User Manual
Nexto Xpress

MU216600 Rev. Z

March 4, 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 trainings 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, directly 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 form. 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

Nexto Xpress is a powerful compact Programmable Logic Controller (PLC) part of Nexto Series family of controllers and I/O modules. Nexto Xpress delivers high-speed processing power in a compact design with embedded I/O. There are several options to choose from, allowing the best solution for entry-level applications.

This product portfolio targets small control systems, offering models containing from a few digital inputs and outputs up to options with 43 I/O points concentrated in a single controller, including analog inputs and outputs with temperature support (RTD sensors). In case of additional I/O needs, the system can be easily expanded using expansion modules (see section Related Products). Additionally, the number of I/O points can be further expanded through remote (distributed) I/O devices communicating via protocols such as CANopen, EtherNet/IP, PROFINET and MODBUS.

Nexto Xpress is suitable for small applications and remote distributed I/O. It may be applied in verticals such as infrastructure, building automation, water, wastewater, food, textiles, factory automation, machines and several other OEM solutions, including motion control applications. The inclusion of an integrated firewall provides enhanced protection and security for the systems, safeguarding data integrity and mitigating potential cyber threats. Additionally, the controller is an ideal solution for complementing big applications along with Nexto Series portfolio, extending the range of applications using the same technology and engineering environment. This is a great advantage for OEMs and systems integrators with needs of small to large applications.

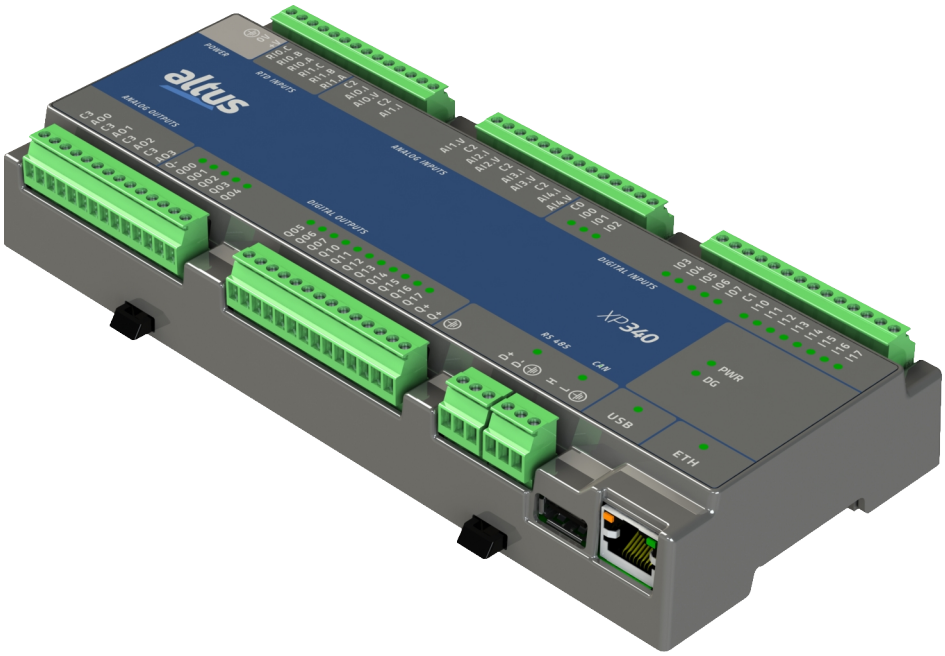


Figure 1: Nexto Xpress

1.1. Documents Related to this Manual

This manual will focus on information that is specific for the controllers of Nexto Xpress family. For other functionalities that are identical along all controllers of Nexto Series, this manual will just point to the corresponding manual of Nexto Series that contains the information. These related manuals are described on the following table, and are available in its last version on the site http://www.altus.com.br/site_en/.

Code	Description	Language
CE114000	Nexto Series – Technical Characteristics	English
CT114000	Série Nexto – Características Técnicas	Portuguese
MU216600	Nexto Xpress User Manual	English
MU216000	Manual de Utilização Nexto Xpress	Portuguese
MU214600	Nexto Series User Manual	English
MU214000	Manual de Utilização Série Nexto	Portuguese
MU299609	MasterTool IEC XE User Manual	English
MU299048	Manual de Utilização MasterTool IEC XE	Portuguese
MP399609	MasterTool IEC XE Programming Manual	English
MP399048	Manual de Programação MasterTool IEC XE	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
MU223603	IEC 60870-5-104 Server Device Profile Document	English
NAP151	Utilização do Tunneller OPC	Portuguese

Table 1: Documents Related

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 if all parts requested are in perfect shape. 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 guaranties 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

For Altus Technical Support contact in São Leopoldo, RS, call +55 51 3589-9500. For further information regarding the Altus Technical Support existent on other places, see https://www.altus.com.br/en/ or send an email to altus@altus.com.br.

If the equipment is already installed, you must have the following information at the moment of support requesting:

- The model from the used equipments and the installed system configuration
- The product serial number
- The equipment revision and the executive software version, written on the tag fixed on the product's side
- CPU operation mode information, acquired through MasterTool IEC XE
- The application software content, acquired through MasterTool IEC XE
- Used programmer version

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 Xpress controllers.

2.1. Panels and Connections

The following figure shows the XP325 front panel:

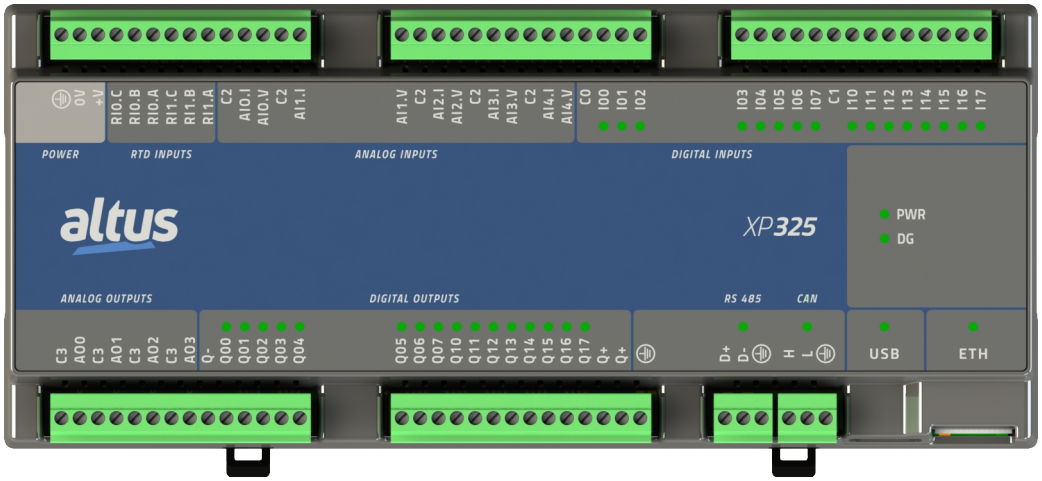


Figure 2: XP325 front panel

The front panel contains the identification of the I/O and communication interfaces available on Nexto Xpress controllers. The digital 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.

Additionally, on the right side of front panel there are 2 LEDs used to indicate power and diagnostics. The following table shows the LEDs description. For further information regarding the LEDs status and meaning, see Maintenance chapter.

LED	Description
PWR	Status of internal power supply
DG	Diagnostic indication
Ixx.x	Status of digital inputs
Qxx.x	Status of digital outputs
D+/-	Status of RS-485 interface (blinks on activity)
H/L	Status of CAN interface (blinks on activity)
USB	Status of USB port (turns on when device is mounted)
ETH	Status of Ethernet interface (turns on when connected, blinks on activity)

Table 2: LEDs Description

2.2. Product Features

2.2.1. General Features

	XP300	XP315	XP325	XP340	XP350	XP351
Digital Inputs	12					
Fast Inputs	4					
Digital Outputs	12					
Fast Outputs	4					
Max. number of high-speed counters	1					
Max. number of external interruptions	2					
Max. number of PTO outputs	2					
Max number of VFO/PWM outputs	4					
V/I analog inputs (AI)	-	5 to 10 See Notes	5 to 10 See Notes	5 to 10 See Notes	5 to 10 See Notes	5 to 10 See Notes
RTD analog inputs (AI)	-	2	2	2	2	2
V/I analog outputs (AO)	-	-	4	4	-	-
Ethernet TCP / IP interface	1					
RS-485 Serial interface	1					
CAN Interface	1					
USB Host port	1					
User web pages (Webvisu)	No	No	No	Yes	No	No
Motion control (Softmotion)	No	No	No	No	Yes, without CNC	Yes, with CNC
Remote I/O Mode **CANopen Slave and EtherNet/IP Adapter**	Yes	Yes	Yes	No	No	No
FTP	Yes					
Firewall	Yes					
VPN	Yes					
Maximum number of tasks	16					
Programming languages	Structured Text (ST) Ladder Diagram (LD) Sequential Function Chart (SFC) Function Block Diagram (FBD) Continuous Function Chart (CFC)					
Online changes	Yes					
Watchdog	Yes					
Real-time clock (RTC)	Yes Resolution of 1 ms, max. variance of 3 seconds per day, retention time of 14 days.					
Status and diagnostic indication	LEDs, web pages and CPU's internal memory					
Isolation						
Protective earth ⏚ to all	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					
Logic/RS-485/CAN/USB to all	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					
Ethernet to all	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					
Power Supply/Analog I/O to all	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					
Digital Inputs to all	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					
Digital Inputs Group I0x to I1x	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					

	XP300	XP315	XP325	XP340	XP350	XP351
Digital Outputs to all	1,500 Vdc / 1 minute (1,000 Vac / 1 minute)					
Maximum power dissipation	5 W					
Maximum wire size	0.5 mm^2 (20 AWG) with ferrule 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					
Operating temperature (UL/cUL)	0 to 60 °C					
Storage temperature	-25 to 75 °C					
Operating and storage relative humidity	5% to 96%, non-condensing					
Vibration resistance (IEC 60068-2-6, sinus)	7 mm from 5 to 8.4 Hz 2 G from 8.4 to 500 Hz 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)	215.5 x 98.8 x 34.0 mm					
Package dimensions (W x H x D)	270.0 x 102.0 x 40.0 mm					
Weight	370 g					
Weight with package	430 g					

Table 3: General Features

Notes:

V/I analog inputs (AI): By default, each analog input is composed by 2 terminals (AIx.V and AIx.I), and when selecting one mode (V, for example), the other pin (I, for example) becomes unused. With the function *AnalogInputProbe*, provided by the *LibIntegratedIoExt* library, it is possible to use these free inputs, allowing to have up to 10 analog inputs (5 on terminals AIx.V and other 5 on terminals AIx.I), with the same technical characteristics informed on this document. For additional information, please consult the Technical Support.

Motion control: PLCopen Motion Control Part 1 function block support for single-axis control, multi-axis synchronization, electronic gearing (CAME), special editor for motion planning (CAM), and others.

Maximum number of tasks: This value represents the maximum total of user and system tasks. The detailed description of possible user tasks can be found on Project Profiles section of User Manual. Before MasterTool IEC XE v3.30, this value was defined as “5”.

Isolation: The *Logic* term refers to the internal interfaces such as processors, memories and USB, serial and CAN communication interfaces.

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

Operating temperature: The minimum operating temperature is 0°C for units with product revision inferior to AS/AS/AW/AE for XP300/XP315/XP325/XP340 respectively.

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 61131-3: Programmable controllers - Part 3: Programming languages
	DNV Type Approval – DNV-CG-0339 (TAA000034G) **(Except the XP350 and XP351)**
	2014/30/EU (EMC) 2014/35/EU (LVD) 2011/65/EU and 2015/863/EU (ROHS)
	S.I. 2016 No. 1091 (EMC) S.I. 2016 No. 1101 (Safety) S.I. 2012 No. 3032 (ROHS)
	Ordinary Locations: cULus LISTED, E473496 Hazardous Locations: cULus LISTED, E536282 Class I Division II, Groups A, B, C, D
	TR 004/2011 (LVD) CU TR 020/2011 (EMC)

Table 4: Standards and Certifications

2.2.3. Memory

	XP300	XP315	XP325	XP340	XP350	XP351
Addressable input variables memory (%I)	2 KB					
Addressable output variables memory (%Q)	2 KB					
Direct representation variable memory (%M)	1 KB					
Symbolic variable memory	2 MB	2 MB	2 MB	6 MB	6 MB	6 MB
Program memory	3 MB	3 MB	3 MB	8 MB	8 MB	8 MB
Total memory **Program memory (max. defined per model) +** **Source code memory (backup) +** **Webvisu files memory**	64 Mbytes					
Retain/persistent memory **(user configurable)**	7,5 KB (Expandable up to 64 KB using Recipes stored on User Files memory (see article on knowledge base))					
User files memory (backup)	8 MB					

Table 5: Memory

Note:

Program memory: From version 3.40 of MasterTool IEC XE, the memory has been increased from 2MB to 3MB in the XP300, XP315, and XP325 models, and from 6MB to 8MB in the XP340 model.

Persistent and Retain symbolic variables memory: Area where the retentive/persistent symbolic variables are allocated. The controller performs retentive/persistent cyclic saves every 5 seconds.

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 the *VAR PERSISTENT* expression in the declaration of field variables of POUs.

Retentive/persistent symbolic memory variables full behaviour can be found in the following table where "X" means that memory data is safe under the presented scenario while "-" means data loss.

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
Reset Process (IEC 60870-5-104)	-	X	X

Table 6: Post-command Variable Behavior

2.2.4. Protocols

		Interface
Open Protocol	✓	COM 1 / USB
MODBUS RTU Master	✓	COM 1
MODBUS RTU Slave	✓	COM 1
MODBUS TCP Client	✓	NET 1
MODBUS TCP Server	✓	NET 1
MODBUS RTU over TCP Client	✓	NET 1
MODBUS RTU over TCP Server	✓	NET 1
CANopen Master	✓	CAN
CANopen Slave	✓ (except XP340, XP350 and XP351)	CAN
CAN low level	✓	CAN
SAE J-1939	✓	CAN
OPC DA Server	✓	NET 1 / USB
OPC UA Server	✓	NET 1 / USB
EtherCAT Master	✓	NET 1
SNMP Agent	✓	NET 1 / USB
IEC 60870-5-104 Server	✓ (only XP340)	NET 1
EtherNet/IP Scanner	✓	NET 1
EtherNet/IP Adapter	✓	NET 1
MQTT Client	✓	NET 1 / USB
SNTP Client (for clock synchronism)	✓	NET 1 / USB
PROFINET Controller	✓	NET 1
PROFINET Device	✗	-
OpenVPN Client	✓	NET 1 / USB
OpenVPN Server	✓	NET 1 / USB
FTP Server	✓	NET 1 / USB

Table 7: Protocols

Notes:

USB: Need to use Serial Converter, WiFi, Modem or Ethernet Adapter.

PROFINET Controller: Enabled for use on a simple (not ring) network with up to 8 devices. For larger applications, consult technical support.

2.2.5. RS-485

	RS-485
Connector	3-pin terminal block
Physical interface	RS-485
Communication direction	RS-485: half duplex
RS-485 max. transceivers	32
Termination	Yes (Configurable)
Baud rate	2400, 4800, 9600, 19200, 38400, 57600, 115200 bps

Table 8: RS-485 Serial Interface Features

2.2.6. CAN

	CAN
Connector	3-pin terminal block
Physical interface	CAN bus
Supported standards	CAN 2.0A 2.0B (11-bit and 29-bit identifiers)
Max. number of nodes	64
Termination	Yes (Configurable)
Baud rate	10, 20, 50, 100, 125, 250, 500, 800, 1000 kbit/s

Table 9: CAN Interface Features

2.2.7. USB

	USB
Connector	USB 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 USB RS-232 Serial Converter USB 3G/4G Modem USB WiFi Adapter USB Ethernet Adapter

Table 10: USB Interface Features

Notes:
USB RS-232 Serial Converter: See the list of supported devices on respective section USB to RS-232 Converters.
USB 3G/4G Modem: See the list of supported devices on respective section Modem Devices.
USB WiFi Adapter: See the list of supported devices on respective section WiFi Adapters.
USB Ethernet Adapter: See the list of supported devices on respective section Ethernet Adapters Configuration.

ATTENTION:

The CPU supports the use of only one USB device at a time. Devices such as USB HUBs, for example, are not supported.

2.2.8. Ethernet

	Ethernet
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) UDP (User Datagram Protocol)
Diagnostics	LED (Link/Activity)

Table 11: Ethernet Interface Features

2.2.9. Power Supply

	Power Supply
Nominal input voltage	24 Vdc
Input voltage	19.2 to 30 Vdc
Maximum input current (in-rush)	50 A / 300 us
Maximum input current	300 mA

Table 12: Power Supply Features

2.2.10. Digital Inputs

	Digital Inputs
Input type	Optoisolated sink type 1 Two isolated groups of 8 inputs each
Input voltage	24 Vdc 15 to 30 Vdc for logic level 1 0 to 5 Vdc for logic level 0
Input impedance	4.95 kΩ
Maximum input current	6.2 mA @ 30 Vdc
Input state indication	Yes
Response time	0.1 ms
Input filter	Disabled or 2 ms to 255 ms – by software

Table 13: Digital Inputs Features

Note:

Input filter: The filter sampling is performed on MainTask (or Refresh function), then it's recommended to use multiple values of the task interval.

2.2.11. Fast Inputs

	Fast Inputs
Number of fast inputs	4 (can be used as high-speed counter, External interrupt or standard digital input)
Max. number of high-speed counters	1
Max. number of external interrupts	2
Connector configuration	I00, I01, I02 and I03
Input voltage	24 Vdc 15 to 30 Vdc for logic level 1 0 to 5 Vdc for logic level 0
Input impedance	1.85 kΩ
Input maximum current	16.2 mA @ 30 Vdc
Configuration mode	**1-input modes** Standard digital input External interrupt **2-input modes** Up/Down (A count, B direction) with zero (uses I00, I01, I02) Quadrature 2x (uses I00, I01) Quadrature 2x with zero (uses I00, I01, I02) Quadrature 4x (uses I00, I01) Quadrature 4x with zero (uses I00, I01, I02)
Counting direction control	Hardware only
Counting input detection edge	Rising edge, active at logic level 1 (except for quadrature 4x, where it counts on both edges)
Data format	Signed 32-bit integer
Operation limit	From - 2,147,483,648 to 2,147,483,647
Maximum input frequency	100 kHz
Minimum pulse width @ 24 Vdc	2 μs

Table 14: Fast Inputs Features

2.2.12. Digital Outputs

	Digital Outputs
Output type	Optoisolated transistor source type
Maximum output current	1.5 A per output 12 A total
Leakage current	35 μA
On state resistance	105 mΩ
External power supply	19.2 to 30 Vdc
Switching time	20 μs - off-to-on transition @ 24 Vdc 500 μs - on-to-off transition @ 24 Vdc
Maximum switching frequency	250 Hz
Configurable parameters	Yes
Output state indication	Yes
Output protections	Yes, protection against surge voltages

Table 15: Digital Outputs Features

Note:
Switching time: The required time to turn off one specific output depends on the load.

2.2.13. Fast Outputs

	Fast Outputs	
Number of outputs	4 (can be used as VFO/PWM, PTO or standard digital output)	
Max. number of PTO outputs	2	
Max number of VFO/PWM outputs	4 when using no PTO 2 when using 1 PTO 0 when using 2 PTO	
Connector configuration	Q14, Q15, Q16 and Q17	
Maximum current	0 to 500 Hz: 1.5A per output / 6.0A total 500 to 200 KHz: 0.5A per output / 2.0A total	
Output type	Transistor source	
Pulse generation maximum frequency	200 kHz @ 60 mA	
Minimum pulse width @ 24 Vdc	MINIMUM LOAD	MINIMUM PULSE TIME
	400 Ω	320 ns
State indication	Through static reserved operands	
Protections	TVS diode at all transistor outputs	
Operation voltage	19.2 to 30 Vdc	
Output impedance	700 mΩ	
Output modes	Standard digital output VFO/PWM PTO (Q14 and Q16 only. Adjacent output is forced to standard digital output)	

	Fast Outputs	
	PTO	VFO/PWM
Functions executed by software	Writing of number of pulses to be generated Writing of acceleration and deceleration number of pulses Start/end outputs operation Fast outputs diagnostics Fast outputs current state monitoring	Writing of the frequency value to be generated (1 Hz to 200 kHz). Writing of outputs duty cycle (1% to 100%) Start/end of outputs operations Fast outputs diagnostics.

Table 16: Fast Outputs Features

2.2.14. Analog Inputs

	Analog Inputs
Input type	Voltage or current input, single ended, individually configured
Data format	16 bits in two's complement, justified to the left
Converter resolution	12 bits monotonicity guaranteed, no missing codes
Conversion time	400 μs (all V/I and RTD channels enabled)
Input state indication	Yes
Module protections	Yes, protection against surge voltages and polarity inversion

Table 17: Analog Inputs Features

	Voltage Input Mode		
Input ranges	Range	Engineering Scale	Resolution
	0 to 10 Vdc	0 to 30,000	2.5 mV
Precision	$\pm$0.3 % of full scale @ 25 °C $\pm$ 0.010 % of full scale / °C		
Over scale	3 % of full scale		
Maximum input voltage	12 Vdc		
Input impedance	21 kΩ		
Configurable parameters	Signal type per input Filters		
Low pass filter time constant	100 ms, 1 s, 10 s or disabled		

Table 18: Voltage Input Mode Features

	Current Input Mode		
Input ranges	Range	Engineering Scale	Resolution
	0 to 20 mA	0 to 30,000	5.12 μA
	4 to 20 mA	0 to 30,000	5.12 μA
Precision	$\pm$0.3 % of full scale @ 25 °C $\pm$ 0.015 % of full scale / °C		
Over scale	3 % of full scale		
Maximum input current	30 mA		
Input impedance	119 Ω		
Configurable parameters	Signal type per input Filters Open Loop Value		
Low pass filter time constant	100 ms, 1 s, 10 s or disabled		

Table 19: Current Input Mode Features

Note:

Input ranges: When configured as 4 to 20 mA, input signals lower than 4 mA will result in negative values (-7,500 for 0 mA). Starting from MasterTool IEC XE version 3.16, a new parameter called *Open Loop Value* was included to select the behavior in this situation. The default value is *Disabled* (which provides a linear reading as described above), having also the option to provide a fixed reading equal to lower and upper limits ("*0*" or "*30000*").

