown | BIT | Indicates link state on NET[x]. |
| NET[x].byOperatingMode | BYTE | Indicates the operating mode of the interface NET[x]. |
| NET[x].byOperatingState | BYTE | Indicates the operating state of the interface NET[x]. |
| NET[x].szIP | STRING (15) | NET[x] IP Address. |
| NET[x].szMask | STRING (15) | NET[x] Subnet Mask. |
| NET[x].szGateway | STRING (15) | NET[x] Gateway Address. |
| NET[x].szMAC | STRING (17) | NET[x] MAC Address. |
| NET[x].abyIP | BYTE ARRAY(4) | NET[x] IP Address. |
| NET[x].abyMask | BYTE ARRAY(4) | NET[x] Subnet Mask. |
| NET[x].abyGateway | BYTE ARRAY(4) | NET[x] Gateway Address. |
| NET[x].abyMAC | BYTE ARRAY(6) | NET[x] MAC Address. |

| DG_Module.tDetailed.Ethernet.* | Size | Description |
|---|---|---|
| NET[x].NICTeaming.bStandbyState | BIT | Indicates the interface is in standby state. |
| NET[x].NICTeaming.bActiveState | BIT | Indicates the interface is in active state. |
| NET[x].NICTeaming.bLinkDown | BIT | Indicates the network link is down. |
| NET[x].NICTeaming.InterMsgTimeout | BIT | Indicates no communication with the other interface. |
| NET[x].NICTeaming.GeneralRxMsgTimeout | BIT | Indicates no network communication activity. |
| NET[x].dwTransmittedBytes | DWORD | Counter of bytes sent via the NET[x] port (0 to 4294967295). |
| NET[x].dwTransmittedPackets | DWORD | Counter of packets sent via the NET[x] port (0 to 4294967295). |
| NET[x].dwTransmittedDropErrors | DWORD | Connection loss counter on transmission over the NET[x] port (0 to 4294967295). |
| NET[x].dwTransmittedCollisionErrors | DWORD | Transmission collision error counter over NET[x] port (0 to 4294967295). |
| NET[x].dwReceivedBytes | DWORD | Counter of bytes received via the NET[x] port (0 to 4294967295). |
| NET[x].dwReceivedPackets | DWORD | Counter of packets received via NET[x] port (0 to 4294967295). |
| NET[x].dwReceivedDropErrors | DWORD | Connection loss counter on reception via the NET[x] port (0 to 4294967295). |
| NET[x].dwReceivedFrameErrors | DWORD | Frame error counter on reception over NET[x] port (0 to 4294967295). |

Table 186: Ethernet Group Detailed Diagnostics NET[x]

**Note:**
**[x]** is the ethernet interface number of the CPU, where for example, NET 1 represents the value 1.

| DG_Module.tDetailed.Ethernet.RSTP.* | Size | Description |
|---|---|---|
| RSTP[x].bProtocolEnabled | BIT | Indicates if the RSTP protocol is enabled. |
| RSTP[x].bBridgeIsRoot | BIT | Indicates if the Bridge is root. |
| RSTP[x].eProtocol | BYTE | Indicates the protocol version. |
| RSTP[x].sBridgeId | STRING (22) | Local bridge ID (priority and MAC). |
| RSTP[x].sRootId | STRING (22) | Root bridge ID (priority and MAC). |
| RSTP[x].eRootPort | BYTE | Indicates which is the root port. |
| RSTP[x].dwRootPathCost | DWORD | Path Cost to the bridge root. |
| RSTP[x].byBridgeHelloTime | BYTE | Hello Time configured. |
| RSTP[x].byRootHelloTime | BYTE | Hello Time learned. |
| RSTP[x].byBridgeMaxAge | BYTE | Max Age configured. |
| RSTP[x].byRootMaxAge | BYTE | Max Age learned. |
| RSTP[x].byBridgeForwDelay | BYTE | Forward Delay configured. |
| RSTP[x].byRootForwDelay | BYTE | Forward Delay learned. |
| RSTP[x].dwTxHoldCount | DWORD | Limit of BPDUs transmitted per cycle. |
| RSTP[x].dwTimeSinceTC | DWORD | Time, in seconds, since the last topology change. |
| RSTP[x].dwTCCount | DWORD | Number of times the topology has been changed. |
| RSTP[x].aPort[y].eName | BYTE | Port name. |
| RSTP[x].aPort[y].eRole | BYTE | Port function. |
| RSTP[x].aPort[y].eState | BYTE | Port state. |

| DG_Module.tDetailed.Ethernet.RSTP.* | Size | Description |
|---|---|---|
| RSTP[x].aPort[y].wIdentity | WORD | Port ID. |
| RSTP[x].aPort[y].dwPathCost | DWORD | Port cost. |
| RSTP[x].aPort[y].bOperEdge | BIT | Indicates if the port is working as Edge. |
| RSTP[x].aPort[y].bOperP2P | BIT | Indicates if the port is working as Point-to-Point. |
| RSTP[x].aPort[y].dwNumTxBPDU | DWORD | Number of BPDUs frames transmitted on the port. |
| RSTP[x].aPort[y].dwNumRxBPDU | DWORD | Number of frames BPDUs received on the port. |

Table 187: RSTP Detailed Diagnostics Group Description

**Note:**

**[x]:** is the bridge number.

**[y]:** is the port number within the bridge.

#### *6.1.2.2.12. UserFiles Detailed Diagnostics Group*

| DG_Module.tDetailed.UserFiles.* | Size | Description |
|---|---|---|
| byMounted | BYTE | Indicates if the memory used for recording user files is able to receive data. |
| dwFreeSpacekB | DWORD | Free memory space for user files in Kbytes. |
| dwTotalSizekB | DWORD | Storage capacity of the memory of user files in Kbytes. |

Table 188: UserFiles Detailed Diagnostics Group Description

**Note:**

**User Partition:** The user partition is a memory area reserved for the storage of data in the CPU. For example: files with PDF extension, files with DOC extension and other data.

#### *6.1.2.2.13. UserLogs Detailed Diagnostics Group*

| DG_Module.tDetailed.UserLogs.* | Size | Description |
|---|---|---|
| byMounted | BYTE | Status of the memory where user logs are inserted. |
| wFreeSpacekB | WORD | User log memory free space in Kbytes. |
| wTotalSizekB | WORD | User logs memory storage capacity in Kbytes. |

Table 189: UserLogs Detailed Diagnostics Group Description

#### *6.1.2.2.14. MemoryCard Detailed Diagnostics Group*

| DG_Module.tDetailed.MemoryCard.* | Size | Description |
|---|---|---|
| byMounted | BYTE | Status of the Memory Card:<br>00: Memory card not mounted<br>01: Memory card inserted and mounted |
| bEnable | BIT | Memory Card enabled. |
| dwFreeSpacekB | DWORD | Free space in the Memory Card in Kbytes. |

| DG_Module.tDetailed.MemoryCard.* | Size | Description |
|---|---|---|
| dwTotalSizekB | DWORD | Storage capacity of the Memory Card in Kbytes. |

Table 190: MemoryCard Detailed Diagnostics Group Description

#### 6.1.2.2.15. Application Detailed Diagnostics Group

| DG_Module.tDetailed.Application.* | Size | Description |
|---|---|---|
| byCPUState | BYTE | Informs the operation state of the CPU:<br>01: All user applications are in Run Mode<br>03: All user applications is in Stop Mode |
| bForcedIOs | BIT | There is one or more forced I/O points. |
| bNetDefinedByWeb | BIT | The IP address is set by the System Web Page. |

Table 191: Application Detailed Diagnostics Group Description

#### 6.1.2.2.16. ApplicationInfo Detailed Diagnostics Group

| DG_Module.tDetailed.ApplicationInfo.* | Size | Description |
|---|---|---|
| dwApplicationCRC | DWORD | 32-bit CRC of the application. When the application is modified and sent to the CPU, a new CRC is calculated. |

Table 192: ApplicationInfo Detailed Diagnostics Group Description

#### 6.1.2.2.17. SNTP Detailed Diagnostics Group

| DG_Module.tDetailed.SNTP.* | Size | Description |
|---|---|---|
| bServiceEnabled | BIT | SNTP service enabled. |
| byActiveTimeServer | BYTE | Indicates which server is active:<br>00: No active server.<br>01: Primary server active.<br>02: Secondary server active. |
| wPrimaryServerDownCount | WORD | Count of times the primary server was unavailable (0 to 65535). |
| wSecondaryServerDownCount | WORD | Count of times the secondary server was unavailable (0 to 65535). |
| dwRTCTimeUpdatedCount | DWORD | Count of times the RTC was updated by the SNTP service (0 to 4294967295). |
| byLastUpdateSuccessful | BYTE | Indicates status of last update:<br>00: Not updated.<br>01: Last update failed.<br>02: Last update was successful. |
| byLastUpdateTimeServer | BYTE | Indicates which server was used in the last update:<br>00: No updates.<br>01: Primary server.<br>02: Secondary server. |

| DG_Module.tDetailed.SNTP.* | Size | Description |
|---|---|---|
| sLastUpdateTime.byDayOfMonth | BYTE | Day of last RTC update. |
| sLastUpdateTime.byMonth | BYTE | Month of last RTC update. |
| sLastUpdateTime.wYear | WORD | Year of last update of RTC. |
| sLastUpdateTime.byHours | BYTE | Hour of last RTC update. |
| sLastUpdateTime.byMinutes | BYTE | Minute of last RTC update. |
| sLastUpdateTime.bySeconds | BYTE | Second of last RTC update. |
| sLastUpdateTime.wMilliseconds | WORD | Millisecond of last RTC update. |

Table 193: SNTP Group Detailed Diagnostics

#### *6.1.2.2.18. Integrated IO Detailed Diagnostics Group*

| DG_Module.tDetailed.IntegratedIO.* | Size | Description |
|---|---|---|
| AnalogInputs.tAnalogInput_xx.bInputNotEnabled | BIT | The input channel is not enabled on the configuration. |
| AnalogInputs.tAnalogInput_xx.bOverRange | BIT | The input signal level is above the maximum value defined for the selected input type. |
| AnalogInputs.tAnalogInput_xx.bOpenLoop | BIT | The impedance of the load connected to the input channel is above the maximum accepted (only for current inputs [AI0 to AI5]). |
| AnalogOutputs.tAnalogOutput_xx.bOutputNotEnabled | BIT | The output channel is not enabled on the configuration. |
| AnalogOututs.tAnalogOutput_xx.bOpenLoop | BIT | The impedance of the load connected to the output channel is above the maximum accepted (only for current output mode). |
| AnalogOututs.tAnalogOutput_xx.bShortCircuit | BIT | The impedance of the load connected to the output channel is below the minimum accepted (only for voltage output mode). |