	RTD Input
Precision	$\pm$0.5 % of full scale @ 25 °C
Supported scales	Pt100, Pt1000, 0 to 400 Ω, 0 to 4000 Ω
Excitation current	1 mA
Resistance range (scale)	0 to 400 Ω (used for PT100) 0 to 4000 Ω (used for PT1000)
Over Scale	5 % of full scale
Configurable parameters	Signal type per input Filters
Low pass filter time constant	100 ms, 1 s, 10 s or disabled
Maximum sensor cable impedance (per wire)	5 Ω

Table 20: RTD Input Features

2.2.15. Analog Outputs

	Analog Outputs
Output type	Voltage or current output, individually configured
Data format	16 bits in two's complement, justified to the left
Converter resolution	12 bits monotonicity guaranteed, no missing codes
Update time	450 μs (all outputs enabled)
Output state indication	Yes
Module protections	Yes, protection against surge voltages and polarity inversion

Table 21: Analog Outputs Features

	Voltage Output Mode		
Output ranges	Range	Engineering Scale	Resolution
	0 to 10 V	0 to 30,000	2.5 mV
Precision	±0.3 % of full scale @ 25 °C ± 0.025 % of full scale / °C		
Stabilization time	4 ms		
Maximum output value	+ 10.3 Vdc		
Load impedance	> 1 kΩ		
Configurable parameters	Signal type per output		

Table 22: Voltage Output Mode Features

	Current Output Mode		
Output ranges	Range	Engineering Scale	Resolution
	0 to 20 mA	0 to 30,000	5.18 μA
	4 to 20 mA	0 to 30,000	5.18 μA
Precision	±0.3 % of full scale @ 25 °C ± 0.020 % of full scale / °C		
Stabilization time	4 ms		
Maximum output value	+ 20.6 mA		
Load impedance	< 600 Ω		
Configurable parameters	Signal type per output		

Table 23: Current Output Mode Features

Note:

Output ranges: When configured as 4 to 20 mA, the output can be set to values lower than 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 Xpress controllers, it is necessary to check the version of MasterTool IEC XE. The following table shows the minimum version required (where the controllers were introduced) and the respective firmware version at that time:

Controller model	MasterTool IEC XE	Firmware version
XP300, XP315 and XP325	3.10 or above	1.7.0.0 or above
XP340	3.18 or above	1.8.0.0 or above
XP350	3.50 or above	1.12.5.0 or above
XP351	3.52 or above	1.12.29.0 or above

Table 24: Compatibility with other products

Additionally, along the development roadmap of MasterTool IEC XE some features may be included (like special FunctionBlocks, etc.), which can introduce a requirement of minimum firmware version. During the download of the application, MasterTool IEC XE checks the firmware version installed on the controller and, if it does not meets 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 Xpress 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.

Additionally, the integrated I/O can be updated asynchronously at any point of user application code using the refresh functions available on section Inputs and Outputs Update.

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

The time required for reading and writing the integrated I/O is dependent of the number and the type of the I/O channels enabled. For digital I/O, all channels are always enabled and the time added to MainTask is not relevant. For analog I/O, the time added to MainTask is determined by the conversion time (for analog inputs) and by the update time (for analog outputs), both described on General Features table.

2.4.3. Time for Instructions Execution

The below table presents the necessary execution time for different instructions in Nexto Xpress CPUs.

Instruction	Language	Variables	Instruction Times (μs)
1000 Contacts	LD	BOOL	10
1000 Divisions	ST	INT	180
		REAL	40
	LD	INT	180
		REAL	40
1000 Multiplications	ST	INT	13
		REAL	13
	LD	INT	13
		REAL	13
1000 Sums	ST	INT	13
		REAL	13
	LD	INT	13
		REAL	13

Table 25: Instruction Times

2.4.4. Initialization Times

The initialization time of Nexto Xpress controllers is approximately 40 s.

2.5. Physical Dimensions

Dimensions in mm.

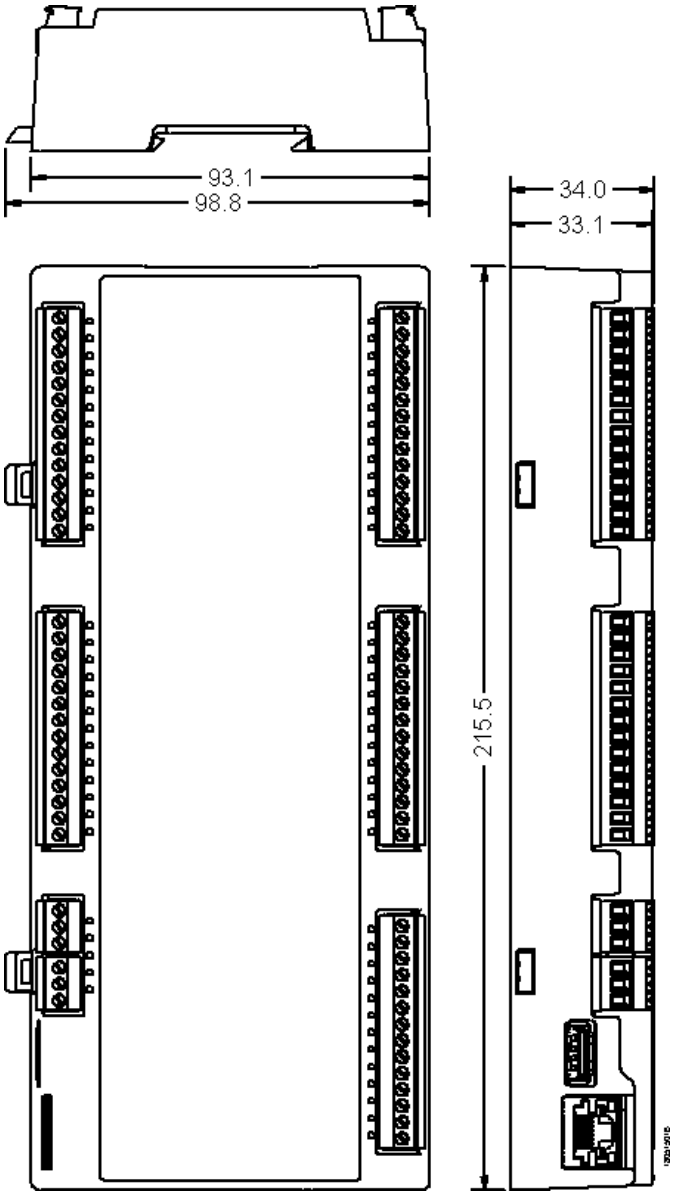


Figure 3: XP3xx Physical Dimensions

2.6. Purchase Data

2.6.1. Integrant Items

The product package has the following items:

- Compact PLC module
- Connectors

2.6.2. Product Code

The following code should be used to purchase the product:

Code	Description
XP300	High-Speed Compact PLC with 16 DI, 16 DO Transistor, 1 Ethernet, 1 RS-485 Serial and CANopen Master
XP315	High-Speed Compact PLC with 16 DI, 16 DO Transistor, 5 V/I AI, 2 RTD AI (3 wire), 1 Ethernet, 1 RS-485 Serial and CANopen Master
XP325	High-Speed Compact PLC with 16 DI, 16 DO Transistor, 5 V/I AI, 2 RTD AI (3 wire), 4 AO, 1 Ethernet, 1 RS-485 Serial and CANopen Master
XP340	High-Speed Compact PLC with 16 DI, 16 DO Transistor, 5 V/I AI, 2 RTD AI (3 wire), 4 AO, 1 Ethernet, 1 RS-485 Serial, CANopen Master and user web pages support
XP350	High-Speed Compact PLC with Standard Softmotion, 16 DI, 16 DO transistor, 5 V/I AI, 2 RTD AI (3 wire), 1 Ethernet port, 1 RS-485 serial and CANopen Master
XP351	High-Speed Compact PLC with Advanced Softmotion (CNC), 16 DI, 16 DO transistor, 5 V/I AI, 2 RTD AI (3 wire), 1 Ethernet port, 1 RS-485 serial and CANopen Master

Table 26: Nexto Xpress Controller Models

2.7. Related Products

The following products must be purchased separately when necessary:

Code	Description
MT8500	MasterTool IEC XE
NX9202	RJ45-RJ45 2 m Cable
NX9205	RJ45-RJ45 5 m Cable
NX9210	RJ45-RJ45 10 m Cable
AL-2600	RS-485 network branch and terminator
AL-2306	RS-485 cable for MODBUS or CAN network
AL-1766	CFDB9-Terminal Block Cable
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
XP101	Nexto Xpress Expansion, 16 DI 24 Vdc
XP106	Nexto Xpress Expansion, 8 DI 24 Vdc and 6 DO Relay
XP201	Nexto Xpress Expansion, 16 DO Transistor
TLE3-21100	Gateway IoT Industrial

Table 27: Related Products

Notes:

MT8500: MasterTool IEC XE is available in four different versions: LITE, BASIC, PROFESSIONAL and ADVANCED. For more details, please check MasterTool IEC XE User Manual - MU299609.

NX92xx: Cable for programming the CPUs of the Nexto Series and Ethernet point-to-point with another device with Ethernet interface communication.

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 that are at the ends of network must be configured with termination, except when there is a device with active internal termination, the rest must be configured without termination.

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

AL-1766: Cable with a female DB9 connector and terminals for communication between HMI P2 and Nexto Xpress/NX3003 controllers.

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

AMJG0808: Cable for programming the CPUs.

XP101 / XP106 / XP201: CANopen expansion modules.

3. Installation

This chapter presents the necessary proceedings for the physical installation of Nexto Xpress controllers, as well as the care that should be taken with other installation within the panel where the controller is been 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.
- The 24 Vdc power supply port shall be equipped with a filter TDK-Lambda model RSMN-2003 or equivalent.
- The cables of all ports (power, I/O and communication) shall be equipped with a pair of low/high frequency ferrites Wurth Electronics 74272221/74271221 or equivalent.

ATTENTION

Additionally to the standard instructions described on this chapter, the following installation requirements shall be met:

- These devices are open-type devices that are to be installed in an enclosure suitable for the environment and accessible only with use of a tool or key.
- This equipment is suitable for use in Class I, Division 2, Groups A, B, C and D or non-hazardous locations only.

DANGER

EXPLOSION HAZARD - Do not disconnect equipment unless power has been removed or the area is known to be non-hazardous.

3.1. Mechanical Installation

Nexto Xpress 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.

CAUTION

For achieving the temperature specification of the controller, the installation must provide a free space around the device as described on section Panel Design of Nexto Series User Manual code MU214600.

3.1.1. Installing the controller

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

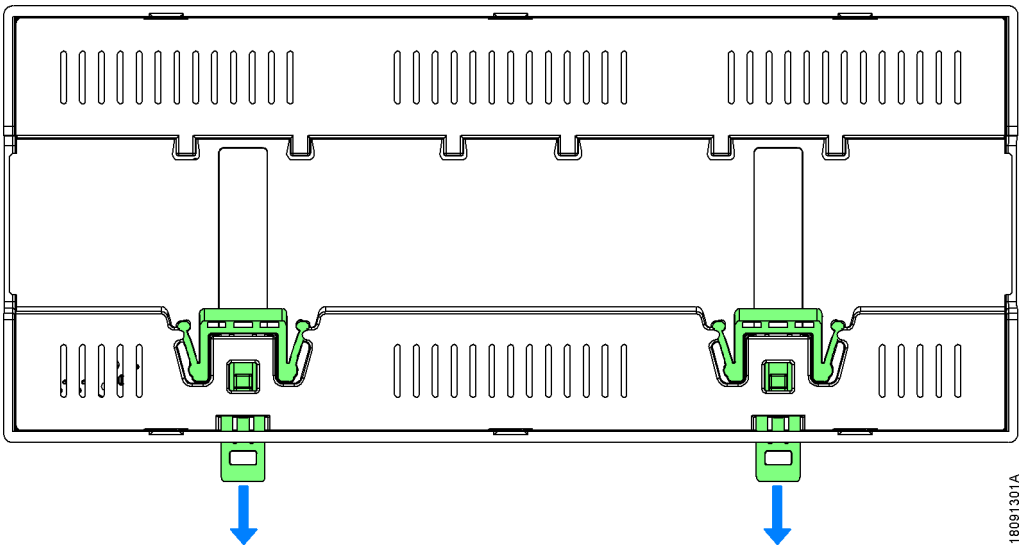


Figure 4: Moving the two locks to open position

Next, place the controller on the DIN rail fitting the top side first and then the bottom side, as indicated on steps 1 and 2 of the figure below:

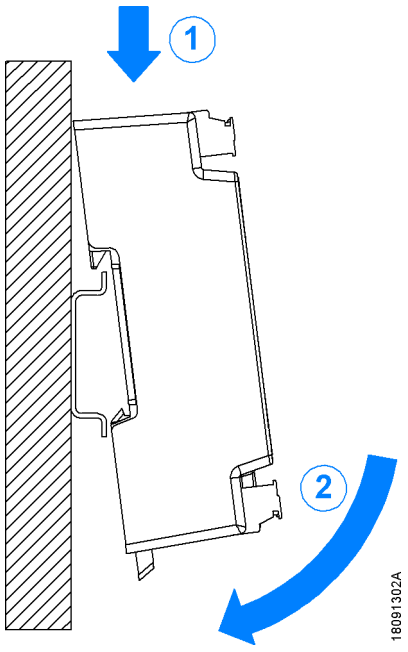


Figure 5: Fixing the controller on the DIN rail

Finally, move the two locks to closed position to lock the controller on the DIN rail, as shown on the figure below:

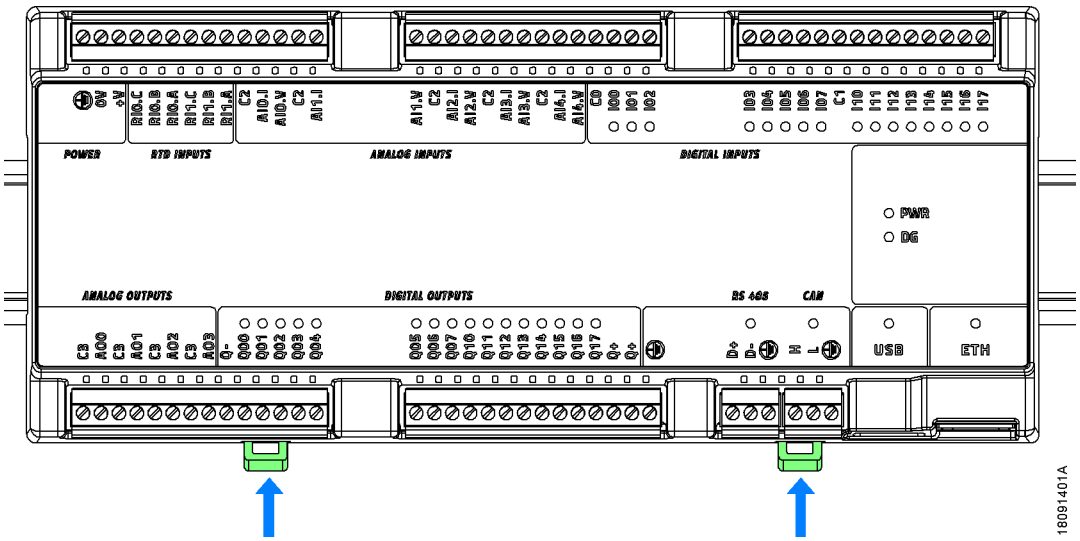


Figure 6: Locking the controller on the DIN rail

3.1.2. Removing the controller

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

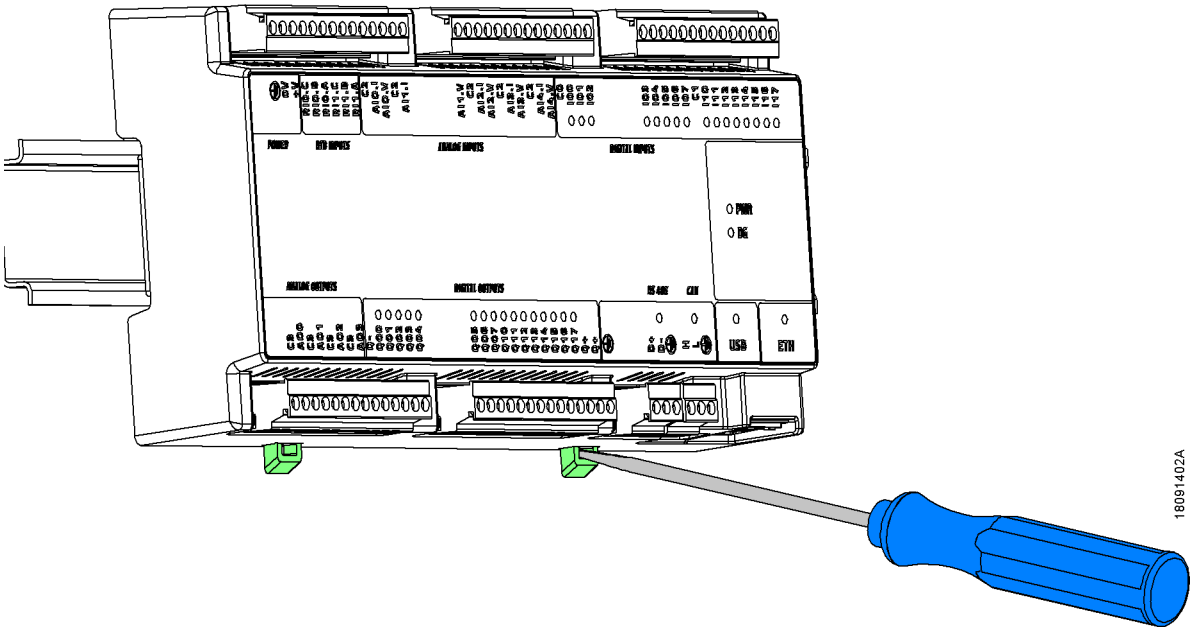


Figure 7: Unlocking the controller from the DIN rail

3.2. Electrical Installation

DANGER

When executing any installation in an electric panel, certify that the main energy supply is OFF.

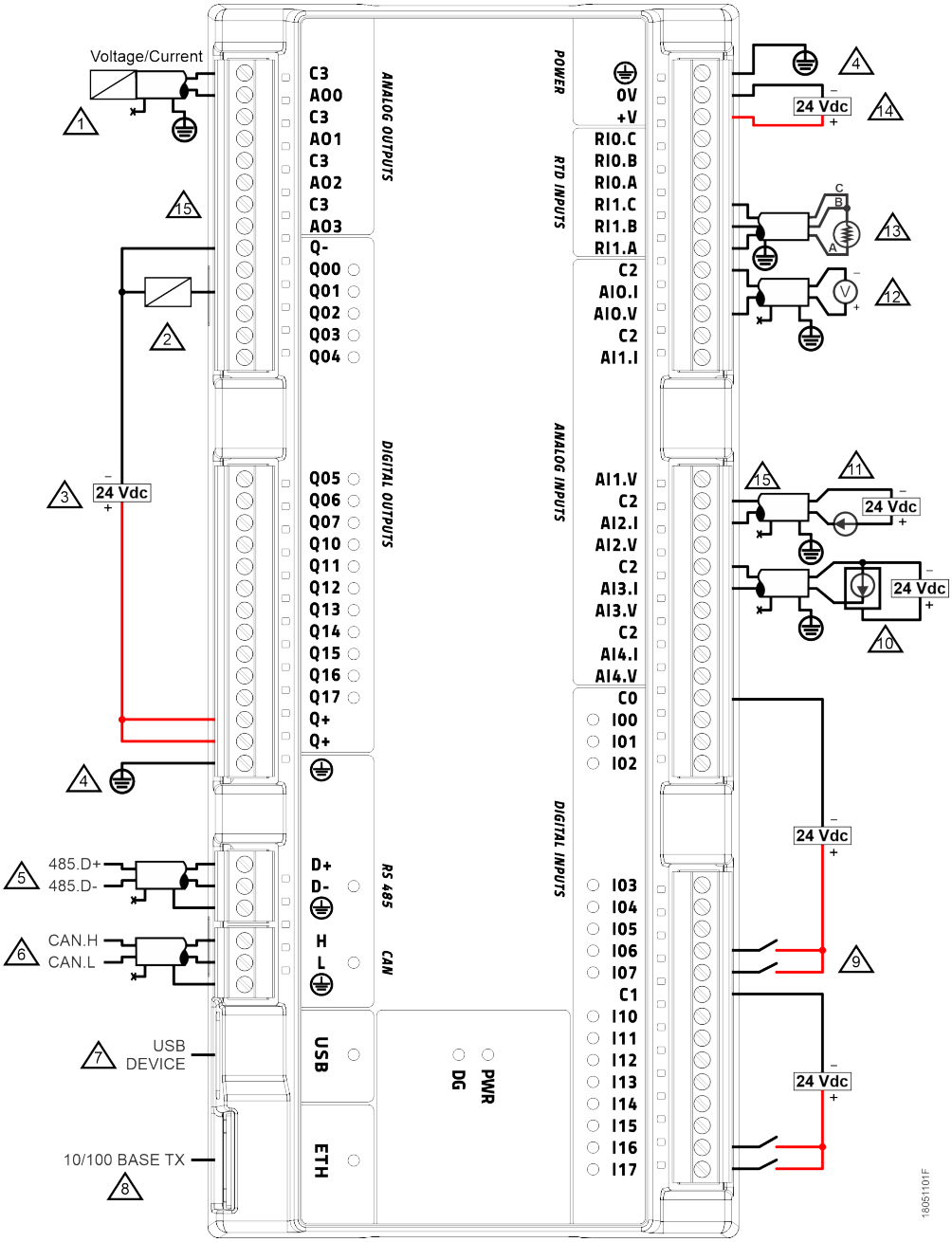


Figure 8: XP3xx Electrical Installation Diagram

Diagram Notes:

1. Typical connection of analog output on voltage/current mode.

2. Typical connection of digital output (source type).

3. External power supply to supply outputs Q00 to Q17, terminals Q + must be connected to +24 Vdc, and terminal Q- must be connected to 0 Vdc.

4. Protective Earth terminals for power supply and communication ports. Both shall be externally connected to ground.

5. Typical connection of RS-485 serial interface.

6. Typical connection of CAN interface.

7. Please check the technical characteristics table of USB port for the list of supported devices.

8. Use Ethernet cables informed on Related Products section.

9. Typical connection of digital input (sink type). C0 and C1 are the common points for the isolated groups I0x and I1x respectively.

10. Typical connection of current analog input (field device with power supplied separately from analog signal).

11. Typical connection of current analog input (field device with power supplied with the analog signal, 2-wire).

12. Typical connection of voltage analog input.

13. Typical connection of RTD analog input (3-wire).

14. External power supply connection.

15. The signals from the analog inputs and outputs are not isolated from the main power supply, so the C2 and C3 signals cannot have a potential difference from the 0V of the main power supply. It's recommended to connect the 0V of the main power supply to the analog references C2 and C3 before connecting to the Xpress.

Protective conductor terminal.

3.3. Ethernet Network Connection

The ETH communication interface, identified as NET 1 on MasterTool IEC XE, 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 Xpress 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.

3.3.1. IP Address

The Ethernet interface comes with the following default parameters configuration:

	NET 1
IP Address	192.168.15.1
Subnet Mask	255.255.255.0
Gateway Address	192.168.15.253

Table 28: 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 IEC XE, where the network parameters can be modified. For further information regarding configuration and parameters modifications, see Ethernet Interface Configuration chapter.