Table 194: Integrated I/O Group Detailed Diagnostics

#### *6.1.2.2.19. OpenVPN Detailed Diagnostics Group*

| DG_Modulo.tDetailed.OpenVPN.* | Size | Description |
|---|---|---|
| byOperationMode | BYTE | VPN operating mode:<br>SERVER(0): Device operating as a server.<br>CLIENT(1): Device operating as a client. |
| bServiceEnabled | BIT | VPN service enabled |
| bServiceRunning | BIT | VPN service running. |
| byConnectionState | BYTE | VPN connection status:<br>DISCONNECTED(0): Device disconnected.<br>CONNECTING(1): Device connecting.<br>CONNECTED(2): Device connected. |
| uliConnectionTime | ULINT | Current connection time in seconds. |
| sIPAddress | STRING(15) | VPN IP address. |
| dwConnectedClients | DWORD | Number of connected clients. |
| dwTransmittedBytes | DWORD | Number of bytes transmitted through the VPN. |
| dwReceivedBytes | DWORD | Number of bytes received through the VPN. |

| DG_Modulo.tDetailed.OpenVPN.* | Size | Description |
|---|---|---|
| CACertificate.CertificateName | STRING(64) | CA certificate name. |
| CACertificate.CommonName | STRING(64) | CA certificate common name. |
| CACertificate.sStartDate | EXTENDED_DATE_AND_TIME | CA certificate start date and time. |
| CACertificate.sExpirationDate | EXTENDED_DATE_AND_TIME | CA certificate expiration date and time. |
| CACertificate.dwValidDaysLeft | DWORD | Days until CA certificate expiration. |
| DeviceCertificate.CertificateName | STRING(64) | Device certificate name. |
| DeviceCertificate.CommonName | STRING(64) | Device certificate common name. |
| DeviceCertificate.sStartDate | EXTENDED_DATE_AND_TIME | Device certificate start date and time. |
| DeviceCertificate.sExpirationDate | EXTENDED_DATE_AND_TIME | Device certificate expiration date and time. |
| DeviceCertificate.dwValidDaysLeft | DWORD | Days until device certificate expiration. |
| sDeviceKeyName | STRING(64) | Device private key name. |

Table 195: Detailed Diagnostics OpenVPN Group

#### *6.1.2.2.20. Firewall Detailed Diagnostics Group*

| DG_Module.tDetailed.Firewall.* | Size | Description |
|---|---|---|
| bServiceEnabled | BIT | The Firewall service is enabled. |
| sLatsModificationDateUTC | NextoStandard.<br>EXTENDED_DATE_AND_TIME | Date and time of the last configuration change in UTC. |

Table 196: Firewall Detailed Diagnostics Group Description

#### *6.1.2.2.21. FTP Detailed Diagnostics Group*

| DG_Module.tDetailed.FTP.* | Size | Description |
|---|---|---|
| bServiceEnabled | BIT | FTP service enabled. |
| dwConnectedClients | DWORD | Number of connected clients. |
| sLastUpdateTime.byDayOfMonth | BYTE | Day of the last FTP update. |
| sLastUpdateTime.byMonth | BYTE | Month of the last FTP update. |
| sLastUpdateTime.wYear | WORD | Year of the last FTP update. |
| sLastUpdateTime.byHours | BYTE | Hour of the last FTP update. |
| sLastUpdateTime.byMinutes | BYTE | Minute of the last FTP update. |
| sLastUpdateTime.bySeconds | BYTE | Second of the last FTP update. |
| sLastUpdateTime.wMilliseconds | WORD | Millisecond of the last FTP update. |

Table 197: Detailed Diagnostics FTP Group

## 6.2. PLC User Button

The button is placed on the controller upper part, in an easy-access place. The figure below shows its placement:

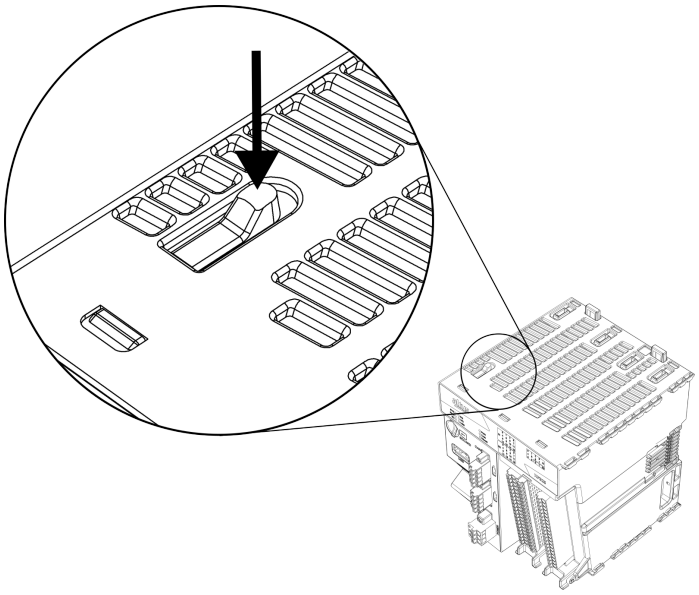


Figure 213: XF3xx button

| Function | Activation |
|---|---|
| Don't load application / Don't apply firewall | Hold the button along with energization until SYS LED remains ON |
| Unmount Memory Card (safe ejection) | Long press → SD card LED will turn off |

Table 198: Button function description

| Touch type | Minimum time | Maximum time | Indication condition |
|---|---|---|---|
| No touch | - | 59.99 ms | - |
| Short touch | 60 ms | 0.99 s | Release |
| Long touch | 1 s | 20 s | More than 1 s till 20 s |
| Locked Switch | 20.01 s | ($\infty$) | Diagnostics indication, see on Table 174 |

Table 199: One Touch Time

### 6.2.1. Diagnostics via Function Blocks

The function blocks allow the visualization of some parameters which cannot be accessed otherwise. The function regarding advanced diagnostics is in the *NextoStandard* library and is described below.

#### 6.2.1.1. GetTaskInfo

This function returns the task information of a specific application.


Figure 214: GetTaskInfo Function

Below, the parameters that must be sent to the function for it to return the application information are described.

| Input parameter | Type | Description |
|---|---|---|
| **psAppName** | POINTER TO STRING | Application name. |
| **psTaskName** | POINTER TO STRING | Task name. |
| **pstTaskInfo** | POINTER TO stTaskInfo | Pointer to receive the application information. |

Table 200: GetTaskInfo Input Parameters

The data returned by the function, through the pointer informed in the input parameters are described on table below.

| Returned Parameters | Size | Description |
|---|---|---|
| **dwCurScanTime** | DWORD | Task cycle time (execution) with 1 $\mu$s resolution. |
| **dwMinScanTime** | DWORD | Task cycle minimum time with 1 $\mu$s resolution. |
| **dwMaxScanTime** | DWORD | Task cycle maximum time 1 $\mu$s resolution. |
| **dwAvgScanTime** | DWORD | Task cycle average time with 1 $\mu$s resolution. |
| **dwLimitMaxScan** | DWORD | Task cycle maximum time before watchdog occurrence. |
| **dwIECCycleCount** | DWORD | IEC cycle counter. |

Table 201: GetTaskInfo Output Parameters

Possible ERRORCODE:

- NoError: success execution;
- TaskNotPresent: the desired task does not exist.

Example of utilization in ST language:

```
PROGRAM UserPrg
VAR
sAppName : STRING;
psAppName : POINTER TO STRING;
sTaskName : STRING;
psTaskName : POINTER TO STRING;
pstTaskInfo : POINTER TO NextoStandard.stTaskInfo;
TaskInfo :NextoStandard. stTaskInfo;
Info : NextoStandard.ERRORCODE;
END_VAR
//INPUTS:
sAppName := 'Application';    //Variable receives the application name.
psAppName := ADR(sAppName); //Pointer with application name.
sTaskName := 'MainTask';    //Variable receives task name.
psTaskName := ADR(sTaskName); //Pointer with task name.
pstTaskInfo := ADR(TaskInfo); //Pointer that receives task info.
//FUNCTION:
//Function call.
Info := GetTaskInfo (psAppName, psTaskName, pstTaskInfo);
//Variable Info receives possible function errors.
```

## 6.3. Preventive Maintenance

- It must be verified, each year, if the interconnection cables are connected firmly, without dust accumulation, mainly the protection devices.
- In environments subjected to excessive contamination, the equipment must be periodically cleaned from dust, debris, etc.
- Bus tightness and cleanness every six months.

# 7. Appendixes

## 7.1. TLS Key and Certificate Management

This section covers the generation of security files, certificates, and keys using TLS. The certificates commented on below are signed by CA. This type of certificate considers an entity, called Certificate Authority (CA), to generate the certificates. This entity can be an official authority service or a simple computer. It is only necessary to restrict access to the CA to avoid any security breach since this entity can generate certificates for any device. The image below shows how each device interacts with the files.

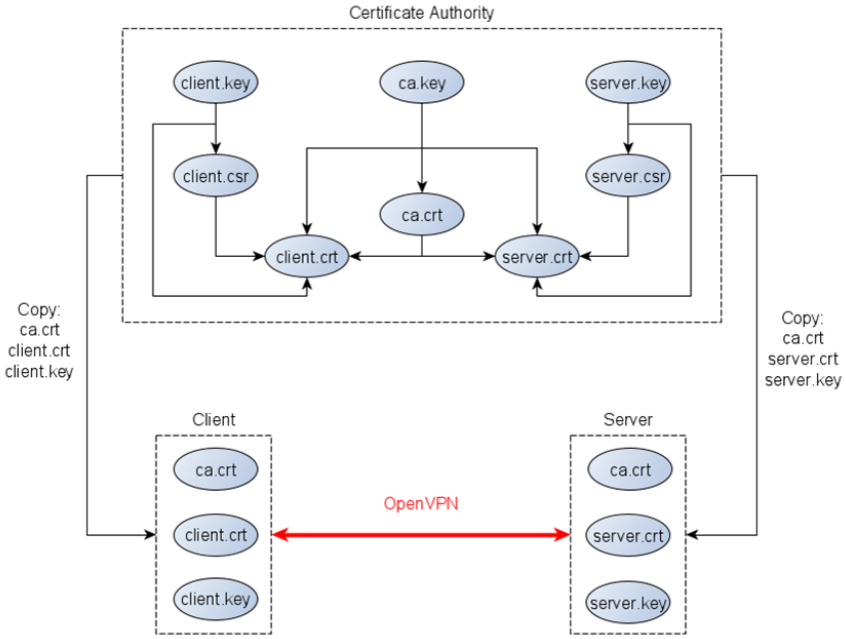



Figure 215: TLS Certificate Generation Flow

First of all, the generated files are private keys. Each device has its key file, created either by the CA entity or the device itself. The most important file is the CA private key *ca.key*, which must not leave the entity. The CA entity generates its certificate based on its private key *ca.crt*. This certificate is a public file used by the devices to validate the VPN connection. Generating certificates from the device first requires a request file (*.csr* or *.req* depending on the tool) based on the device's private key. This document presents two possible tools for generating certificate files: Easy-RSA and OpenSSL.