3.3.2. Gratuitous ARP

The NET1 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 the MasterTool IEC XE software and in the controller startup when the application goes into Run mode.

Five ARP commands are triggered with 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.

3.3.3. Network Cable Installation

Nexto Xpress Ethernet port have standard pinout which are the same used in PCs. The connector type, cable type, physical level, among other details, are defined in the General Features table. Below is the description of the RJ-45 female connector, with the identification and description of the valid pinout for 10Base-T and 100Base-TX physical levels.

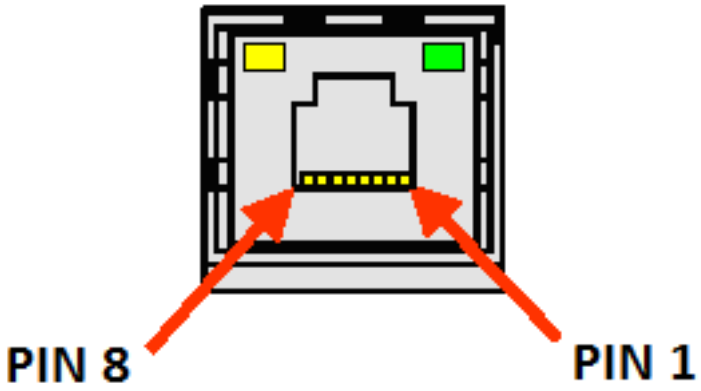


Figure 9: RJ45 Female Connector

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 29: RJ45 Female Connector Pin out

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 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.

3.4. Serial RS-485 and CAN Network Connection

As illustrated on Electrical Installation diagram, both RS-485 and CAN interface uses 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 internal termination shall be enabled (see CPU Configuration and Serial Interface Configuration configuration sections for CAN and RS-485 respectively).

4. Initial Programming

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

Just like for the other devices of Nexto Series, the programming of Nexto Xpress controllers is made through the MasterTool IEC XE (IDE) development interface, which offers a full IEC 61131-3 programming system with all languages defined by this standard (ST, LD, SFC, FBD, etc...) plus an additional one, the CFC. These languages can be used simultaneously on the same project, allowing the user to use the best features of each language, resulting in more efficient applications development, for easy documentation and future maintenance.

For further information regarding programming, see User Manual MasterTool IEC XE - MU299609, Programming Manual MasterTool IEC XE - MU399609 or IEC 61131-3 standard.

4.1. Memory Organization and Access

Different from other devices of Nexto Series (which are based on big-endian CPU), the Nexto Xpress controllers are based on a 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 30, 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 30: Memory Organization and Access Example

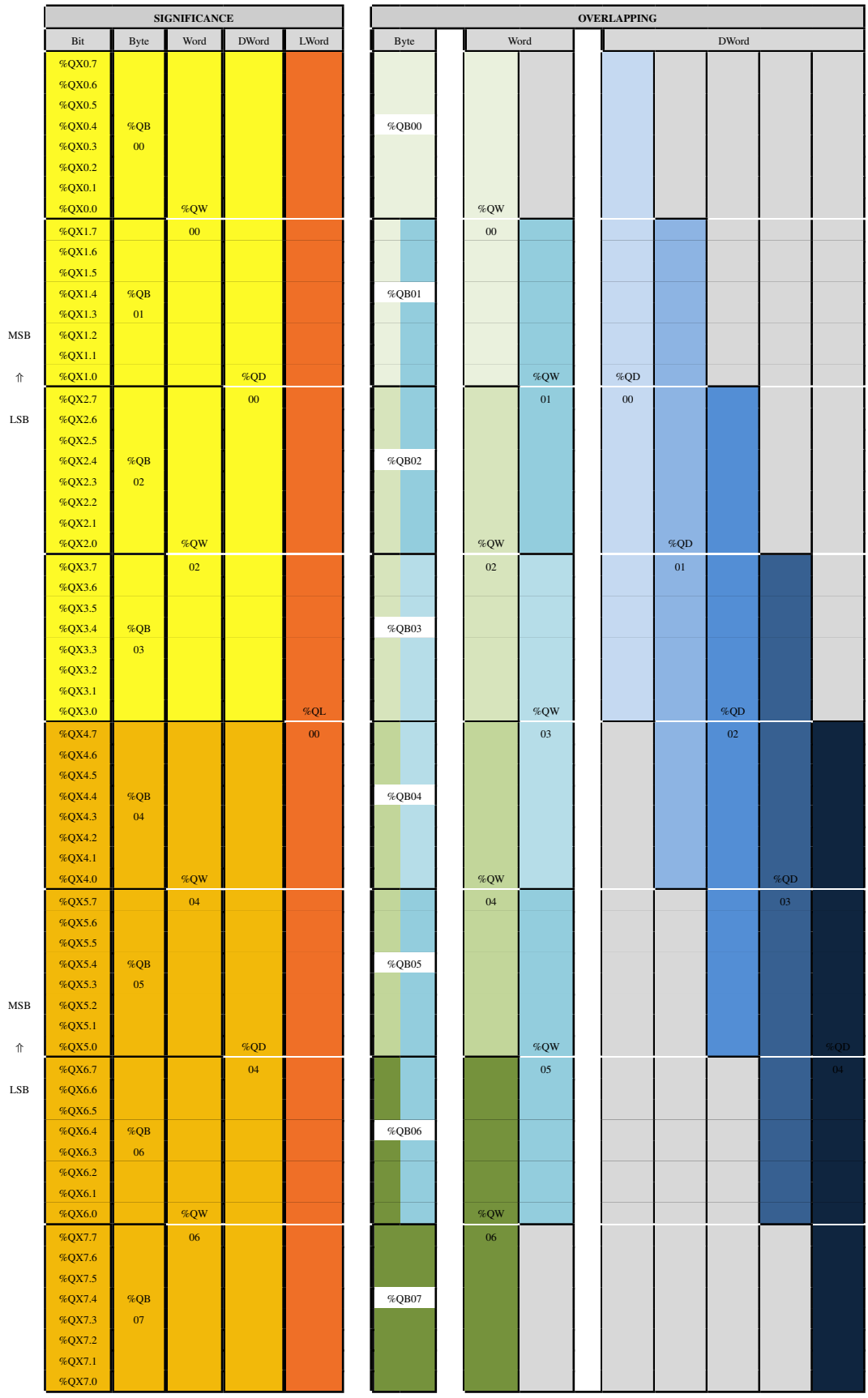


Table 31: Memory Organization and Access

4.2. Project Profiles

A project profile in the MasterTool IEC XE 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 Xpress controllers, there is only one project profile available: Machine Profile.

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

The following sections details the characteristics of each profile. It is important to note that the programming tool allows the profile change from an existent project (see project update section in the MasterTool IEC XE User Manual – MU299609), 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 IEC XE User Manual – MU299609.

4.2.1. Machine Profile

In the Machine 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, called TimeInterruptTask00, 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	-
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	4 ms	-

Table 32: Machine Profile Tasks

Also, this profile supports the inclusion of additional tasks associated to counter and external interruptions, resulting in a maximum of 5 tasks for user application.

ATTENTION

The suggested POU names associated with the tasks are not consisted. They can be changed, as long as they are also changed in the tasks configurations.

4.3. CPU Configuration

The controller's CPU configuration is located in the device tree, as shown on the figure below, and can be accessed by a double-click on the corresponding object. In this tab it's possible to configure watchdog behavior, clock synchronism, among other parameters, as described on section CPU Configuration.

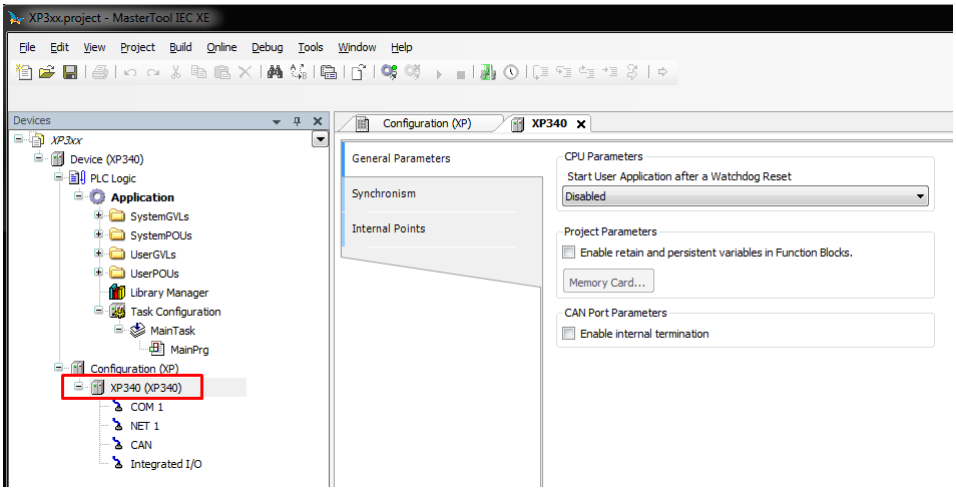


Figure 10: CPU Configuration

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

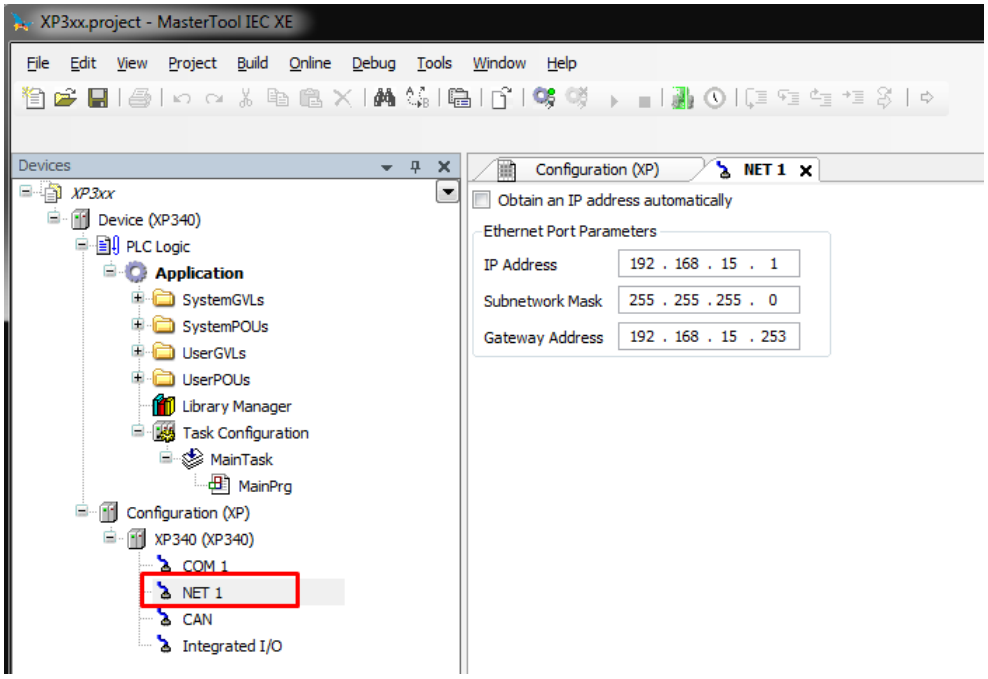


Figure 11: Configuring the Communication Port

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 Xpress 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.

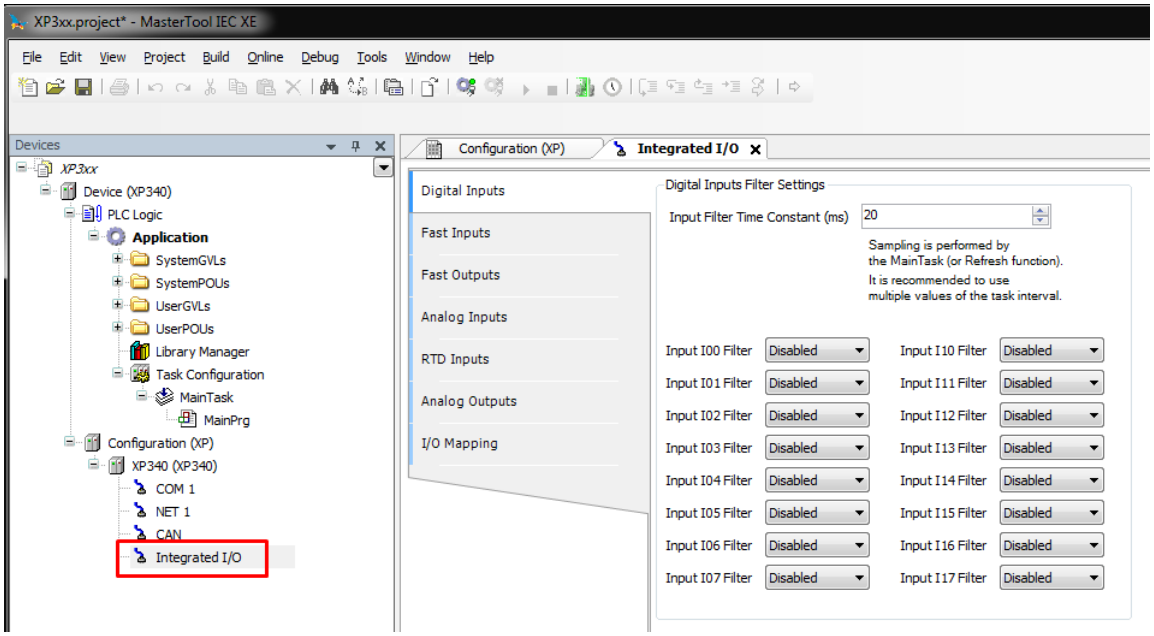


Figure 12: Configuring the Integrated I/O

4.4. Libraries

There are several programming tool 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 may be found in the MasterTool Programming Manual – MP399609.

4.5. Inserting a Protocol Instance

The Nexto Xpress controllers, as described on General Features table, offers several communication protocols. Except for the OPC 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 are presented some examples.

4.5.1. MODBUS Ethernet

The first step to configure the MODBUS Ethernet (Server in this example) is to include the instance in the desired NET (in this case, NET 1, as the XP3xx has only one Ethernet interface). Click on the NET with the mouse right button and select Add Device..., as shown on Figure 13:

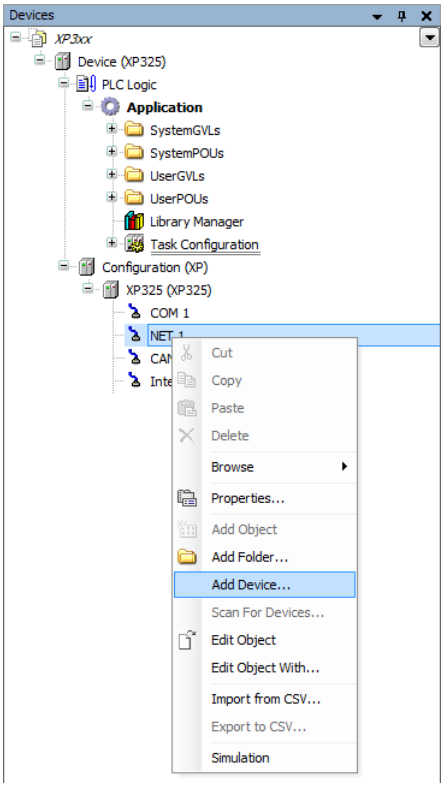


Figure 13: Adding the Instance

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

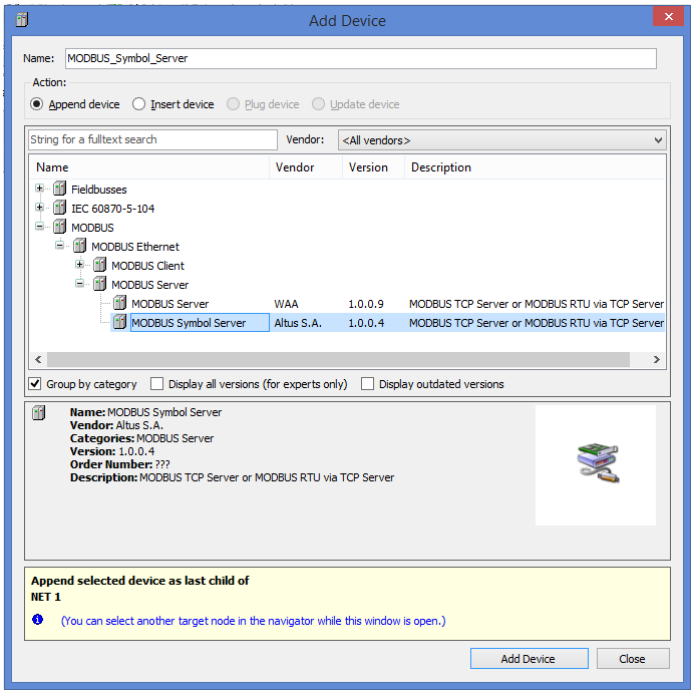


Figure 14: Selecting the Protocol

4.6. Finding the Device

To establish the communication between the controller and MasterTool IEC XE, 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*, the software MasterTool IEC XE performs a search for devices and shows the controllers found on the network of the Ethernet interface of the station where the tool is running.

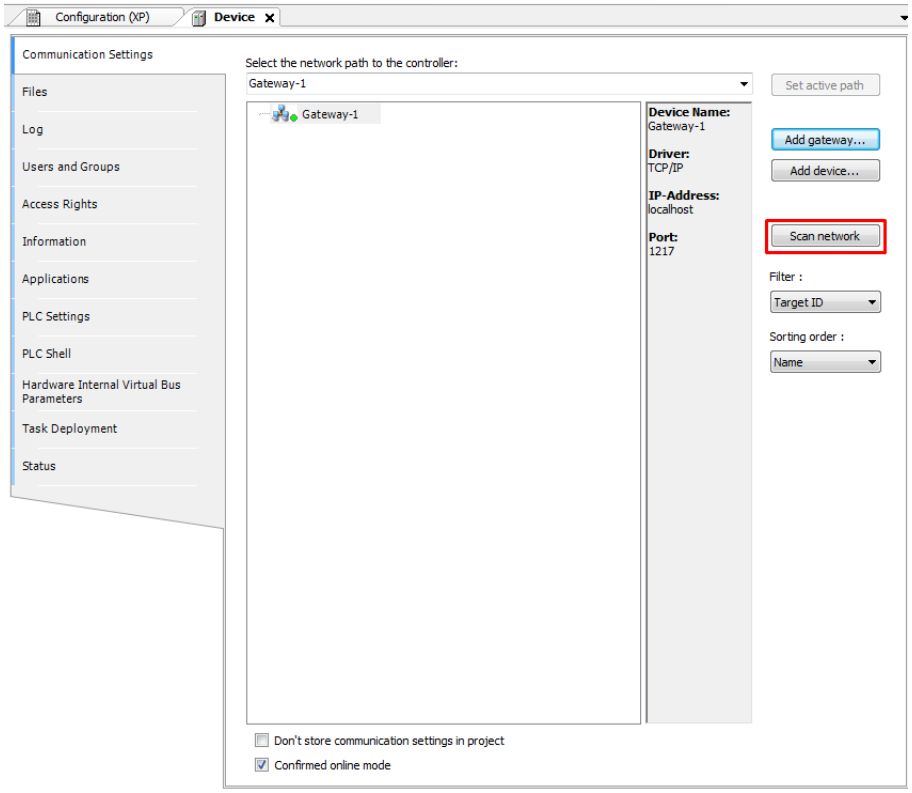


Figure 15: Finding the Device

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 clocking on *Set active path* clicked. This action selects the controller and informs the configuration software which controller shall be used to communicate and send the project.

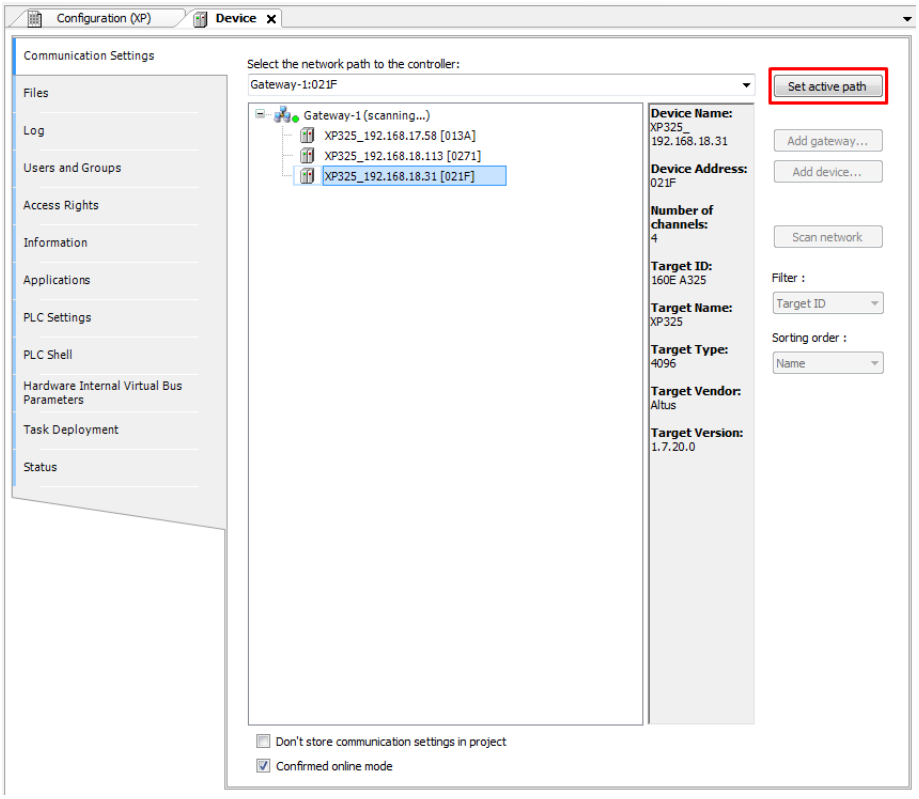


Figure 16: Selecting the controller

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 Node Name*. After a name change, the device will not return to the default name under any circunstances.