Make sure you have the date and time set correctly in the CA entity so that the generation of the certificates is based on a current setting.

### 7.1.1. Easy-RSA Certificate Generation

The OpenVPN project provides this tool to help with the certificate and keys. Easy-RSA is available for Windows and Linux. See below for step-by-step instructions to generate the files in a Windows configuration:

1- Open a Windows prompt in the Easy-RSA folder and run the following command to enter the tool shell:

```
.\EasyRSA-Start.bat
```

```
C:\Users\igor.franco\Downloads\EasyRSA-3.0.8-win64\EasyRSA-3.0.8>.\EasyRSA-Start.bat

Welcome to the EasyRSA 3 Shell for Windows.
Easy-RSA 3 is available under a GNU GPLv2 license.

Invoke './easyrsa' to call the program. Without commands, help is displayed.

EasyRSA Shell
# _
```

Figure 216: Certificate Generation using Easy-RSA (step 1)

2 - Copy the file *vars.example* and rename it to *vars* in the tools folder.

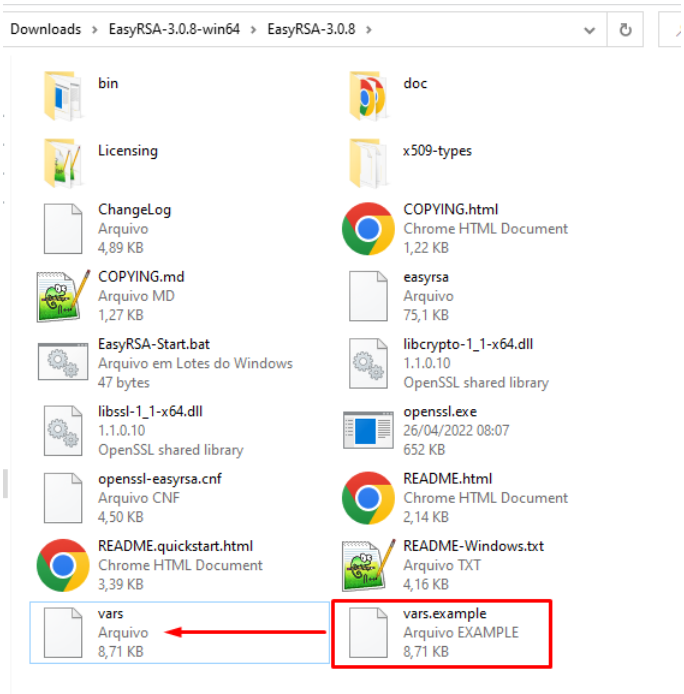
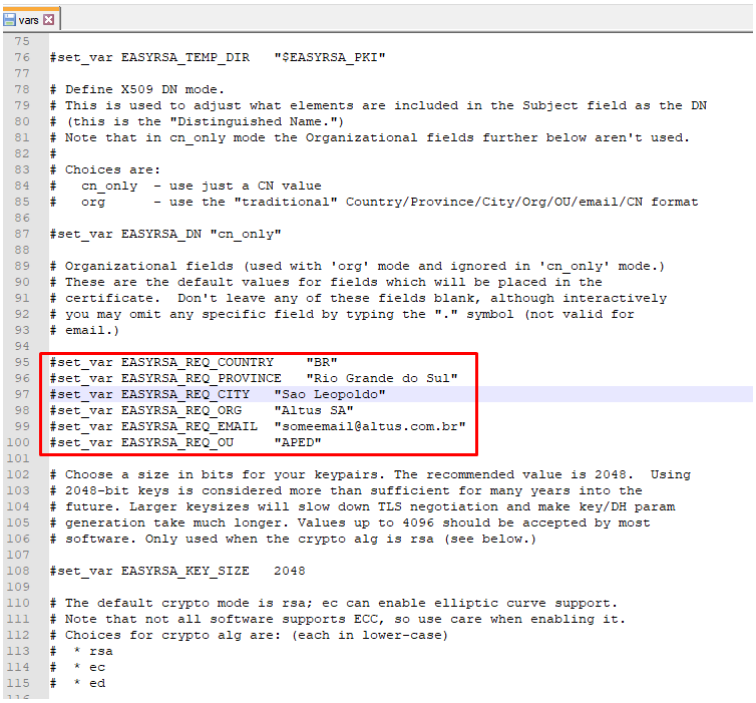



Figure 217: Certificate Generation using Easy-RSA (step 2)

3- Open the file *vars* with a text editor and change the Certification Authority information.



vars

```
#set_var EASYRSA_TEMP_DIR   "$EASYRSA_PKI"

# Define X509 DN mode.
# This is used to adjust what elements are included in the Subject field as the DN
# (this is the "Distinguished Name.")
# Note that in cn_only mode the Organizational fields further below aren't used.
#
# Choices are:
#   cn_only  - use just a CN value
#   org      - use the "traditional" Country/Province/City/Org/OU/email/CN format

#set_var EASYRSA_DN "cn_only"

# Organizational fields (used with 'org' mode and ignored in 'cn_only' mode.)
# These are the default values for fields which will be placed in the
# certificate.  Don't leave any of these fields blank, although interactively
# you may omit any specific field by typing the "." symbol (not valid for
# email.)

#set_var EASYRSA_REQ_COUNTRY    "BR"
#set_var EASYRSA_REQ_PROVINCE   "Rio Grande do Sul"
#set_var EASYRSA_REQ_CITY   "Sao Leopoldo"
#set_var EASYRSA_REQ_ORG    "Altus SA"
#set_var EASYRSA_REQ_EMAIL  "someemail@altus.com.br"
#set_var EASYRSA_REQ_OU     "APED"

# Choose a size in bits for your keypairs. The recommended value is 2048.  Using
# 2048-bit keys is considered more than sufficient for many years into the
# future. Larger keysizes will slow down TLS negotiation and make key/DH param
# generation take much longer. Values up to 4096 should be accepted by most
# software. Only used when the crypto alg is rsa (see below.)

#set_var EASYRSA_KEY_SIZE   2048

# The default crypto mode is rsa; ec can enable elliptic curve support.
# Note that not all software supports ECC, so use care when enabling it.
# Choices for crypto alg are: (each in lower-case)
#  * rsa
#  * ec
#  * ed
```