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

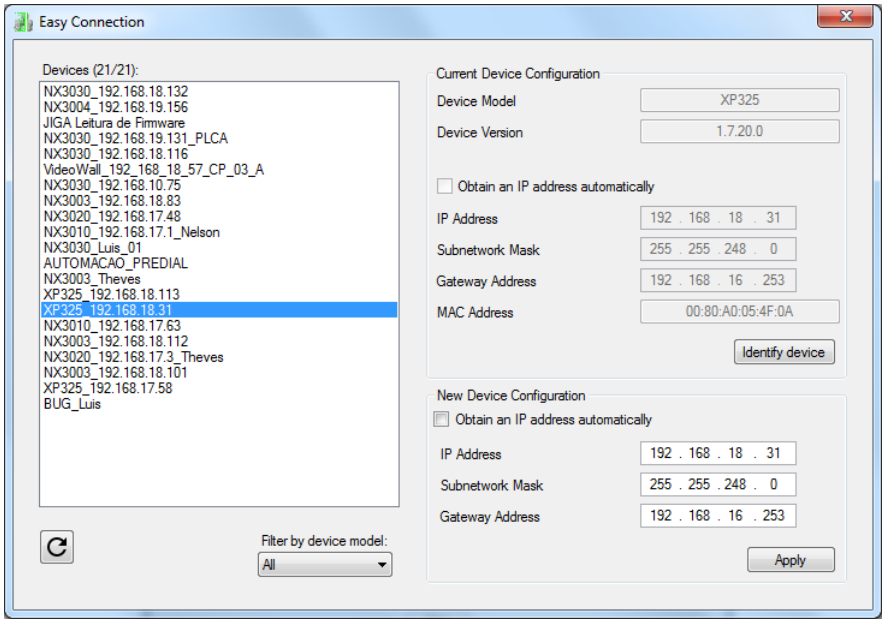


Figure 17: Easy Connection

This command performs a MAC level communication with the device, allowing to permanently change the configuration of the controller'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 IEC XE is running, allowing to find and select the device for the communication. The complete description of Easy Connection command can be found on User Manual of MasterTool IEC XE code MU299609.

4.7. Login

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

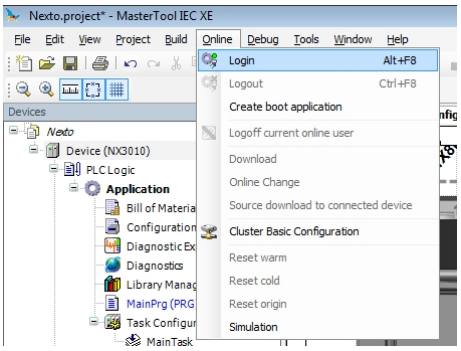


Figure 18: Sending the Project to the controller

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:

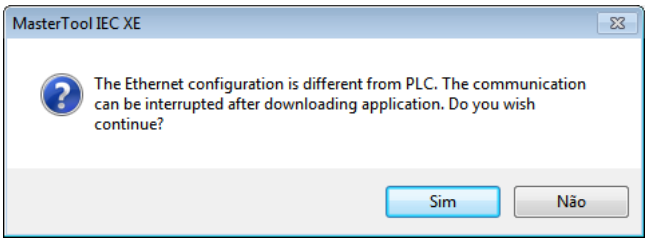


Figure 19: IP Configuration Warning

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

Figure 20: No application on the device

If there is already an application on the controller, 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 controller 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 controller stopped (see Stop Mode item). When the application is initiated, the update will have been done already
- Login without any change: executes the login without sending the new project

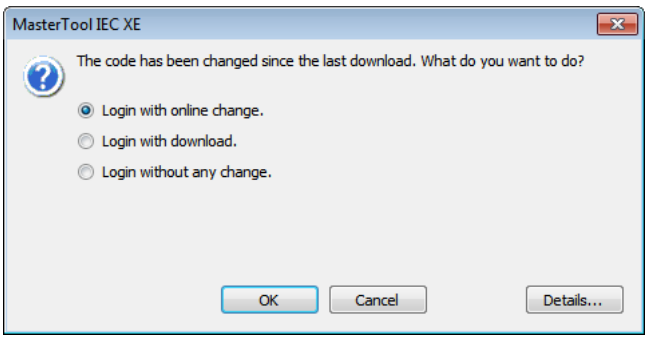


Figure 21: New application download

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, the MasterTool IEC XE requests whether the login must be executed as download (stopping the application) or if the operation must be canceled, as shown on the following figure.

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

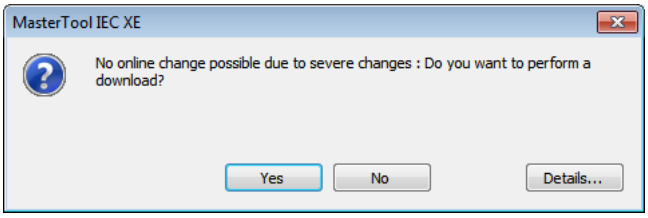


Figure 22: Configuration change

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

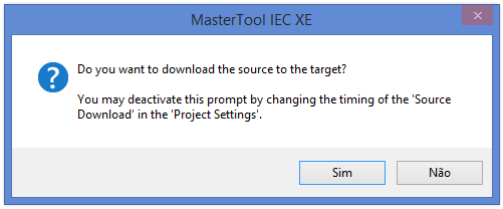


Figure 23: Source code download

Transfering 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 controller, 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 controller, allowing to pre-configure initial values which will be used by the controller on the first execution cycle.

To execute this command, simply go to the Debug menu and select the option *Start*, as shown on Figure 24.

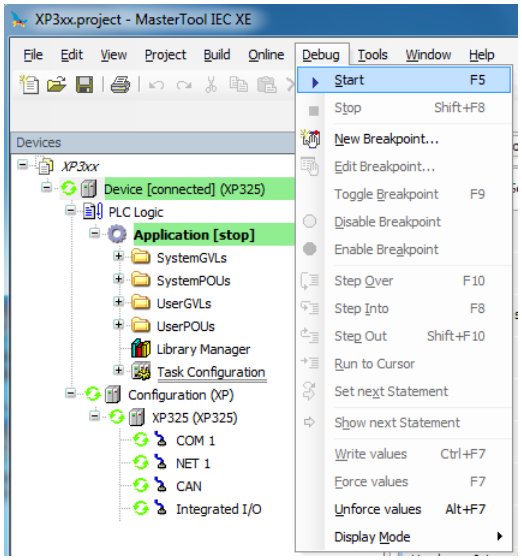


Figure 24: Starting the Application

Figure 25 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.

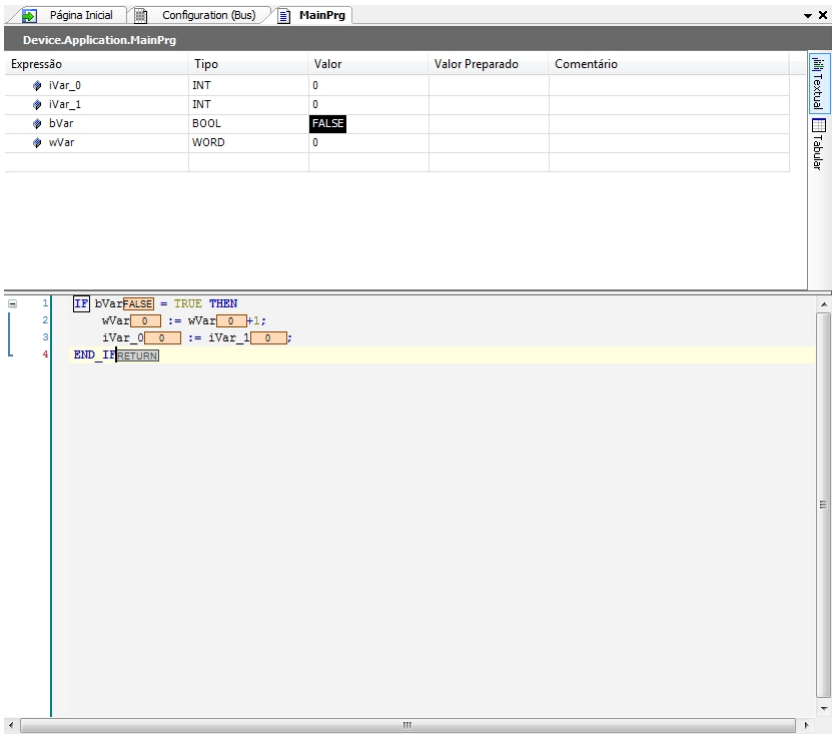


Figure 25: Program running

If the controller 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 IEC XE.

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 26.

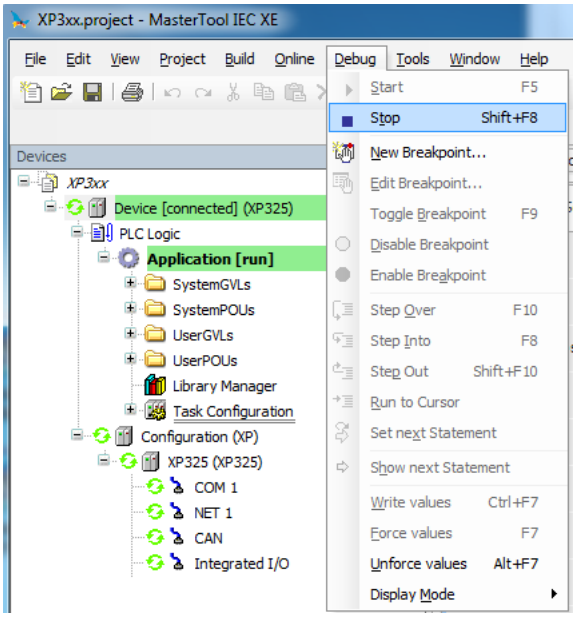


Figure 26: Stopping the Application

In case the controller 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 be released.

It is important to highlight that, when using 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 only in Online mode.
Diagnostic variables cannot be forced, only written, because diagnostics are provided by the controller and will be overwritten by it.

ATTENTION

When a controller is with forced variables and it is de-energized, the variables will lose the forcing in the next initialization.
The limit of forcing for all models of Nexto controllers is 128 variables.

4.11. Logout

To finalize the online communication with the controller, the command *Logout* must be executed, located in the *Online* menu, as shown on Figure 27.

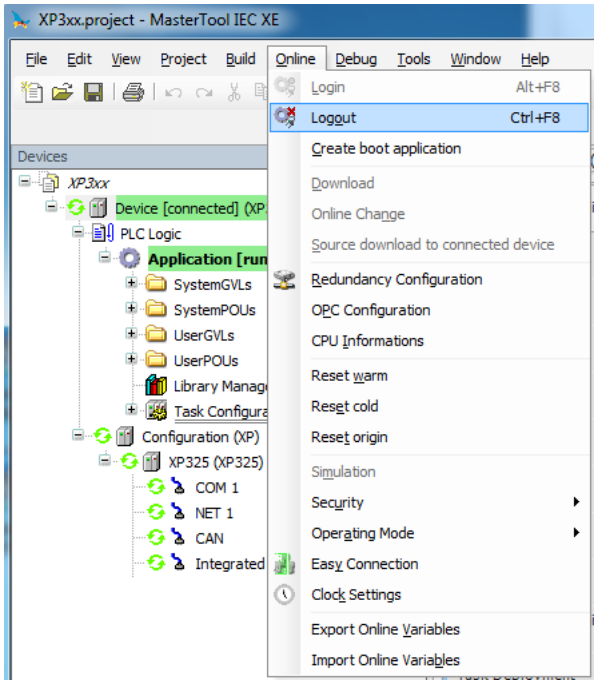


Figure 27: Ending the online communication with the controller

4.12. Project Upload

Nexto Xpress controllers 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 controller, the command located on *File* menu must be executed as shown on the following figure.

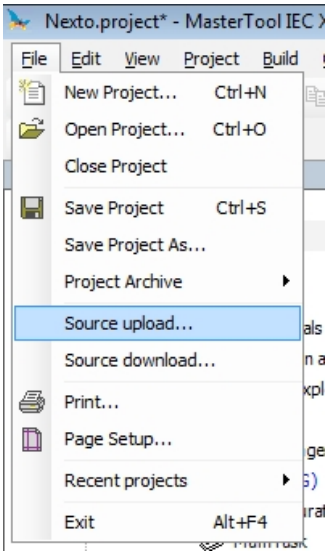


Figure 28: Project Upload Option

Next, just select the desired controller and click *OK* as shown on Figure 29.

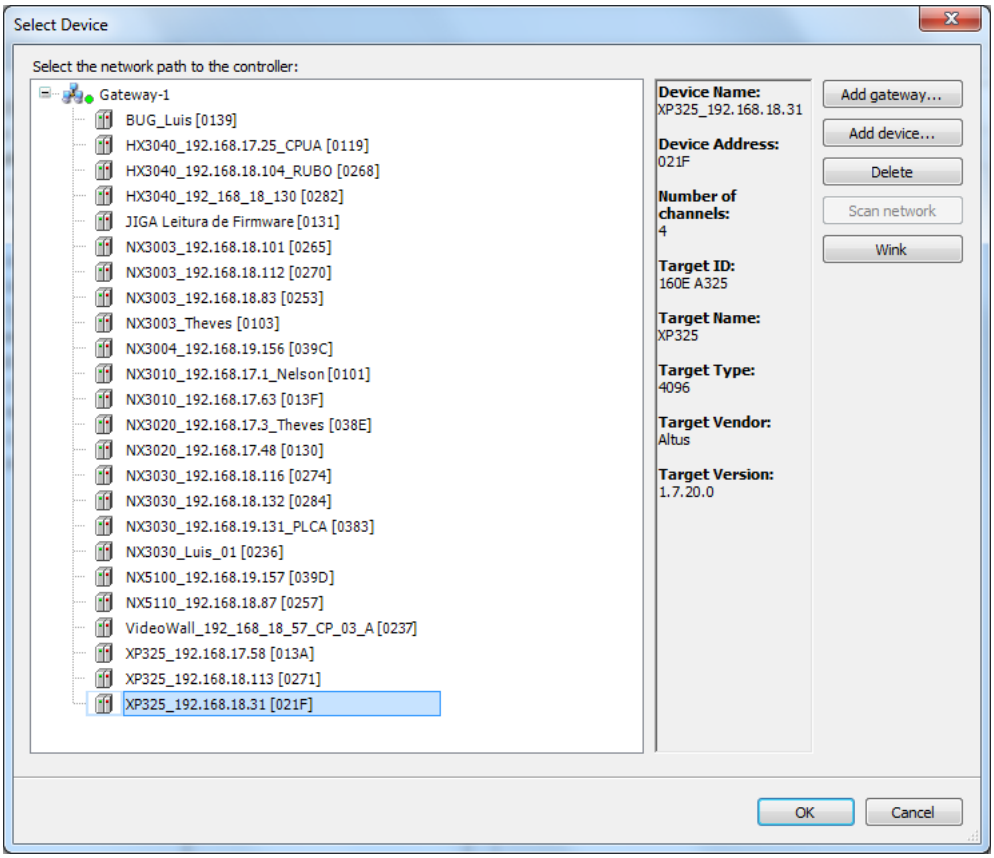


Figure 29: Selecting the controller

To ensure that the project loaded in the controller is identical and can be accessed in other workstations, consult the chapter **Projects Download/Login Method without Project Differences** at the MasterTool IEC XE User Manual MT8500 - MU299609.

ATTENTION

The memory size area to store a project in the Nexto Xpress controller is defined on General Features table.

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 transfering its source code, it will not be possible for it to be recovered by the *Upload* procedure.

4.13. CPU Operating States

4.13.1. Run

When the controller 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 the MasterTool IEC XE Utilization Manual - MU299609.

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 the application tasks variables, including the persistent type variables and deletes the CPU application.

Notes:

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

Command: To execute the commands *Reset Warm*, *Reset Cold* or *Reset Origin*, is necessary to have MasterTool IEC XE in *Online* mode with the controller.

4.13.8. Reset Process Command (IEC 60870-5-104)

This process reset command can be solicited by IEC 60870-5-104 clients. After answer the client, the CPU start a rebooting process, as if being done an energizing cycle.

The standard IEC 60870-5-104 foresee a qualification value pass (0..255) with the process reset command, but this *parameter* is not considered by the CPU.

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

The project created by MasterTool IEC XE 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 describes the main elements that are part of this standard project structure.

4.14.1. MainPrg Program

The MainTask task 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

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.

Differently from the *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 physical input and output channels integrated into the controller.

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

Device.Application.IntegratedIO

Expression	Type	Value	Prepared value	Address
DigitalInputs	T_DIGITAL_INPUTS_1			
I00	BOOL	FALSE		
I01	BOOL	FALSE		
I02	BOOL	FALSE		
I03	BOOL	FALSE		
I04	BOOL	FALSE		
I05	BOOL	FALSE		
I06	BOOL	FALSE		
I07	BOOL	FALSE		
I10	BOOL	FALSE		
I11	BOOL	FALSE		
I12	BOOL	FALSE		
I13	BOOL	FALSE		
I14	BOOL	FALSE		
I15	BOOL	FALSE		
I16	BOOL	FALSE		
I17	BOOL	FALSE		
DigitalOutputs	T_DIGITAL_OUTPUT...			
Q00	BOOL	FALSE		
Q01	BOOL	FALSE		
Q02	BOOL	FALSE		
Q03	BOOL	FALSE		
Q04	BOOL	FALSE		
Q05	BOOL	FALSE		
Q06	BOOL	FALSE		
Q07	BOOL	FALSE		
Q10	BOOL	FALSE		
Q11	BOOL	FALSE		
Q12	BOOL	FALSE		
Q13	BOOL	FALSE		
Q14	BOOL	FALSE		
Q15	BOOL	FALSE		
Q16	BOOL	FALSE		
Q17	BOOL	FALSE		
FastInputs	T_FAST_INPUTS_1			
FastOutputs	T_FAST_OUTPUTS_1			
AnalogInputs	T_ANALOG_INPUTS_1			
AI0	INT	0		
AI1	INT	0		
AI2	INT	0		
AI3	INT	0		
AI4	INT	0		
AnalogOutputs	T_ANALOG_OUTPUT...			
AO0	INT	0		
AO1	INT	0		
AO2	INT	0		
AO3	INT	0		
RtdInputs	T_RTD_INPUTS_1			
RI0	INT	0		
RI1	INT	0		

Figure 30: 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.

ATTENTION

In *System_Diagnostics* GVL, are also declared the diagnostic variables of the direct representation MODBUS Client/Master requisitions.

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

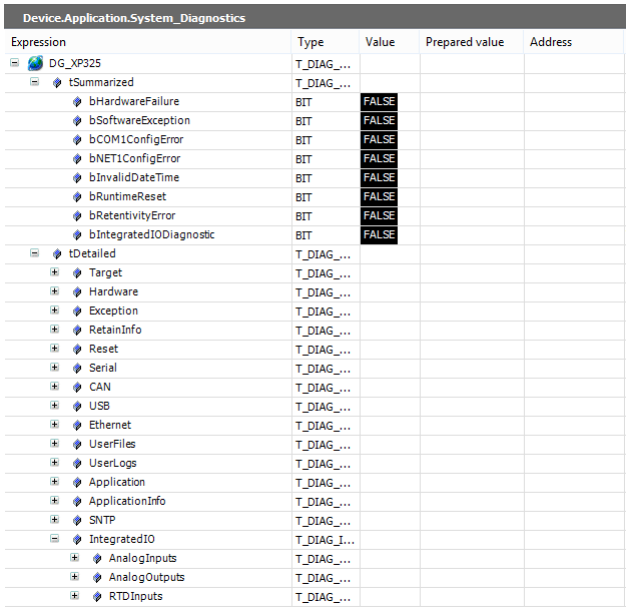


Figure 31: System_Diagnostics GVL in Online Mode

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 TreeView 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 belongs is disabled and the opposite is valid when the variable value is FALSE.

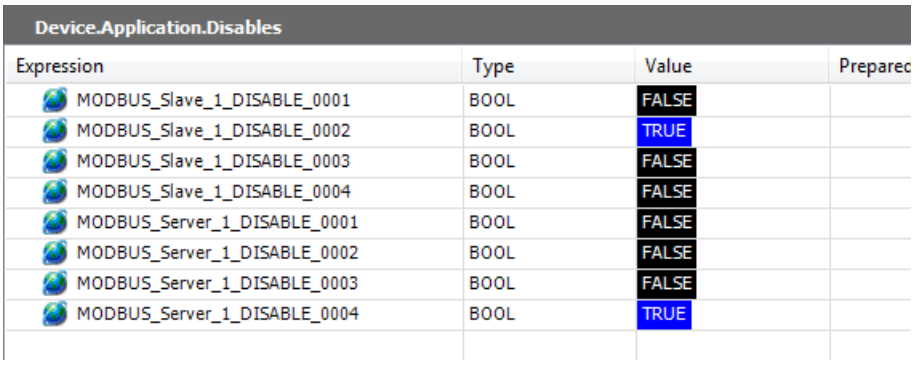


Figure 32: Disable GVL in Online Mode

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. Therefor it is recommended the use of symbolic mapping MODBUS drivers.

Examples:

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 *Quality* 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.

To the MODBUS communication devices the quality variables behave on the way showed at Table 41.

ATTENTION

If a symbolic mapping MODBUS Client/Master driver's variable be mapped in Server IEC 60870-5-104 driver, it is necessary that the MODBUS mapping quality variables had been created to generate valid quality events to such Server IEC 60870-5-104 points. Case opposite, aren't going to be generated "bad" quality events to Server IEC60870-5-104 clients in the situations that MODBUS Master/Client can't communicate with its slaves/servers, by example.

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 33: GVL Qualities in Online Mode

4.14.8. GVL ReqDiagnostics

In *ReqDiagnostics GVL*, are declared the requisition diagnostics variables of symbolic mapping MODBUS Master/Client. It is nor mandatory, but recommended the use of these variables' automatic generation, what is done by clicking in the button *Generate Diagnostic Variables* in device requisitions 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 TreeView 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 reffering 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 34: ReqDiagnostics GVL in Online Mode

5. Configuration

The Nexto Xpress controllers are configured and programmed through the MasterTool IEC XE software. The configuration defines the behavior and utilization modes for peripherals use and special features of the controller. The programming represents the application developed by the user, also known as *Application*.

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 the software MasterTool IEC XE, 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 the MasterTool IEC XE User Manual MT8500 – MU299609, 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.

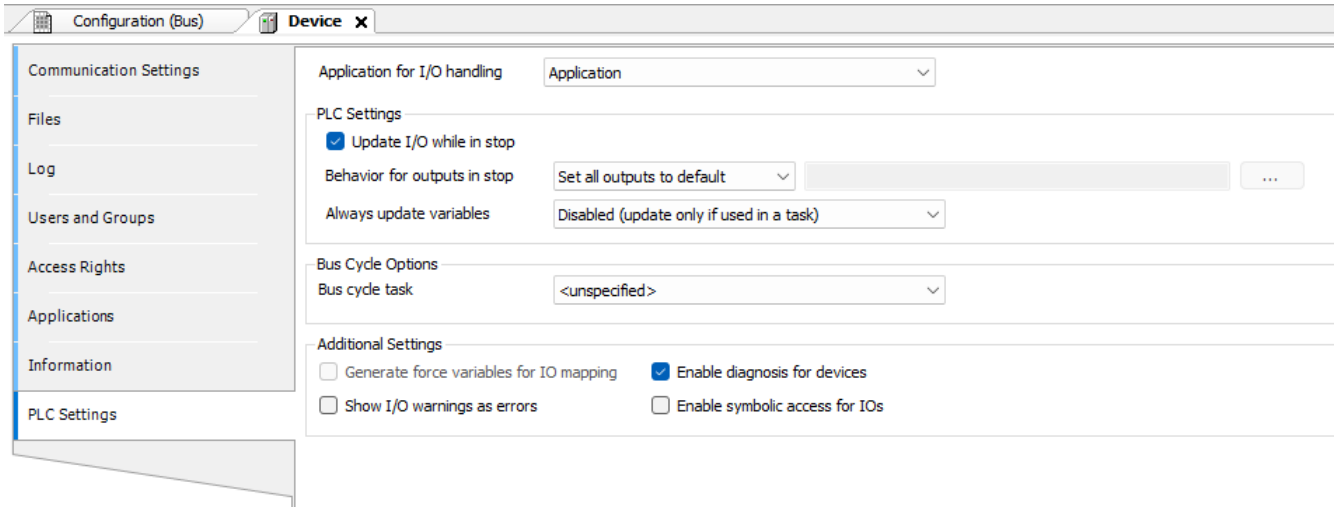


Figure 35: PLC Settings

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. **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: **Retain values:** The current values are retained. **All outputs to default value:** The default values resulting from the I/O mapping are assigned. **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. This setting is effective for the I/O variables of the slaves and modules only if "deactivated" is defined in their update settings. **Deactivated (update only if used in a task):** The I/O variables are updated only if they are used in a task. **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. **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. 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 33: 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. CPU Configuration

5.2.1. General Parameters

The parameters related to the controller's CPU are located at project treeview on item XP3xx just below *Configuration*. Each item must be properly verified for the correct project execution. These parameters are described below:

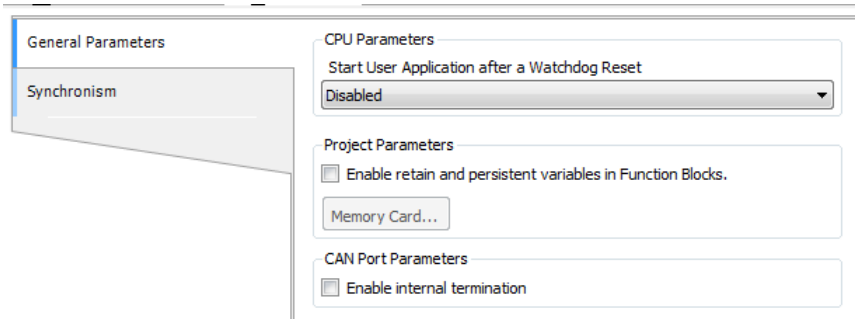


Figure 36: CPU configuration

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 Disabled
Enable retain and persistent variables in Function Blocks	Configuration to allow the use of retain and persistent variables on Function Blocks	Unmarked	Marked: allows the use of retain and persistent variables on Function Blocks. Unmarked: If this is done with this option unmarked, it may occur an exception error on startup.
Enable internal termnation	When enabled use internal termination on CAN interface.	Unmarked	Marked: enabled internal termination on CAN interface. Unmarked: disabled internal termination on CAN interface.

Table 34: CPU Configuration

5.2.2. Time Synchronization

For the time synchronization, Nexto Xpress controllers use the SNTP (Simple Network Time Protocol) protocol or the synchronism through IEC 60870-5-104.

To use the time sync protocols, the user must set the following parameters at *Synchronism* tab located at *CPU configuration* on project treeview.

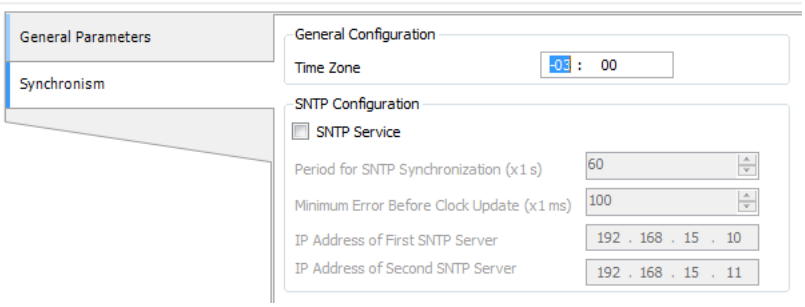


Figure 37: SNTP Configuration

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 or Enabled
Period for SNTP Synchronization (x1 sec)	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 the 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 the Second SNTP Server	IP Address of the secondary SNTP server.	192.168.15.11	1.0.0.1 to 223.255.255.254.

Table 35: 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 (IEC 60870-5-104 protocol, *SetDateAndTime* Function), others use the UTC time (SNTP). The UTC time is usually used to stamp events (IEC 60870-5-104 protocol and MasterTool Device LOG), while the local time is used by anothers CPU's features (*GetDateAndTime* function).

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 is implicitly defined a priority mechanism. The synchronism through SNTP is more prioritary than through IEC 60870-5-104 protocol. So, when both sources are enabled and SNTP server is present, it is going to be responsible for the CPU's clock sync, and any sync command from IEC 60870-5-104 is going to be denied.

5.2.2.1. IEC 60870-5-104

In case the synchronism is through IEC 60870-5-104 protocol, the user must enable the time sync at the protocol configuration screen to receive the clock synchronization. To set this option on the device, check the parameter *Enable Time Synchronization* available at the Application Layer section.

ATTENTION

If the PLC receives a time sync command from the control center, and this option is disabled, an error answer will be returned to that command. But if this option is enabled then a success message will be returned to the control center, even that the sync command be discarded for there is another synchronism method active with higher priority.

This synchronism method should be used only as an auxiliary synchronism method, once the precision of the clock sync process depends a lot on delays and traffic on the network, as well as the processor load on the CPU, as this mechanism is treated by a low priority task.

5.2.2.2. SNTP

When enabled, the controller 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. The SNTP client works with a 1 ms resolution. 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 controller sends the cyclic synchronization requests according to the time set in the SNTP Synchronization Period 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 controller 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 controller and the one received by the server is higher than the value set in the *Minimum Error Before Clock Update* parameter, the controller 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 controller waits the time for a new synchronization (*Synchronization Period*) 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 controller waits the time for a new synchronization (*Synchronization Period*) 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).

ATTENTION

The SNTP Service depends on the user application only for its configuration. Therefore, this service will be performed even when the controller is in *STOP* or *BREAKPOINT* modes since there is an application in the controller with the SNTP client enabled and properly set.

5.2.2.3. 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 of this manual.

5.2.3. Internal Points

A communication point is storage on the CPU memory under form of two distinct variables. One represents the point's value (type BOOL, BYTE, WORD, etc. . .), while another, represents its quality (type QUALITY). Internal Points are those

which the value and the quality are calculated internally by the user application, that is, they don't have an external origin like occur with points linked to IEDs (Communication drivers of type Master/Client).

This *Internal Points* configuration tab's function is to relate the variable which represents a point's value with the one which represents its quality. It must be used to relate value and quality variables internally created on the PLC program (as in a GVL), which ones typically will be afterlly mapped to a communication driver, of type *Server*, for communication with the control center.

ATTENTION

If a value variable doesn't own a related quality variable, it will be reported as default a constant good quality (no significant indication) when the value variable is reported to a client or control center.

In this way, this tab purpose isn't to create or declare internal points. To do that, just declare value and/or quality variables in a GVL and map it on the communication driver.

The internal points configuration, shown in the figure below, follow the parameters described in the table below. It's possible to configure up to 5120 entries on *Internal Points* table.

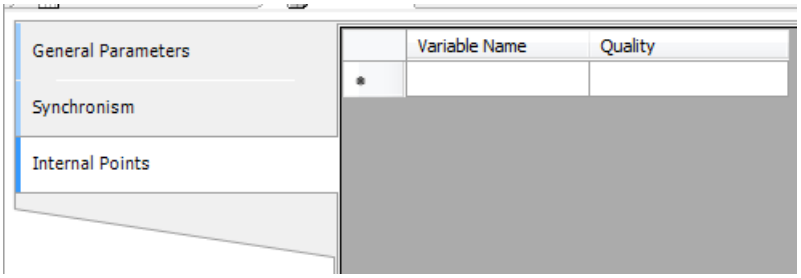


Figure 38: Internal Points Configuration Screen

Configuration	Description	Default	Options
Variable Name	Symbol variable which storage the internal point value.	-	Accept variables of type BOOL, WORD, DWORD, LWORD, INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL or DBP. The variable can be simple, array or array's element and can be part of a struct.
Quality	Symbol variable which storage the internal point quality.	-	QUALITY type variables (LibRtuStandard), which can be simple, array or array's element and can be part of a struct.

Table 36: Internal Points Configuration

The figure below show an example of two internal points configuration.

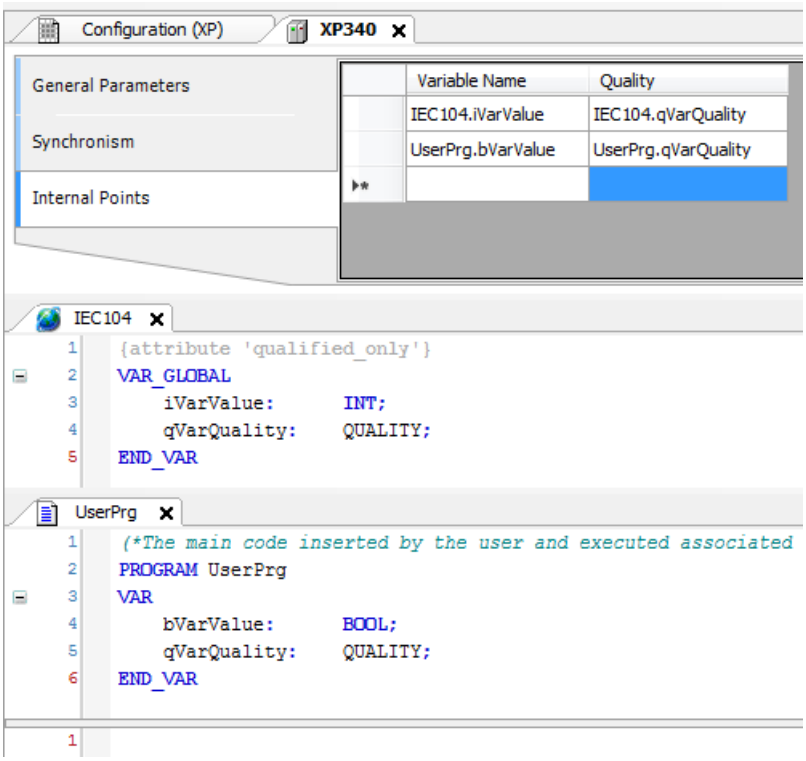


Figure 39: Internal Points Configuration Example

5.2.3.1. Quality Conversions

The internal point's quality is a trust level information about the value stored on that point. The quality may inform, for example, that the value stored is out of range, or yet that it is valid, but low trusted.

The Standards like IEC 104 have their own formats to representation of point's quality information. The Nexto Series, by its turn, have its own quality format (but quite similar to IEC 61850) called *Internal Quality*. This format is defined by type QUALITY (library *LibRtuStandard*) and it is used internally to quality storage, allowing to be done conversion between protocols without information loss.

The following tables define the protocols own formats conversion to internal format. Case it is necessary to consult the conversion between protocols, it is needed to analyze in two steps, looking each of the tables to internal format and after correlating them.

5.2.3.1.1. Internal Quality

This is the QUALITY structure. The table below shows detailed each of its components.

Bit	Name	Type	Description
0	FLAG_RESTART	BOOL	The RESTART flag indicates that the data haven't been updated by the field since the device's reset.
1	FLAG_COMM_FAIL	BOOL	Indicates there is a communication failure on the way between the data origin device and the reports device.
2	FLAG_REMOTE_SUBSTITUTED	BOOL	If TRUE the data values are overwritten in the remote communication devices.

Bit	Name	Type	Description
3	FLAG_LOCAL_SUBSTITUTED	BOOL	If TRUE the data value is overwritten by the device which generated this flag. This behavior might occur due to a working in diagnostic or temporary due to human intervention.
4	FLAG_FILTER	BOOL	Flag used to signalize and prevent the event communication channel overload. As oscillations (rapid changes) on the digital inputs.
5	FLAG_OVERFLOW	BOOL	This flag should indicates a quality problem, that the value, of the attribute to which the quality has been associated, is beyond representation.
6	FLAG_REFERENCE_ERROR	BOOL	This flag should identify that the value cannot be correct due to out of calibration reference.
7	FLAG_INCONSISTENT	BOOL	This flag should identify that an evaluation function has found an inconsistency.
8	FLAG_OUT_OF_RANGE	BOOL	This flag should indicates a quality problem that the attribute to which the quality has been associated is beyond the predefined values capacity.
9	FLAG_INACCURATE	BOOL	This flag should indicates that the value doesn't attend the declared precision of the source.
10	FLAG_OLD_DATA	BOOL	A value seems to be outdated. In case an update doesn't occur during a specific time period.
11	FLAG_FAILURE	BOOL	This flag should indicates that a watch function detected an internal or external failure.
12	FLAG_OPERATOR_BLOCKED	BOOL	Update blocked by operator.
13	FLAG_TEST	BOOL	This must be an additional identifier which can be used to classify a value being that a test value which won't be used to operational ends.
14-15	RESERVED	-	Reserved
16-17	VALIDITY	QUALITY_VALIDITY	0 – Good (Trustfull value, means that there is no abnormal conditions) 1 – Invalid (Value doesn't match the IED's value) 2 – Reserved (Reserved) 3 – Questionable (Present value might be not the same from the IED)

Table 37: QUALITY Structure

5.2.3.1.2. IEC 60870-5-104 Conversion

The tables below presents the digital, analog, Step Position, Bitstring and counters internal point's conversion to IEC 60870-5-104 of Nexto Series available to MT8500.

Internal Points -> IEC 60870-5-104 Digital		
Internal Quality		
Flags	**VALIDITY**	**IEC 60870-5-104 Quality**
FLAG_RESTART	ANY	NOT TOPICAL
FLAG_COMM_FAIL	ANY	NOT TOPICAL
FLAG_REMOTE_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_LOCAL_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_FILTER	ANY	-
FLAG_OVERFLOW	ANY	-
FLAG_REFERENCE_ERROR	ANY	-
FLAG_INCONSISTENT	ANY	-
FLAG_OUT_OF_RANGE	ANY	-
FLAG_INACCURATE	ANY	-
FLAG_OLD_DATA	ANY	NOT TOPICAL
FLAG_FAILURE	ANY	INVALID
FLAG_OPERATOR_BLOCKED	ANY	BLOCKED
FLAG_TEST	ANY	-
ANY	VALIDITY_INVALID	INVALID

Table 38: Digital Points Conversion Internal to IEC 60870-5-104

Internal Points -> IEC 60870-5-104 Analog, Step Position and Bitstring		
Internal Quality		
Flags	**VALIDITY**	**IEC 60870-5-104 Quality**
FLAG_RESTART	ANY	NOT TOPICAL
FLAG_COMM_FAIL	ANY	NOT TOPICAL
FLAG_REMOTE_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_LOCAL_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_FILTER	ANY	-
FLAG_OVERFLOW	ANY	OVERFLOW
FLAG_REFERENCE_ERROR	ANY	INVALID
FLAG_INCONSISTENT	ANY	INVALID
FLAG_OUT_OF_RANGE	ANY	OVERFLOW
FLAG_INACCURATE	ANY	INVALID
FLAG_OLD_DATA	ANY	NOT TOPICAL
FLAG_FAILURE	ANY	INVALID
FLAG_OPERATOR_BLOCKED	ANY	BLOCKED
FLAG_TEST	ANY	-
ANY	VALIDITY_INVALID	INVALID

Table 39: Analog, Step Position and Bitstring Points Conversion Internal to IEC 60870-5-104

Internal Points -> IEC 60870-5-104 Counters		
Internal Quality		
Flags	VALIDITY	IEC 60870-5-104 Quality
FLAG_RESTART	ANY	-
FLAG_COMM_FAIL	ANY	-
FLAG_REMOTE_SUBSTITUTED	ANY	-
FLAG_LOCAL_SUBSTITUTED	ANY	-
FLAG_FILTER	ANY	-
FLAG_OVERFLOW	ANY	OVERFLOW
FLAG_REFERENCE_ERROR	ANY	-
FLAG_INCONSISTENT	ANY	-
FLAG_OUT_OF_RANGE	ANY	-
FLAG_INACCURATE	ANY	-
FLAG_OLD_DATA	ANY	-
FLAG_FAILURE	ANY	INVALID
FLAG_OPERATOR_BLOCKED	ANY	-
FLAG_TEST	ANY	-
ANY	VALIDITY_INVALID	INVALID

Table 40: Counters Conversion Internal to IEC 60870-5-104

5.2.3.1.3. MODBUS Internal Quality

As the MODBUS standard don't specify quality types to each point, but for help on use of each point's communication diagnostic, MasterTool allows the quality variables mapping, through an internal own structure, to each MODBUS point. The table below describes the quality types that each MODBUS point can assume.

Resulting Quality	Resulting VALIDITY	Description
FLAG_RESTART	VALIDITY_INVALID	Initial value. The point was never updated.
-	VALIDITY_GOOD	Communication OK. The point is updated.
FLAG_COMM_FAIL AND FLAG_RESTART	VALIDITY_INVALID	Communication error. The point never was updated.
FLAG_COMM_FAIL AND FLAG_OLD_DATA	VALIDITY_QUESTIONABLE	An error has occurred but the point was updated and now has an old value.
FLAG_FAILURE AND FLAG_RESTART	VALIDITY_INVALID	It has received an exception response and the point kept its initial value.
FLAG_FAILURE AND FLAG_OLD_DATA	VALIDITY_QUESTIONABLE	It has received an exception response, but the point has a valid old value.
FLAG_RESTART AND FLAG_OLD_DATA	VALIDITY_QUESTIONABLE	Device stopped. The point has an old value.

Table 41: MODBUS Quality

5.3. Serial Interface Configuration

5.3.1. COM 1

The COM 1 interface is a RS-485 standard serial interface. It allows the point to point or network communication in the 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 and Events Serial) are automatically adjusted by MasterTool tool for the correct operation of this protocol.

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

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
Parity	Serial port parity configuration	None	Odd Even No parity
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 note on COM 1 section) - Normal Mode: Serial communication normal operation mode

Table 42: RS-485 Standard Serial Configurations

5.3.2. Advanced Configurations

The advanced configurations section allows to configure additional parameters of the serial port as described below:

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
RS-485 Termination	Enables the internal standard RS-485 termination. Must be enabled only if the controller is physically positioned at one of the extremities of the RS-485 network.	Enabled	Disabled or Enabled

Table 43: RS-485 Standard Serial Advanced Configurations

5.4. Ethernet Interface Configuration

The interface is composed by a RJ45 communication connector 10/100Base-TX standard. It allows the point to point or network communication with several protocols, for example: MODBUS, OPC UA, etc...

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

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 controller in the Ethernet bus.	192.168.15.1	1.0.0.1 to 223.255.255.254
Sub network Mask	Subnet mask of the controller in the Ethernet bus.	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Controller Gateway address in the Ethernet bus.	192.168.15.253	0.0.0.0 to 223.255.255.254

Table 44: NET 1 Configuration

5.4.2. 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 45: Reserved TCP/UDP ports

* Default port, but user changeable.

5.4.3. 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)

   If none of these conditions occurs, the operating system will forward the communication to the Default Gateway configured on the integrated NET interface.

2. Server protocols will accept connections only on the interface where it is instaciated, except for MODBUS, where a new parameter (introduced on MasterTool 3.75) called *Allow Connections from Any Interface* gives the ability to receive incoming connections from other interfaces like a USB Modem or VPN.

5.5. CAN Interface Configuration (Controller Area Network)

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 46: CAN Configuration

The parameters related to CANopen protocol are described on Communication Protocols section.

5.6. Integrated I/O Configuration

Nexto Xpress 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:

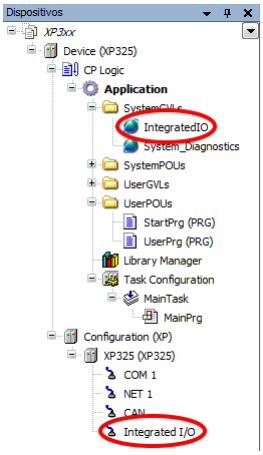


Figure 40: Integrated I/O objects on project treeview

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

The other object is the connector *Integrated I/O*, which contains the configuration for each type of I/O point. These configurations will be detailed on next sections.

5.6.1. Digital Inputs

The parameters related to the *Digital Inputs* are located on the screen below (example from XP325), for both standard and fast inputs (when configured as standard digital inputs):

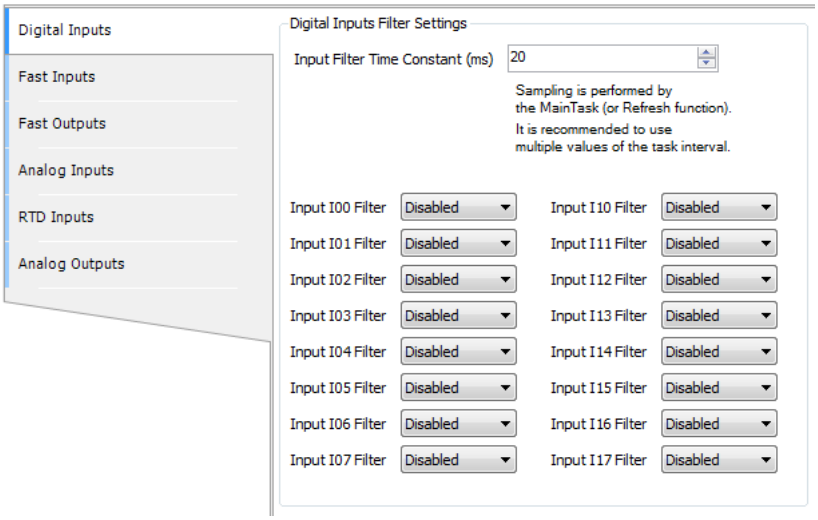


Figure 41: Digital Inputs Parameters

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 47: Digital Inputs Parameters

Note:

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

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			
		I00	I01	I02	I03
Counter 0	Up/Down (A count, B direction) with zero	B	Z	A	-
	Quadrature 2X	A	B	-	-
	Quadrature 2X with zero	A	B	Z	-
	Quadrature 4X	A	B	-	-
	Quadrature 4X with zero	A	B	Z	-

Table 48: 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:

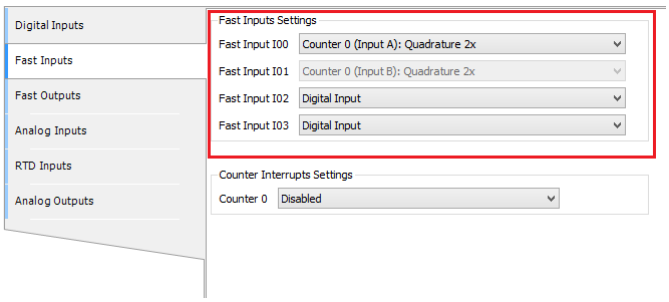


Figure 42: Fast Inputs settings

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:

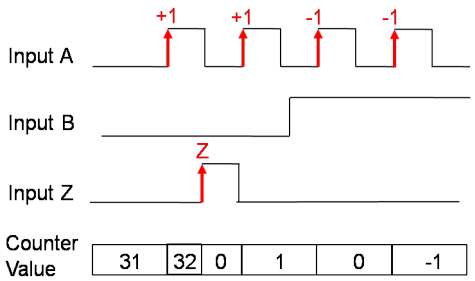
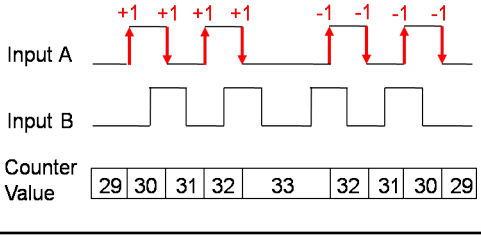
Configuration	Description	Default	Options
Fast Input I00	Fast Input I00 configuration	Digital Input	Digital Input Counter 0 (Input B): Up/Down (A count, B direction) with zero Counter 0 (Input A): Quadrature 2X Counter 0 (Input A): Quadrature 2X with zero Counter 0 (Input A): Quadrature 4X Counter 0 (Input A): Quadrature 4X with zero
Fast Input I01	Fast Input I01 configuration	Digital Input	Digital Input Obs: This field will be set automatically when Fast Input I00 is configured as Up/Down or Quadrature Counter.
Fast Input I02	Fast Input I02 configuration	Digital Input	Digital Input Interruption (Rising Edge) Obs: This field will be set automatically when Fast Input I00 is configured as Up/Down or Quadrature Counter with zero.
Fast Input I03	Fast Input I03 configuration	Digital Input	Digital Input Interruption (Rising Edge)

Table 49: Fast Inputs Parameters

Even if a fast input is configured as a counter or interruption, it's digital value can still be read through *IntegratedIo.DigitalInputs* variable. The next subsections give more details about the *Fast Inputs* configuration and programming.