Figure 218: Certificate Generation using Easy-RSA (step 3)

4- Use the following command to prepare the configuration.

```
./easyrsa init-pki
```

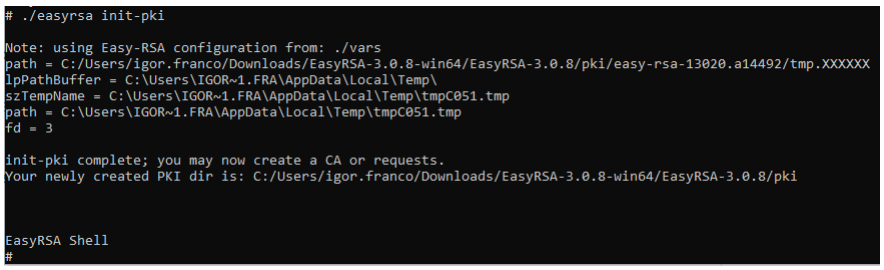



Figure 219: Certificate Generation using Easy-RSA (step 4)

5- Then type the following to generate the CA certificate.

```
./easyrsa build-ca nopass
```

Remove the *nopass* argument if you want to set a password for the file. Enter the common name of the CA certificate when prompted (press enter to use the default *Easy-RSA CA* as the common name).

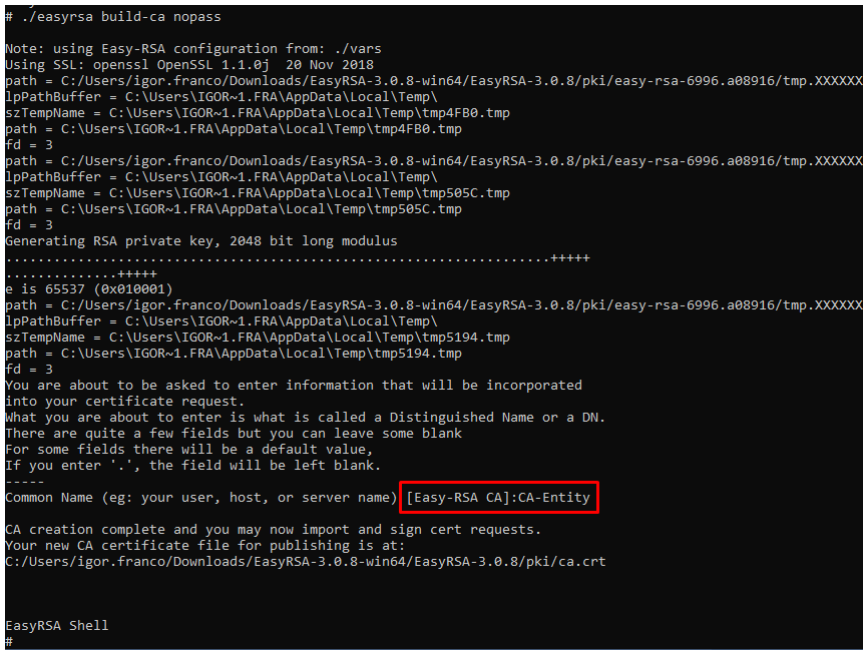



Figure 220: Certificate Generation using Easy-RSA (step 5)

6 - Generate the device key and request files using the following command (change the *DeviceName* with the desired common name):

```
./easyrsa gen-req DeviceName nopass
```

Again, remove the *nopass* argument to use a password for the certificate file. When entering the Common Name as an argument, simply press Enter when prompted (red square).

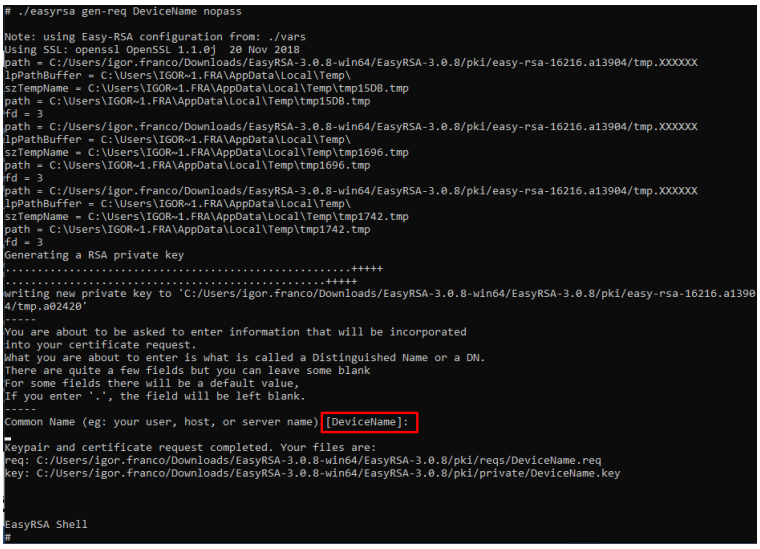



Figure 221: Certificate Generation using Easy-RSA (step 6)