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	

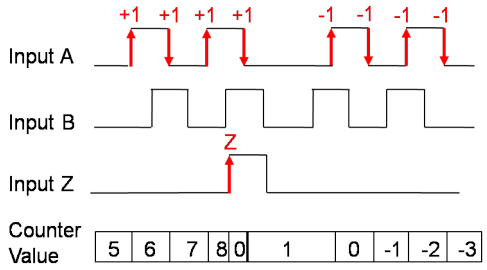
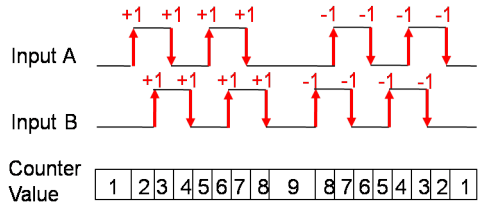
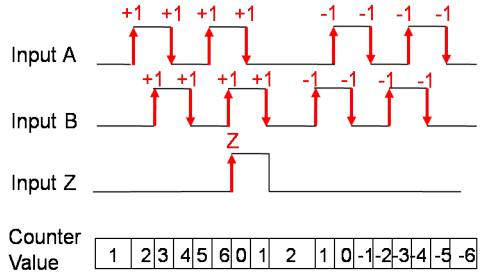
Counter Mode	Counting waveforms
Quadrature 2X with zero	
Quadrature 4X	
Quadrature 4X with zero	

Table 50: 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:

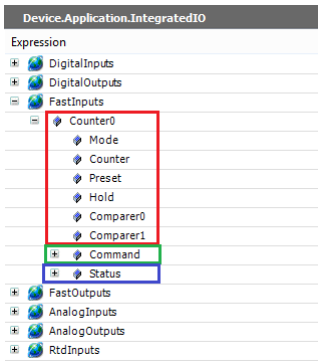


Figure 43: Counter structure

The table below describes the counter variables structure:

Variable	Description	Type	Allowed Values
Mode	Configured counter mode (read only)	ENUM_COUNTER _MODE	DISABLED UP_DOWN_A_COUNT_B _DIR_WITH_ZERO QUADRATURE_2X QUADRATURE_2X _WITH_ZERO QUADRATURE_4X 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 51: 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
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 52: 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

Variable	Description	Type	Allowed Values
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 53: 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.

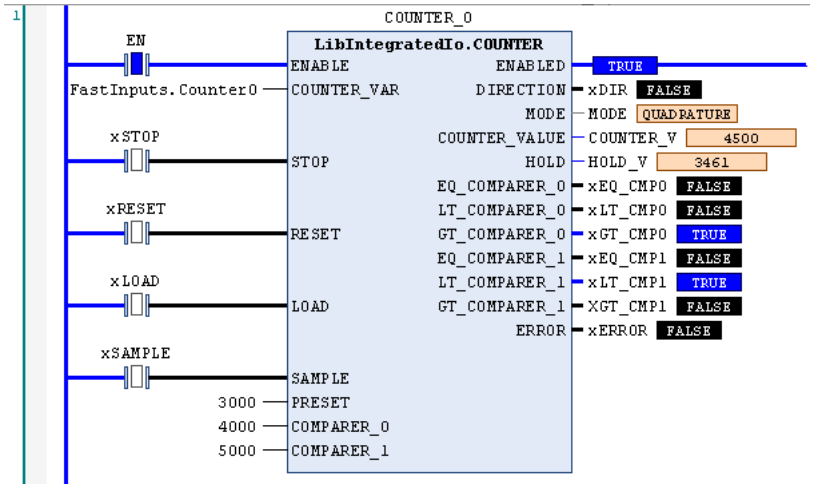


Figure 44: LibIntegratedIo.COUNTER function block

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

Variable	Description	Type	Allowed Values
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 54: 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:

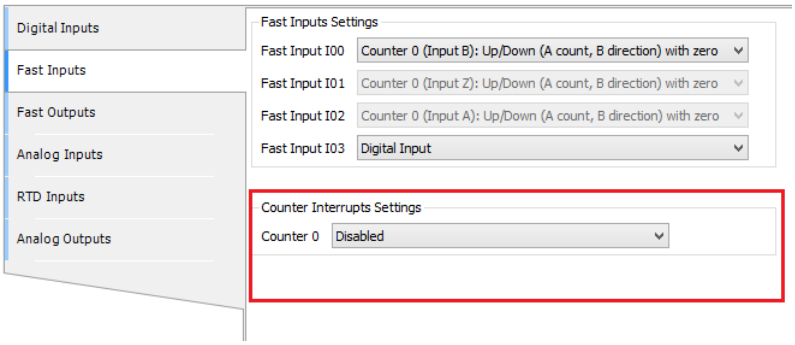


Figure 45: Counter interrupt settings

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

Configuration	Description	Default	Options
Counter 0	Counter0 comparator interrupt configuration	Disabled	Disabled Counter0InterruptTask: Counter equal to Comparer0 Obs: This configuration is available when the Counter0 is associated to some Fast Input.

Table 55: 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 IEC XE.

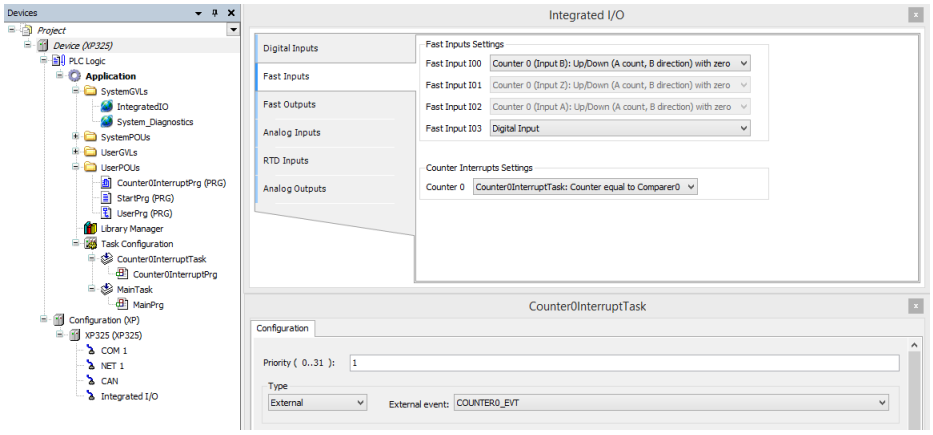


Figure 46: Counter Interrupt Settings

5.6.2.2. External Interruption

The fast inputs can be set as *Interruption (Rising Edge)* mode, which means that when a rising edge (0V to 24V transition) is performed on the input, an 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 I02 is called *FIN2_EVT*. So, an external event task called *FastInputI02InterruptTask* must be configured to be triggered by this event, and must call a POU called *FastInputI02InterruptPrg* which will contain the user program to be executed.

The figure below shows this configuration scenario in MasterTool IEC XE.

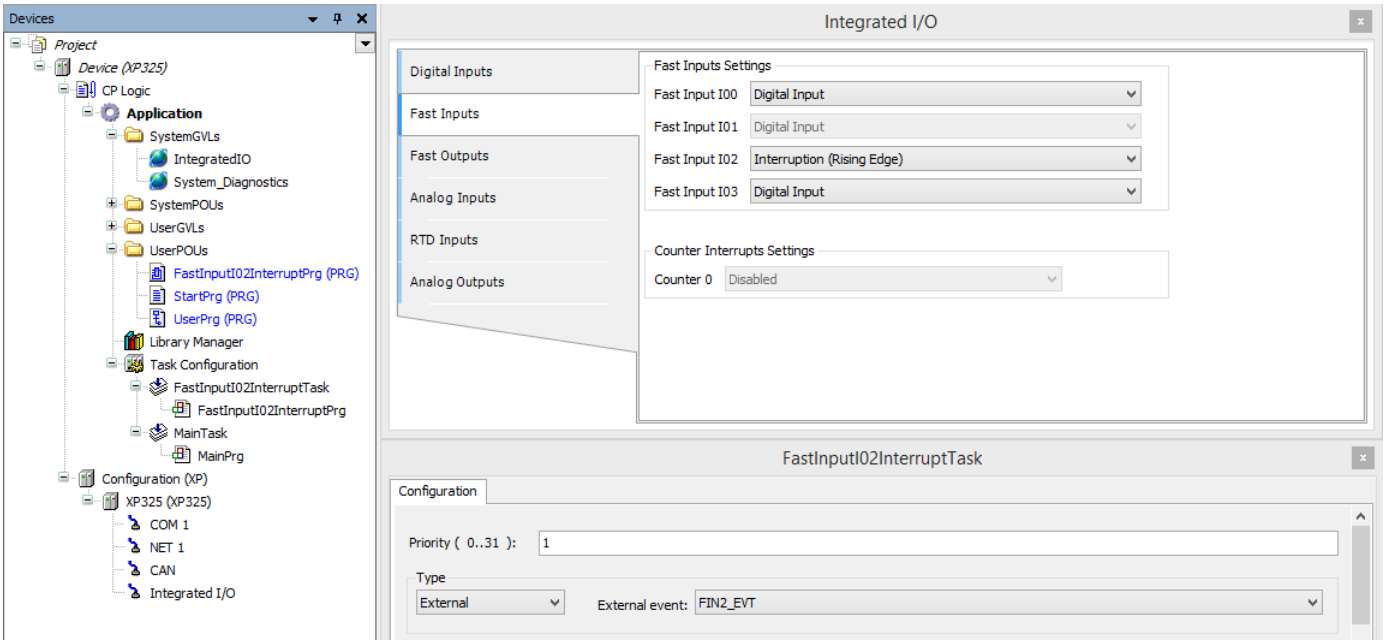


Figure 47: Fast Inputs Interruption Settings

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:

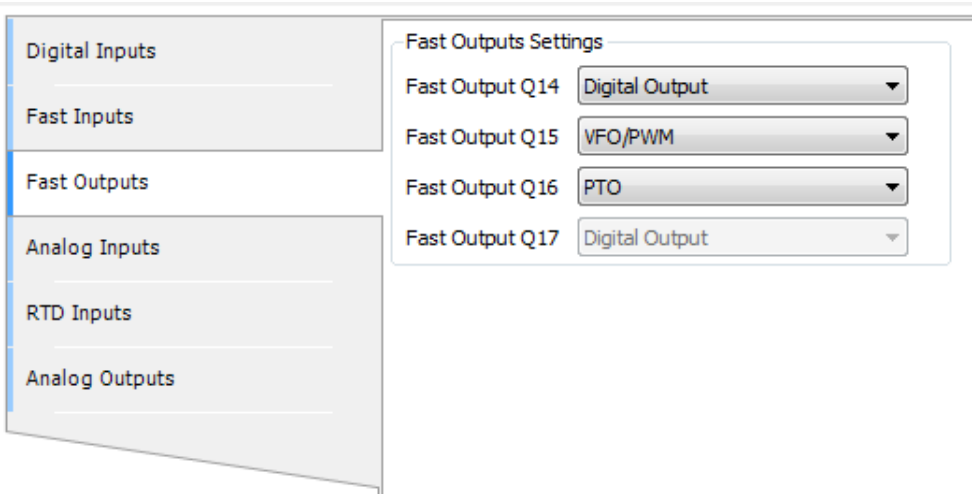


Figure 48: Fast Outputs Parameters

The table below shows the possible configuration values:

Configuration	Description	Default	Options
Fast Output Q14	Fast Output Q14 configuration.	Digital Output	Digital Output VFO/PWM PTO
Fast Output Q15	Fast Output Q15 configuration.	Digital Output	Digital Output VFO/PWM
Fast Output Q16	Fast Output Q16 configuration.	Digital Output	Digital Output VFO/PWM PTO
Fast Output Q17	Fast Output Q17 configuration.	Digital Output	Digital Output VFO/PWM

Table 56: Fast Outputs Parameters

The PTO function can be assigned only for Q14 and Q16. When the output is configured on this mode, the adjacent output (Q15 or Q17) will be forced to standard digital output mode.

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 global variable *IntegratedIo.DigitalOutputs*.

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

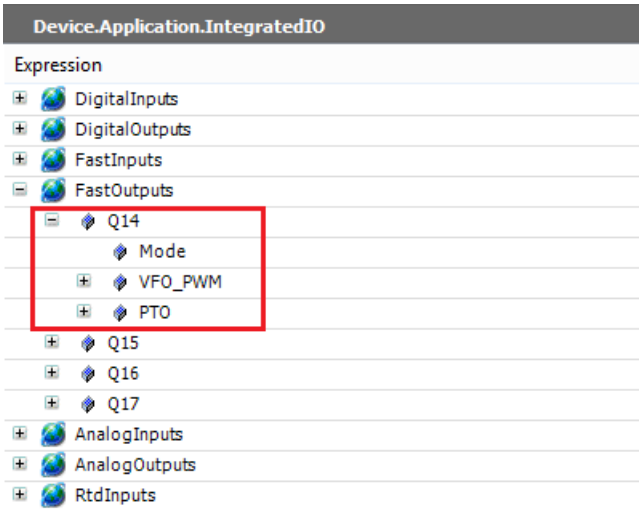


Figure 49: Fast Output structure

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 PWM 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 57: 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:

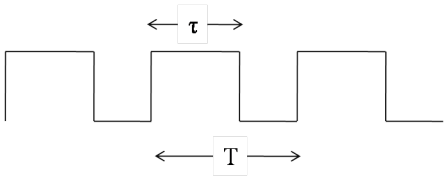


Figure 50: VFO/PWM waveform

The figure shows a pulsed waveform, where T is the period of the pulses and τ 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	1 to 200000
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 58: 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 59: 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 60: 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.

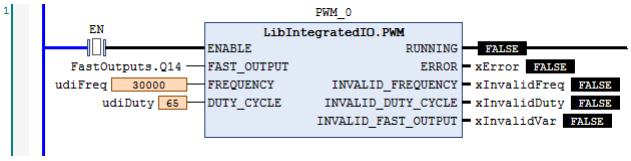


Figure 51: LibIntegratedIo.PWM function block

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.Q14 FastOutputs.Q15 FastOutputs.Q16 FastOutputs.Q17
FREQUENCY	Frequency in Hertz.	UDINT	1 to 200000
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 61: 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.

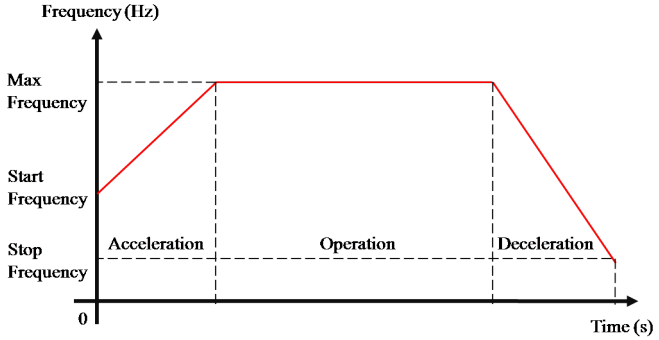


Figure 52: PTO with trapezoidal profile

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.

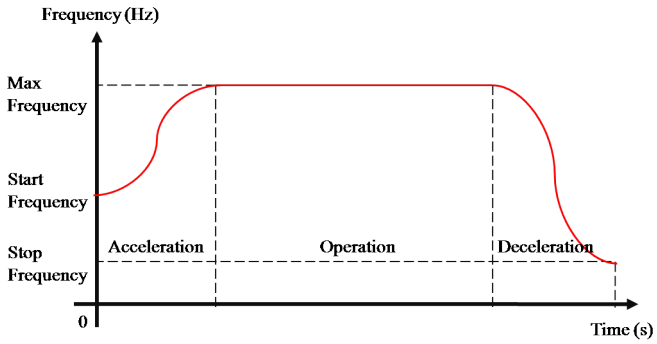


Figure 53: PTO with “S” profile

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.

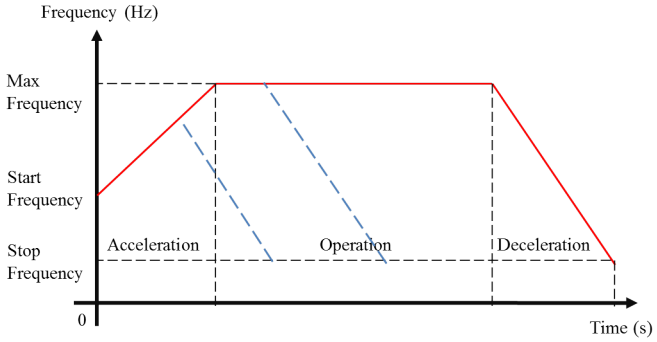


Figure 54: PTO Softstop on trapezoidal profile

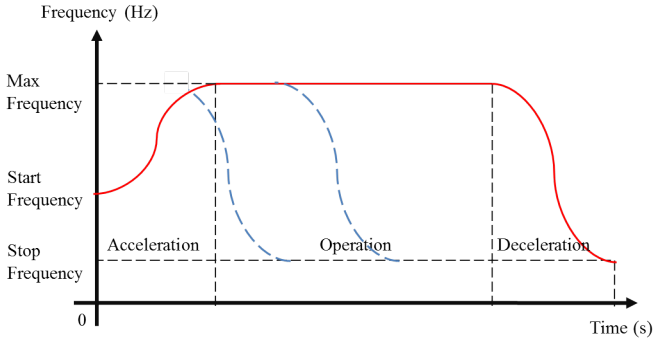


Figure 55: PTO Softstop on "S" profile

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	0 to 200000
StopFrequency	Stop frequency in Hertz	UDINT	0 to 200000
MaxFrequency	Maximum frequency in Hertz	UDINT	1 to 200000
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 4294967295
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 62: 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 63: 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 64: 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.

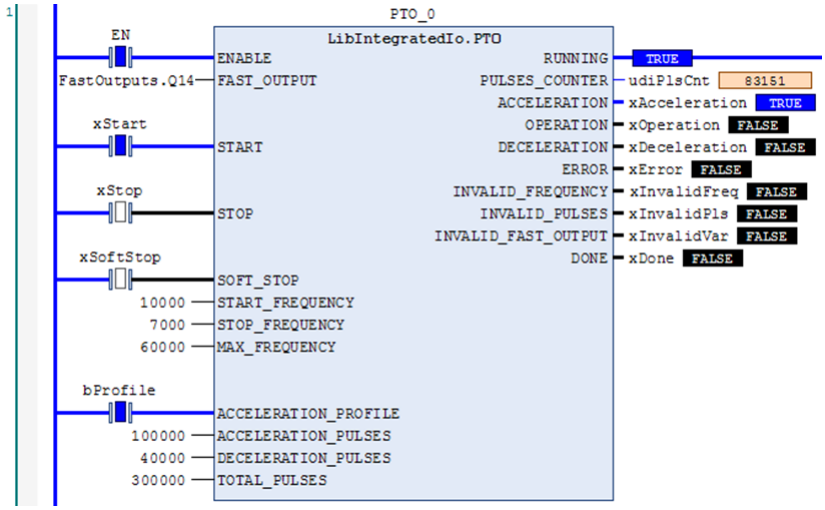


Figure 56: LibIntegratedIo.PTO function block

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.Q14 FastOutputs.Q15 FastOutputs.Q16 FastOutputs.Q17
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	1 to 200000
STOP_FREQUENCY	Stop frequency in Hertz	UDINT	1 to 200000
MAX_FREQUENCY	Maximum frequency in Hertz	UDINT	1 to 200000
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 4294967295
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 65: LibIntegratedIo.PTO function block description

5.6.4. Analog Inputs

The parameters related to the Analog Inputs are shown below:

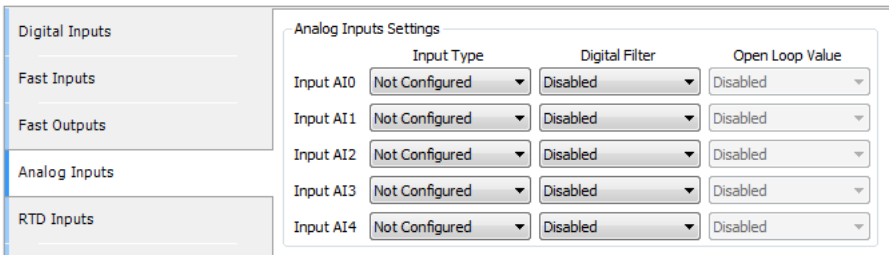


Figure 57: Analog Inputs Parameters

The table below shows the possible configuration values:

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

Table 66: 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. RTD Inputs

The parameters related to the RTD Inputs are shown below:

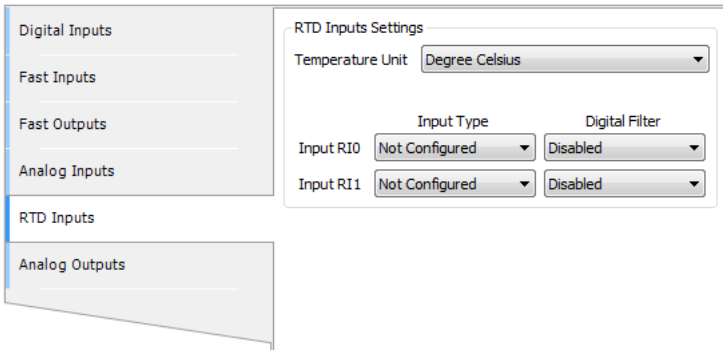


Figure 58: RTD Inputs Parameters

The table below shows the possible configuration values:

Configuration	Description	Default	Options
Temperature Unit	Selects the temperature unit	Degree Celsius	Degree Celsius Degree Fahrenheit
Input Type	Selects the input type	Not configured	Not configured 400 Ω 4000 Ω Pt100A Pt100E Pt1000A Pt1000E
Digital Filter	Enable/Disable a 1st order low pass digital filter for each input	Disabled	Disabled 100 ms 1 s 10 s

Table 67: RTD Inputs Parameters

The next table describes additional details about each input type:

Input type	Temperature Coefficient (α)	Measurement Band	Count	Resolution
400 Ω	-	0 to 400 Ω	0 to 4000	0.1 Ω
4000 Ω	-	0 to 4000 Ω	0 to 4000	1 Ω
Pt100E, Pt1000E	0,00385	-200 to 850 °C -328 to 1562 °F	-2000 to 8500 -3280 to 15620	0.3 °C 0.6 °F
Pt100A, Pt1000A	0,003916	-200 to 630 °C -328 to 1166 °F	-2000 to 6300 -3280 to 11660	0.3 °C 0.6 °F

Table 68: RTD Input Types

5.6.6. Analog Outputs

The parameters related to the Analog Outputs are shown below:

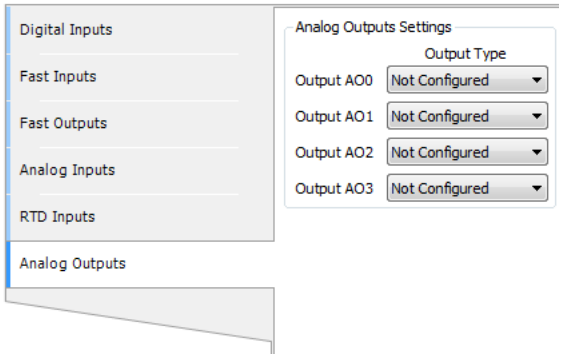


Figure 59: Analog Outputs Parameters

The table below shows the possible configuration values:

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

Table 69: Analog Outputs Parameters

5.6.7. I/O Mapping

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

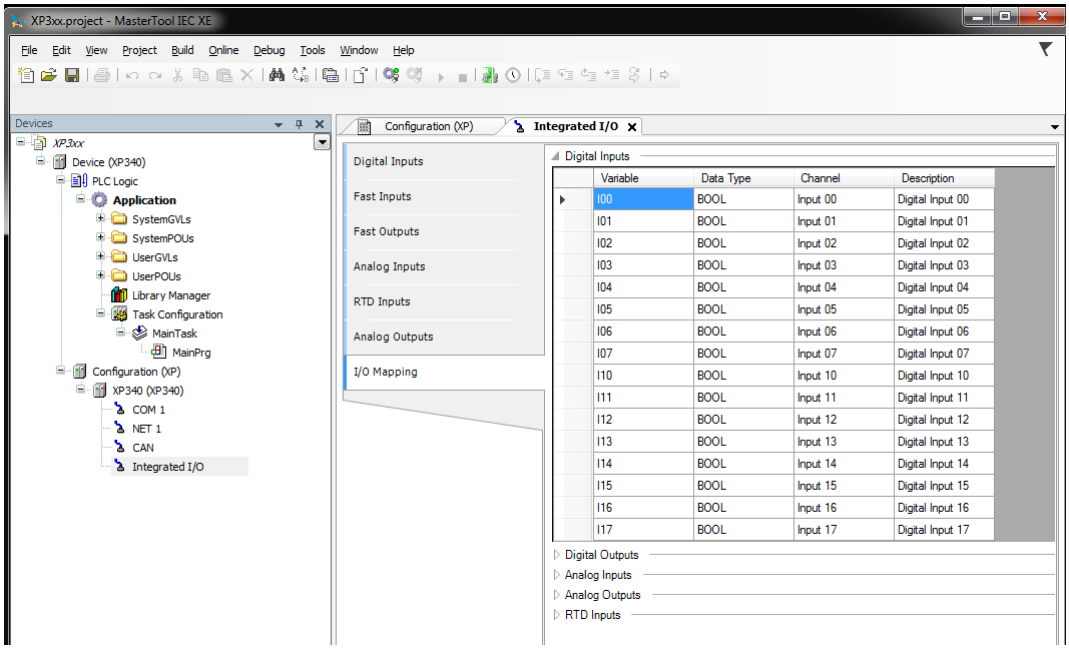


Figure 60: I/O tag mapping

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.

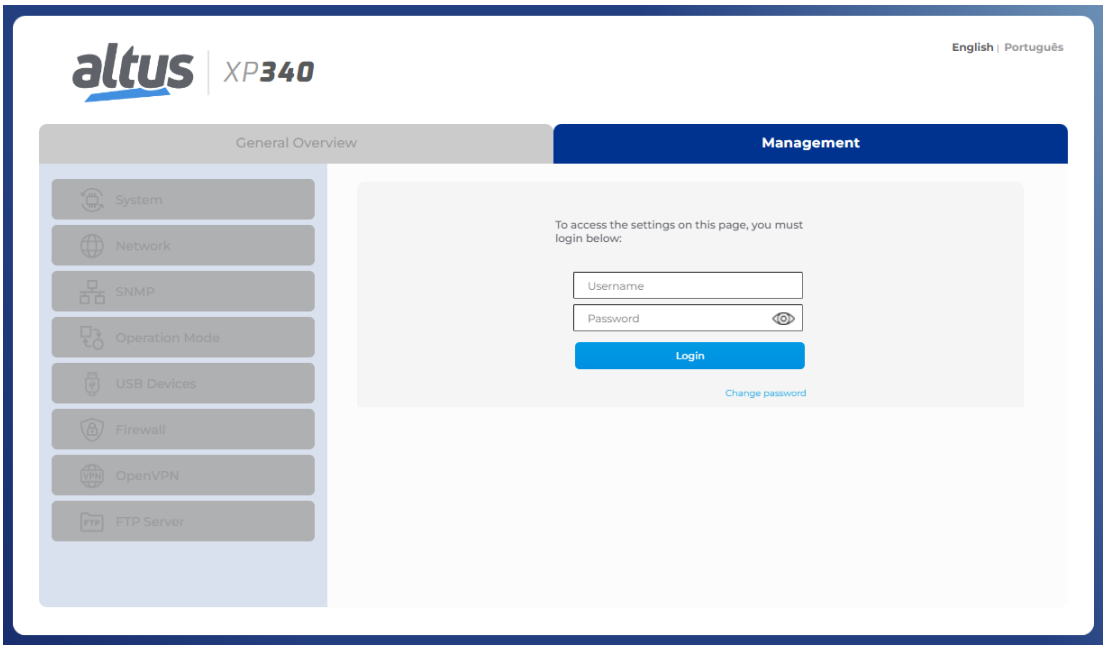


Figure 61: Management Tab Access

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:

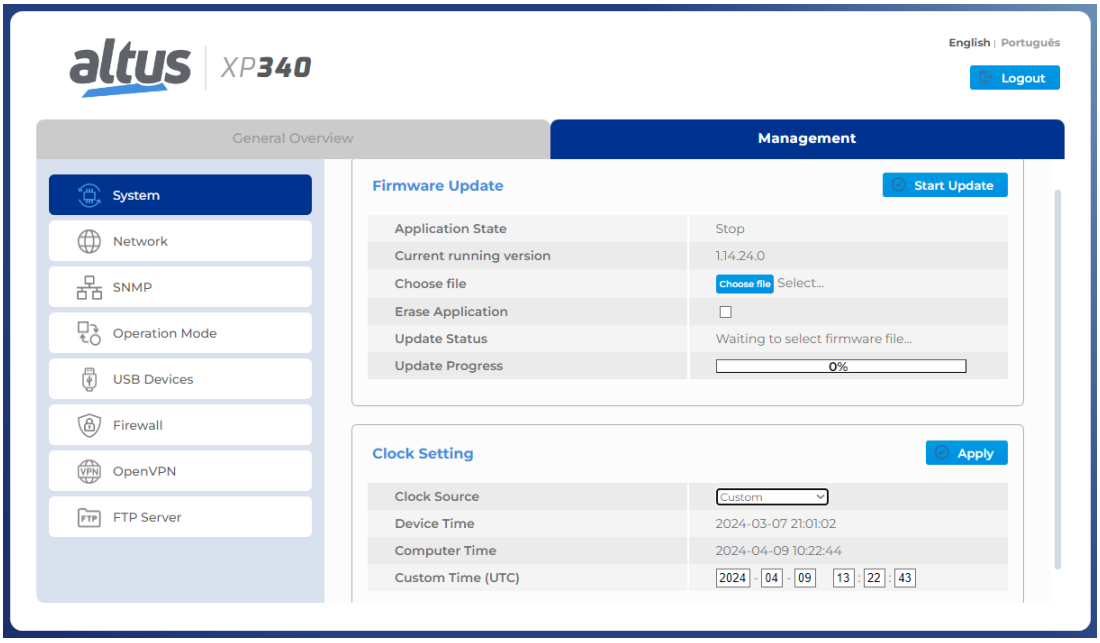


Figure 62: Clock Setting

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.

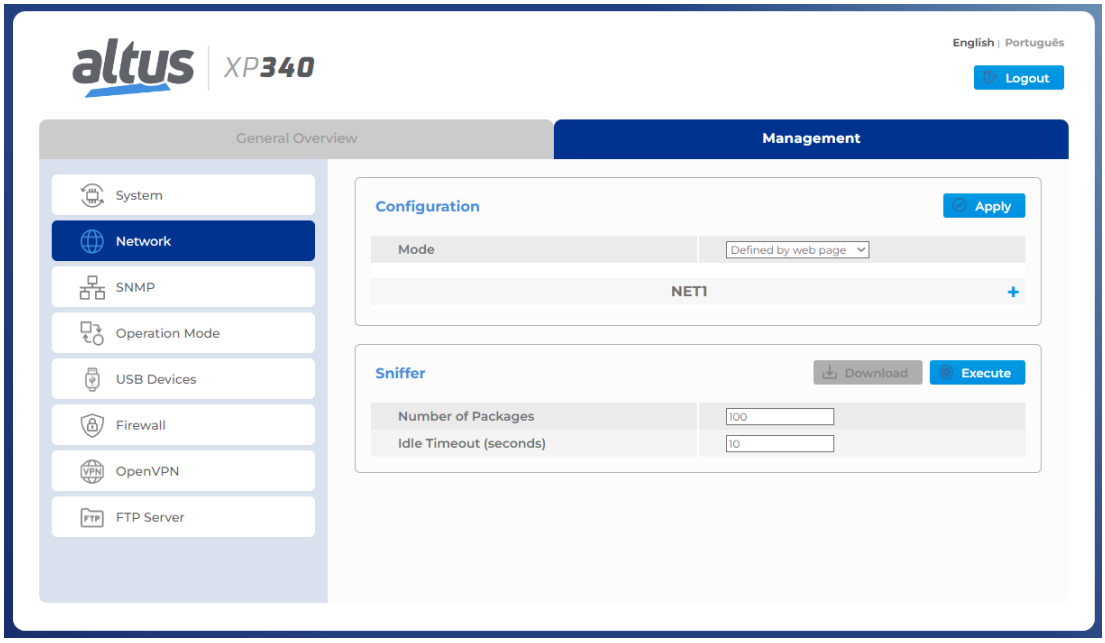


Figure 63: Network Section

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.

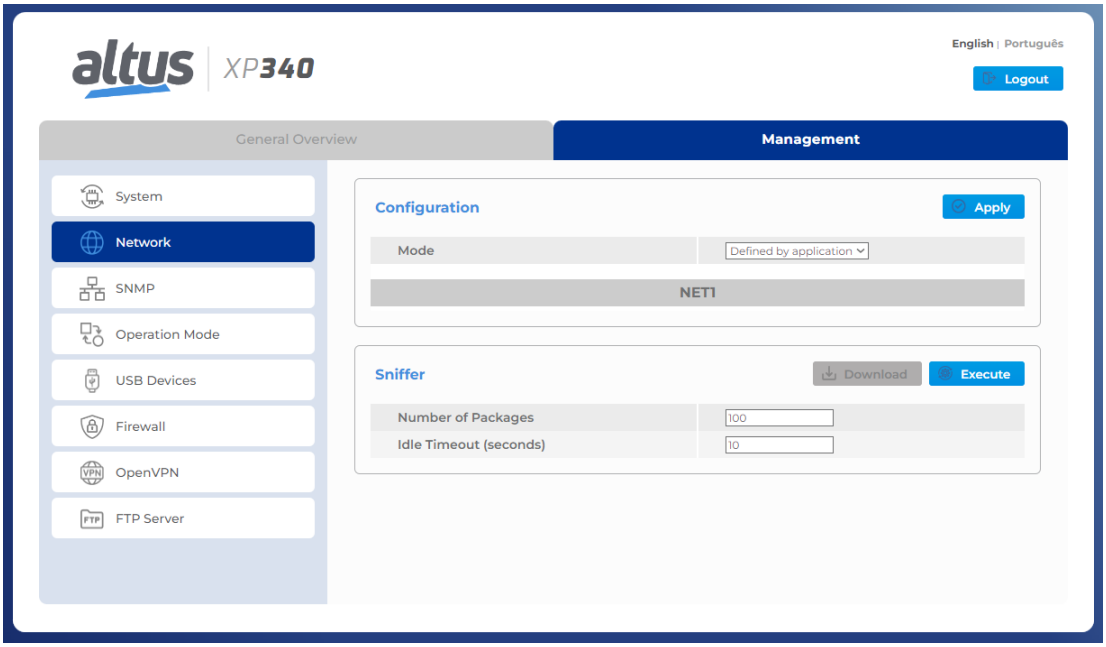


Figure 64: Interfaces Table - Application Mode

5.7.2.1.2. *Defined by web page*

For the *Defined by web page* mode, the interface table remains enabled, as shown in the figure below.
In this mode, the user can configure the IP Address, Netmask, and Gateway of each of the available Ethernet interfaces.

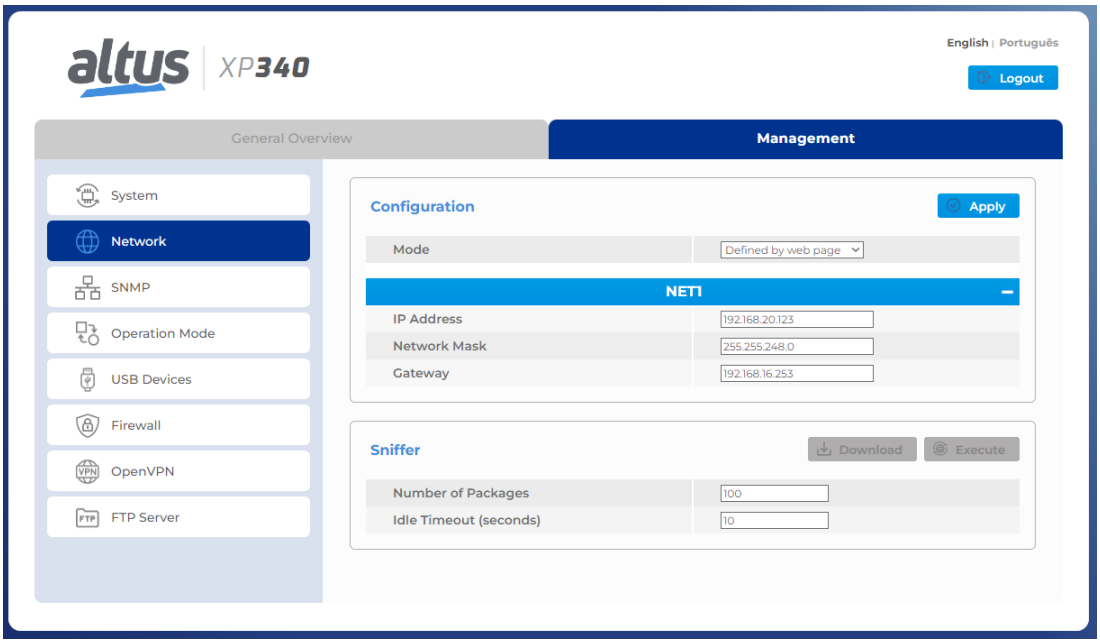


Figure 65: Interfaces Table - Web Mode

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 your settings are correct, after clicking *Apply*, a confirmation window appears in your 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. This is especially true 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.

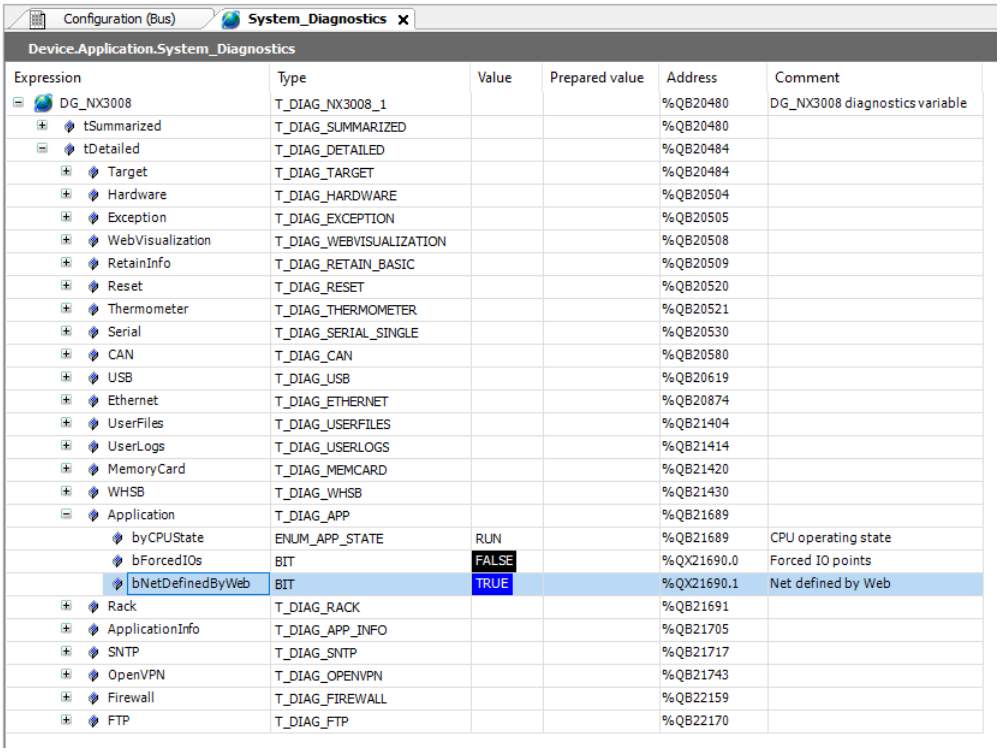


Figure 66: Diagnostics - IP defined by the Web Page

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.

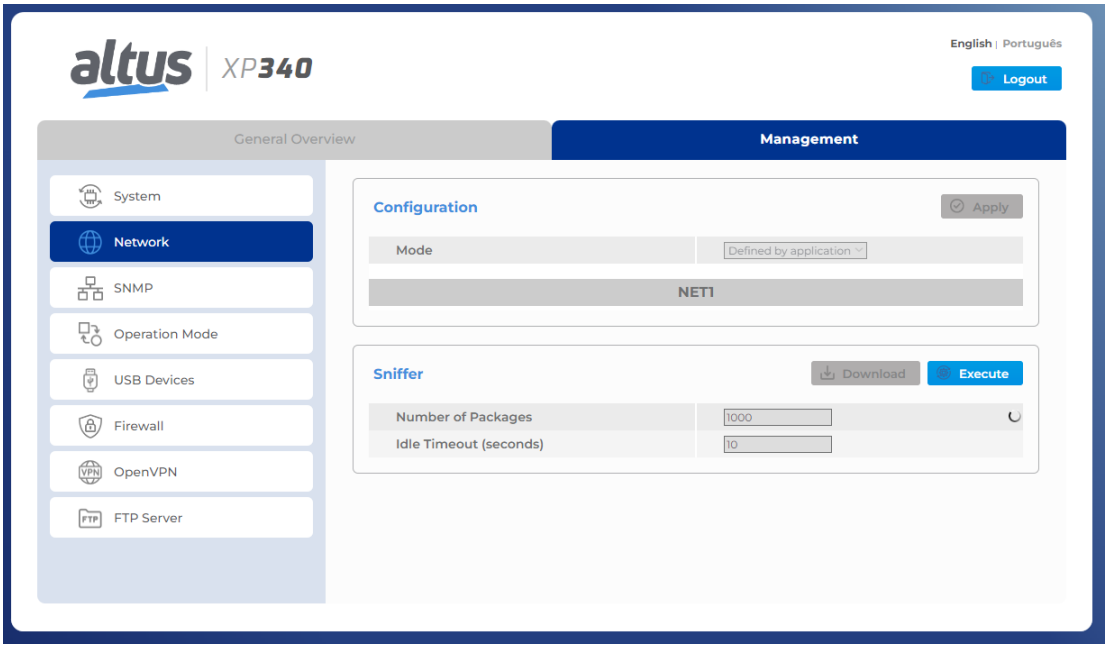


Figure 67: Network Sniffer Running

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

For each of the interfaces on which Sniffer runs, it generates a **.pcap** file. These files are named according to the name of the controller and the interface that was analyzed, for example, **`[PLC_MODEL]_capture.zip`**. These files are found inside a **.zip** file, also named according to the name of the controller, for example, **`[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 Interface Configuration

The USB Host port present on Nexto Xpress controllers allows to extend the controller's functionalities by using several types of USB dongles. Due to the wide range of USB devices available on the market (flash drives, Ethernet/Wifi adapters, 3G/4G modem, etc...), the support for each specific device is provided by a firmware update.