7- Finally, type the following command to generate the device certificate (the *DeviceName* is the desired common name, and the *server* is the type; use *client* if you are generating for a VPN client).

```
./easyrsa sign-req server DeviceName
```

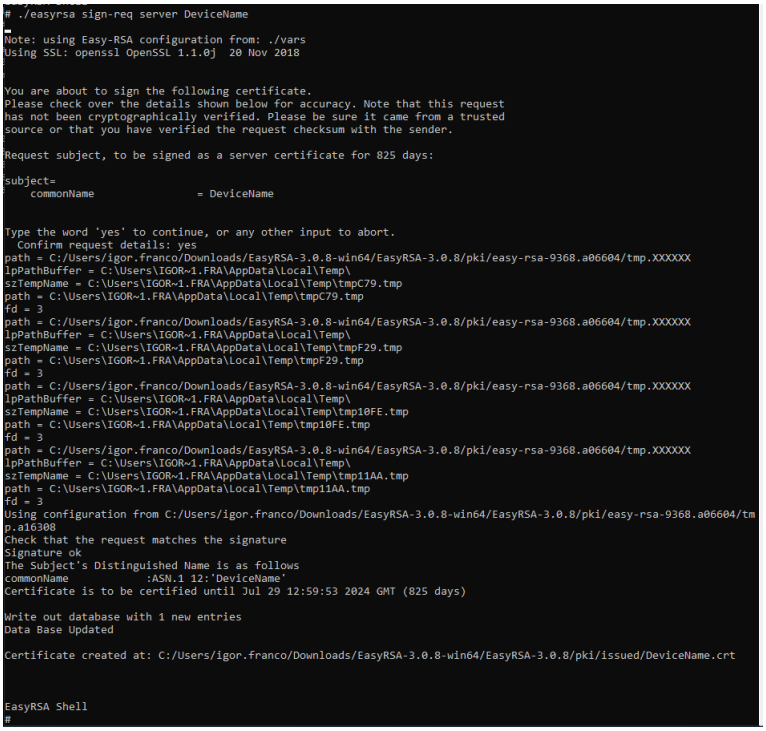



Figure 222: Certificate Generation using Easy-RSA (step 7)

8- Repeat steps 6 and 7 to generate more device certificates.

9- Find the *ca.crt* in the *pki* folder, the device private keys in the *pki/private* path, and the device certificates in the *pki/issued* directory.


Figure 223: Certificate Generation using Easy-RSA (step 9)

### 7.1.2. OpenSSL Certificate Generation

OpenSSL is an open-source package with tools that help generate many files and security features. This package is native to most Linux distributions and is available for Windows. Just remember to set the OpenSSL folder in the PATH (environment variable) to allow you to use the command from anywhere via the prompt. Find below the step-by-step using this feature (all files can have any name as desired, the steps consider only an example):

1- Open a prompt in the certificate folder (where you will create the files).

2- Generate the CA private key with the following command:

```
openssl genrsa -out ca.key 4096
```

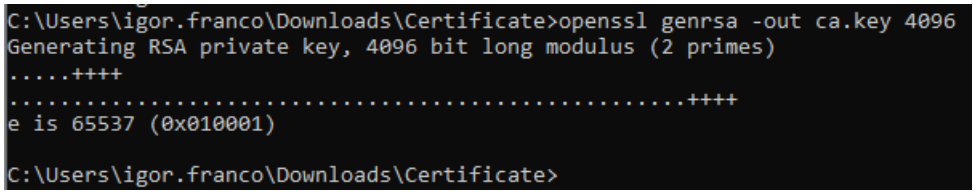



Figure 224: Certificate Generation using OpenSSL (step 2)

3- Then generate the CA certificate based on the private key, using the following command.

```
openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

The parameter *-days* represents the expiration time for the certificate. Set it as desired. In this example, the certificate is valid for one year. Fill in the values requested at the prompt as needed (press enter to use the default, which is enclosed in square brackets []). It is mandatory to define a Common Name for the certificate work.

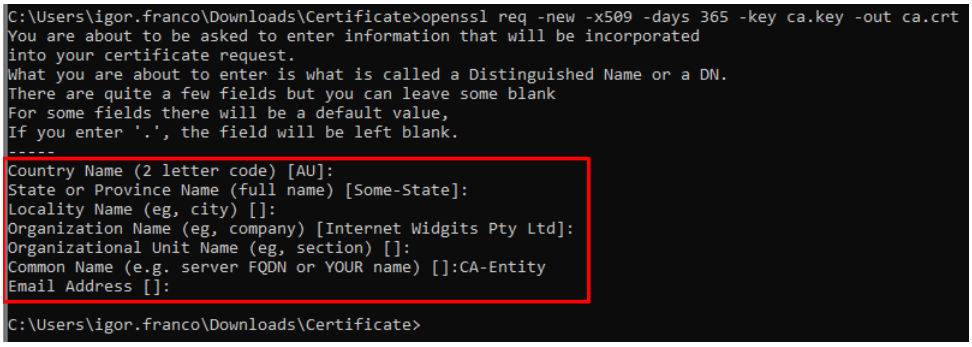



Figure 225: Certificate Generation using OpenSSL (step 3)

4- Now generate the device's private key, similar to step 2, using the following command:

```
openssl genrsa -out DeviceName.key 2048
```

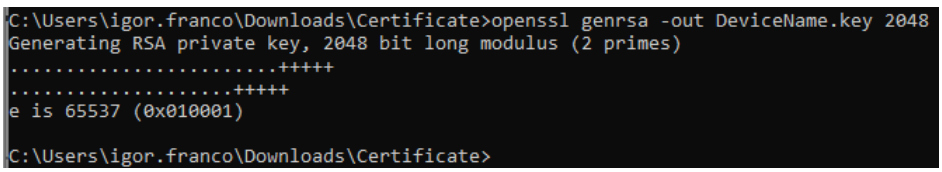



Figure 226: Certificate Generation using OpenSSL (step 4)

5- After that, generate the certificate request file based on the private key using the following command:

```
openssl req -new -key DeviceName.key -out DeviceName.csr
```

Enter the desired information, and remember to use a common name other than CA.

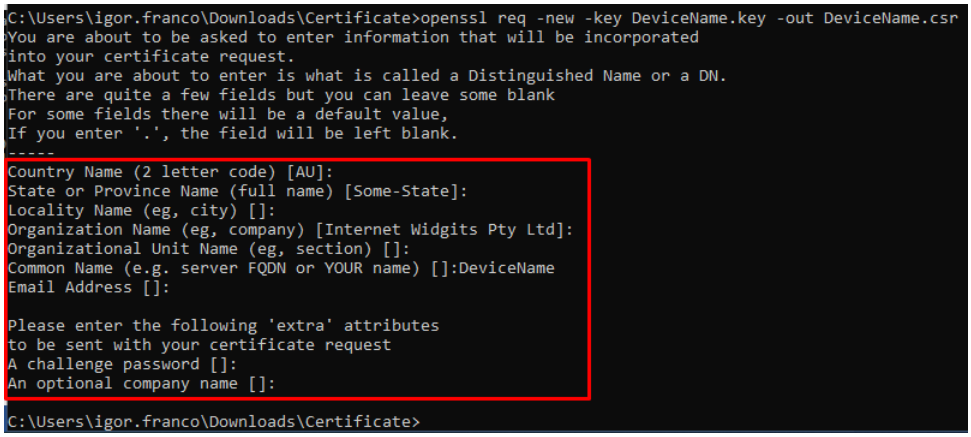



Figure 227: Certificate Generation using OpenSSL (step 5)

6- Finally, generate the device certificate using the CA private key, the CA certificate, and the device certificate request file using the following command:

```
openssl x509 -req -days 365 -in DeviceName.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out DeviceName.crt
```

Set the expiration date as desired with the parameter *-days* and the serial number of the certificate with the argument *-set_serial*.

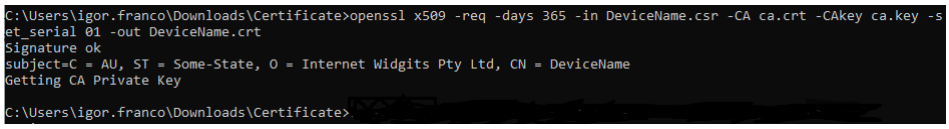



Figure 228: Certificate Generation using OpenSSL (step 6)

7- Repeat steps 4 to 6 for any new device.

8- (Optional) OpenSSL provides a tool to verify that the device certificate works with CA:

Use the following command:

```
openssl verify -purpose sslserver -CAfile ca.crt DeviceName.crt
```

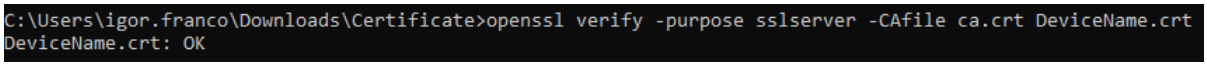



Figure 229: Certificate Generation using OpenSSL (step 8)

### 7.1.3. TA Key Generation by OpenVPN

The OpenVPN project provides a tool for generating a TLS key, commonly called ta.key. This key is an extra layer of protection on OpenVPN's UDP/TCP communication ports, so the use of this key can be interpreted as an HMAC Firewall for VPN communication, requiring the existence of the parameter on both sides of the communication for it to be established.

The key generation in Windows can be done with the following command:

```
openvpn --genkey secret ta.key
```

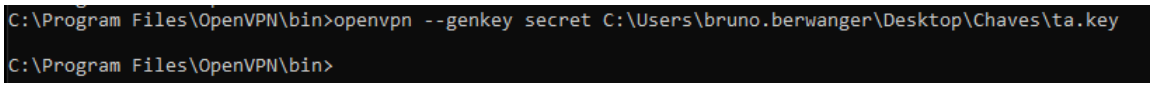



Figure 230: TA Key Generation in Windows example

To execute the command, we used the executable installed with the OpenVPN package. The directory used in the image above is an example and is optional. You can use only the desired file name too.

The command can be used to generate the key from Linux, but there is a minor change in the command compared to Windows. To generate the key on Linux, use the following command in the terminal:

```
openvpn --genkey --secret ta.key
```

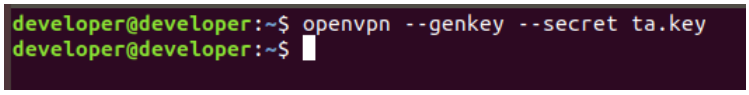



Figure 231: TA Key Generation in Linux example

This parameter is not mandatory for VPN communication, but if the server uses it, all its clients must also use it, and the key for the server and the clients must be the same.