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

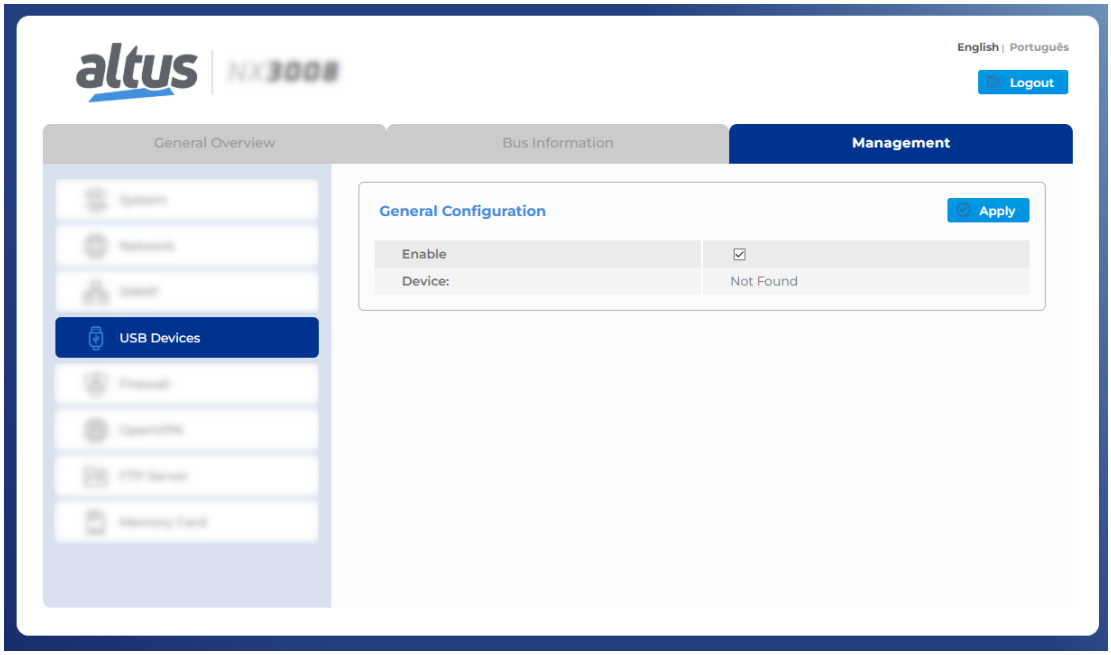


Figure 68: USB Devices Section

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. In the case where USB is disabled, the content is the presented in the figure, 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:

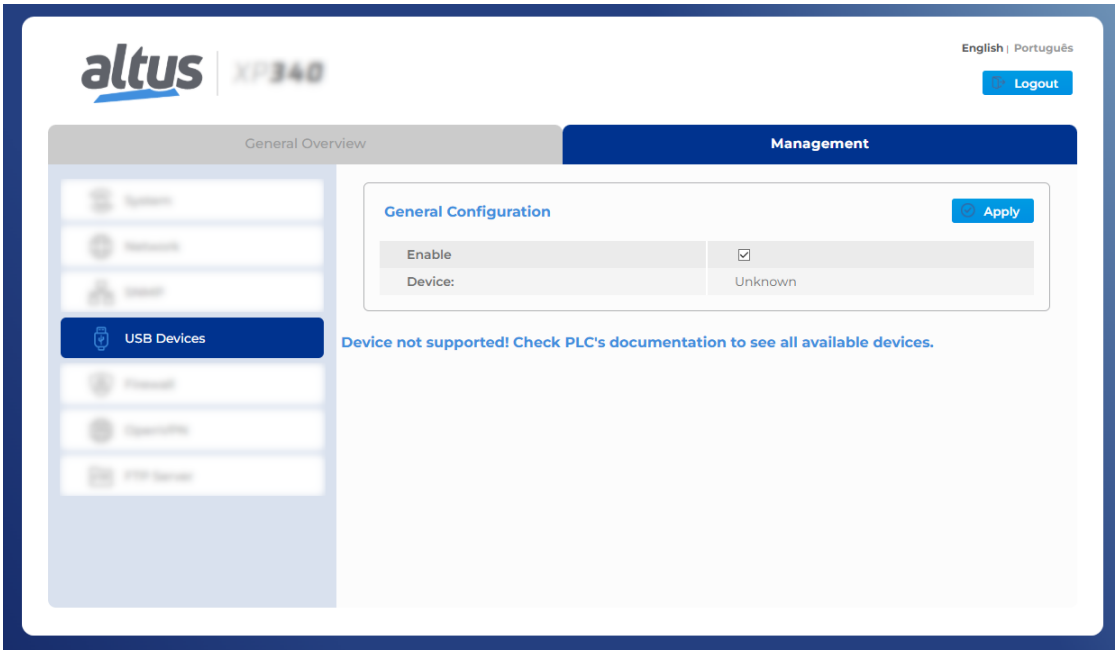


Figure 69: USB Devices - Unknown

5.8.1. Mass Storage Devices

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 tab *PLC Management* of the controller's diagnostics webpage as shown below:

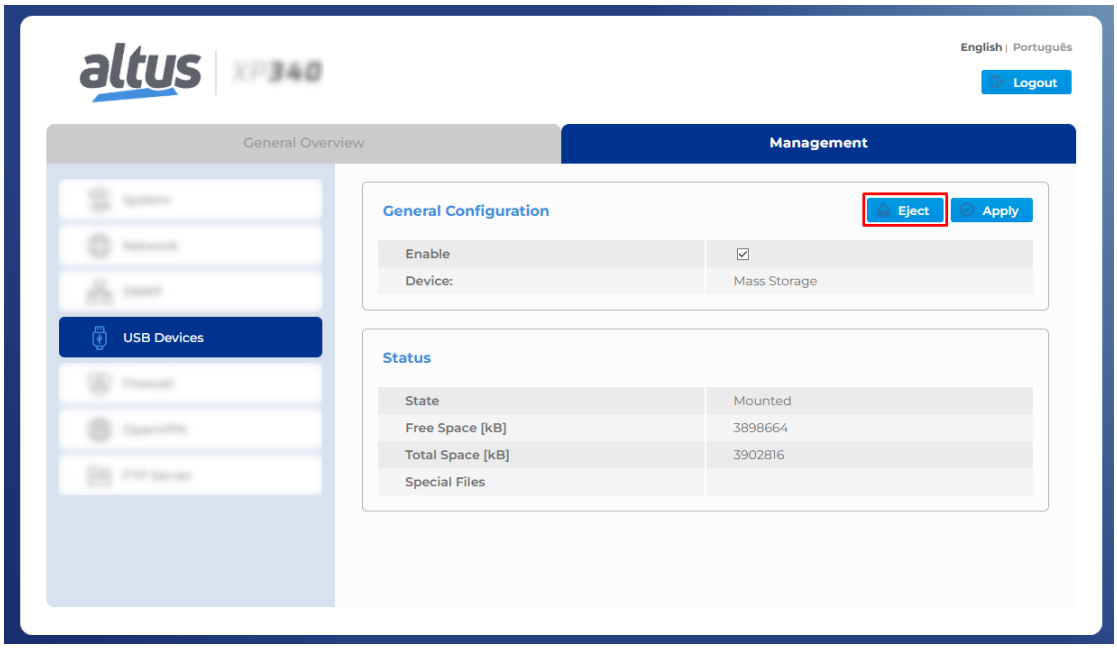


Figure 70: Mass Storage Device Information

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:

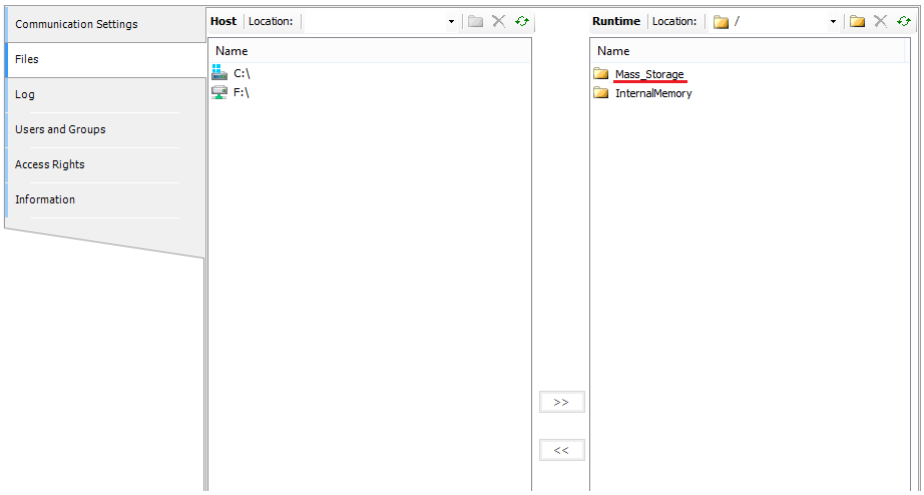


Figure 71: USB Mass Storage Folder

5.8.1.2. Not Loading the Application at 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 webpage as shown below.

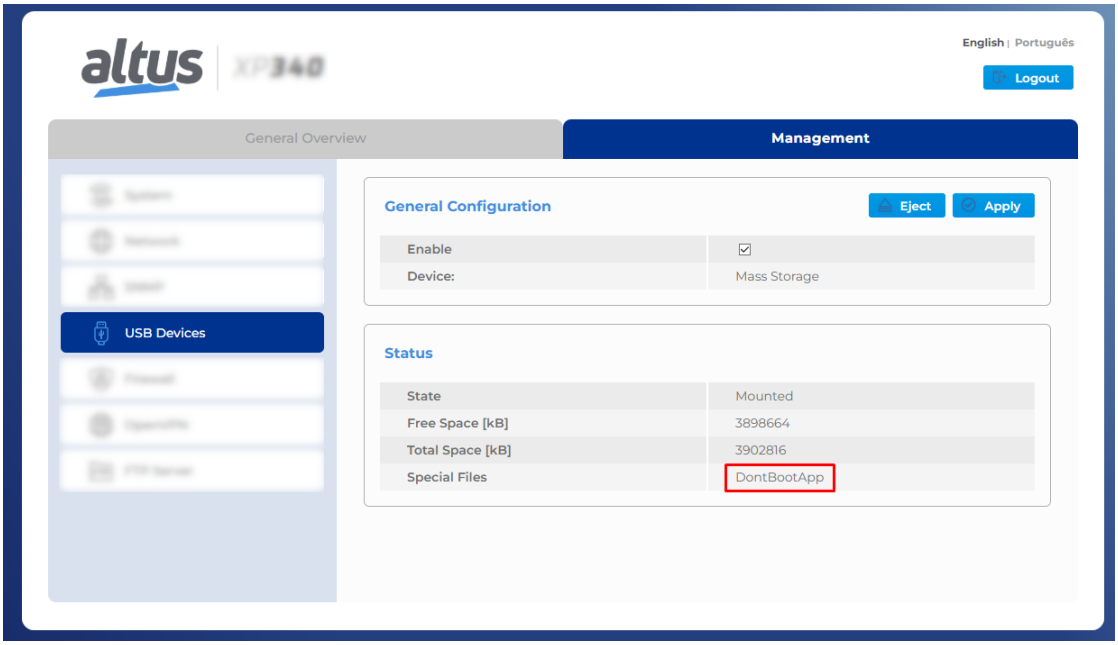


Figure 72: DontBootApp in the Mass Storage Device

5.8.1.3. Transfering an Application from the USB device

The USB mass storage device can also be used to transfer an application to the controller. To do that, place the two files *Application.app* and *Application.crc* on the root folder of 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 IEC XE 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 webpage as shown below:

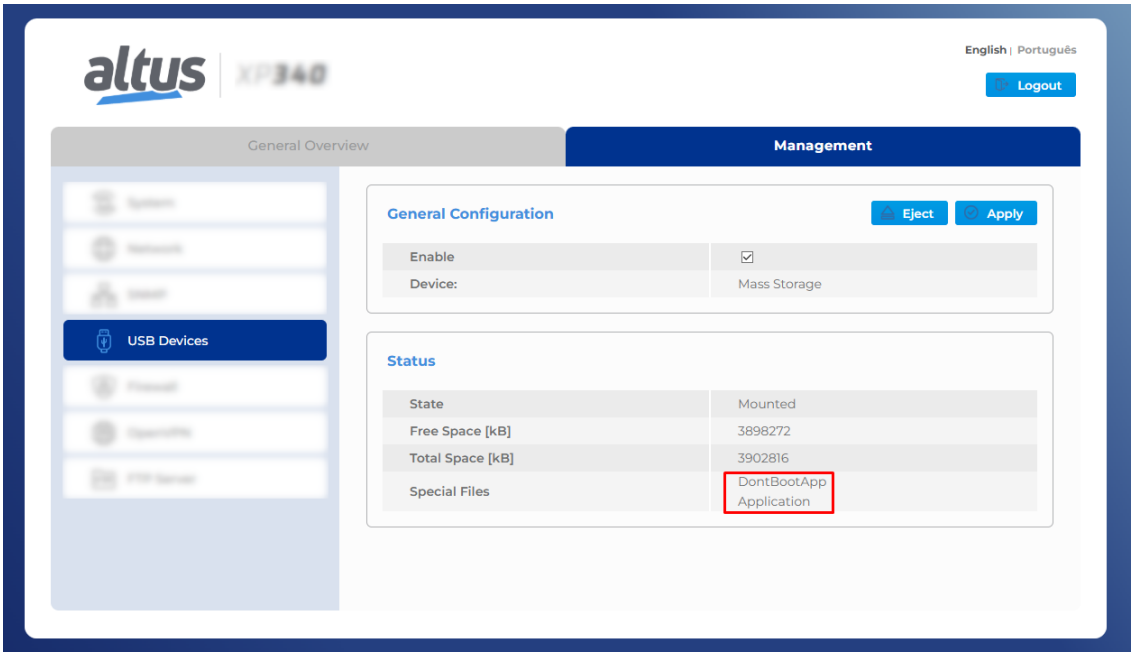


Figure 73: Application in the Mass Storage Device

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 Converters

Nexto Xpress 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 70: 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, the USB LED may turn on indicating that the device was properly detected and mounted and the device information will appear on section *USB Devices* located at the tab *PLC Management* of the controller's System Web Page as shown below:

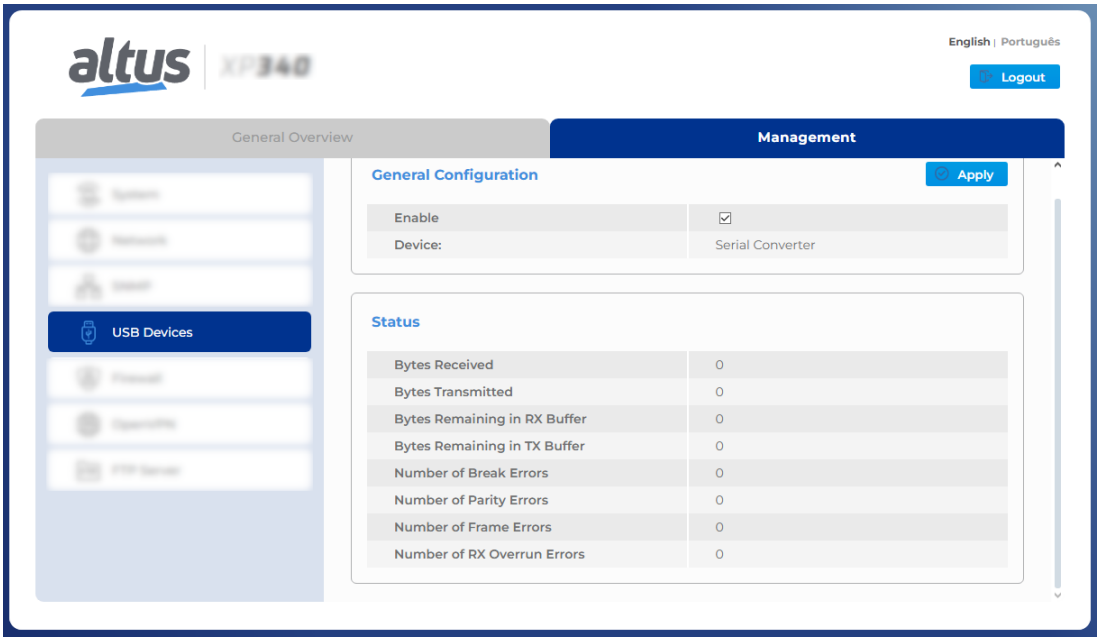


Figure 74: USB Devices - Serial Converter

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. Modem Devices

An USB Modem with a SIM chip can be used to connect the PLC to the internet using the cellular data network (telephone services, like sending SMS, are not implemented). This feature allows to use Nexto Xpress controllers to implement telemetry and IoT applications. There are basically two types modems: bridge and router. The following table shows the list of supported devices.

Model	Manufacturer	Type	Remarks
E303	Huawei	Bridge	-
E3272	Huawei	Bridge	-
E3276	Huawei	Bridge	-
E8372	Huawei	Router	Redirection of the configuration web page (button *Open Modem Configuration*) is not supported for this model. In this case, the modem configuration must be done externally by plugging it directly on a PC.

Table 71: Supported USB modems

The bridge modem is a non-managed device that implements a direct connection (pass through) to the mobile data network, so all the connection requests coming from the internet will reach the controller's operating system. For this kind of device, the configuration is performed through the controller's system web page as described further on this section.

On the other hand, the router modem is a managed device that implements a firewall with configurable network policies. For this kind of device, all the configuration is performed through a proprietary web page that is embedded into the device. By default, the Modem blocks any kind of inbound connections (i.e, will not allow the remote access). To allow this, the user must configure the modem through the device embedded web page to open the TCP port related to the desired service (some manufacturers calls this feature as *Virtual Server* or *Port Forwarding*). The following table describes the number of the TCP port associated to the main services of the PLC.

Port	Service
80	Embedded system web page
8080	Embedded Webvisu web page
1217	MasterTool programming

Table 72: TCP ports of main PLC services

The management of the USB Modem functionality is done through the *USB Devices* page in the *PLC Management* tab of the controller's system webpage. Once the Modem device is properly detected and mounted, the USB LED will turn on and the device information will appear as shown below:

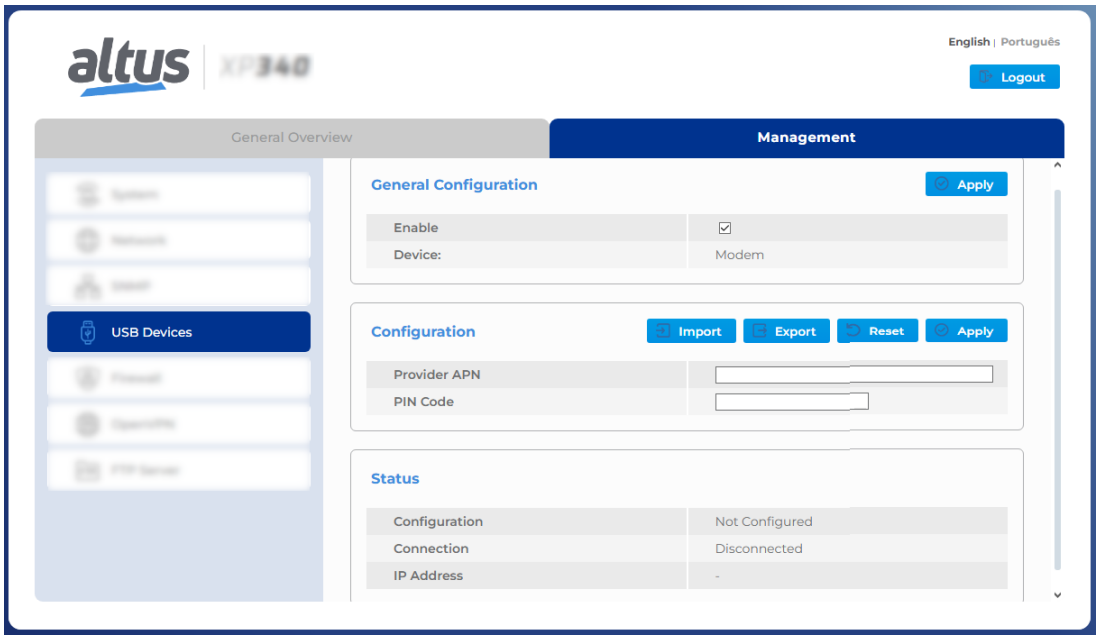


Figure 75: USB Modem Page Configuration

This area contains basically two sections: *Status* and *Configuration*.

The *Status* section is where all diagnostics related to the USB modem are displayed: configuration state (which does not depends on the device state), internet connection state and the modem IP address. These fields are updated automatically every time that a value is updated (does not require to reload the page manually). The same information is also provided on the symbolic variables diagnostics structure (see Section Diagnostics via Variables), which also contains the detailed description of the possible values for each field.

The *Configuration* section is where the user performs the modem configuration. For bridge modems, this section is used to enter the SIM chip information. For router modems, this section will show only a button called *Open Modem Configuration* that will redirect to the internal modem web page (check on the list of compatible devices if the redirection is supported, otherwise the modem must be configured externally on a PC). This section contains the following buttons:

- **Import:** loads the configuration from an external file
- **Export:** saves the current configuration on an external file
- **Reset:** erases the modem configuration from controller's memory
- **Apply:** write the configuration on the controller's memory

ATTENTION

The provider APN and PIN code fields are mandatory for every SIM chip. If the provider informs these parameters, they shall be used. In other hand, it's known that several SIM chips simply doesn't care for the content of these fields, using internal predefined values. In this case, these fields of the web page can be left with the default values and the connection will proceed successfully. Values like "zero" for PIN Code and "empty" for the Provider APN are not allowed.

The operation principle of USB Modem functionality is rather simple. For bridge devices, once the device is properly detected and the SIM parameters configured, the controller starts a background process, which continuously controls the modem to keep it connected to the internet. This eliminates the need of any kind of manual intervention, so if the connection is dropped for some reason (bad reception, carrier outage, etc.), the controller will try to reconnect automatically. The status of this process can be observed by the Connection *Status* field. For router modems, the device contains a similar process that runs internally (independent from the controller) that is called *Auto Connect*. It generally comes enabled by default, but can be disabled through the modem web page.

One important aspect to consider is that, if the USB Modem is configured, the controller system will set it as the default gateway for all Ethernet communication. It means that, if the controller is simultaneously connected to a local network (NET1), which has also access to the internet, all the Ethernet messages addressed to external IPs will route through the USB Modem (and not through NET1). NET1 returns as the default gateway when the USB Modem was removed.

The configuration and operation sequence can be summarized into the following steps:

- Plug the device into the USB port. After some seconds, the *USB Devices* page will show the *Device* as "Modem". If not, the device might be unsupported or defective.
- (For bridge devices) Set the SIM parameters (*Provider APN* - Access Point Name and the chip *PIN code*) as informed by the SIM chip provider. After clicking on *Apply* button, the connection process will start. Once configured, it is not necessary to set this information anymore. It will be saved on the controller's memory.
- Watch the connection status on the corresponding field. If everything goes fine, after some seconds it will inform that the modem is connected to the internet, and an IP address will appear in its field.

Once connected to the internet, this communication channel can be used for several purposes. One typical use case is to implement telemetry solution using MQTT Client Function Block to publish data. Another use case is to access the controller remotely. In this case, it is necessary to know the modem IP address on the internet. However, this address is dynamic and changes on every connection process. One way to workaround this problem is to publish the IP address (available on the modem diagnostic variables) through MQTT.

With the Modem IP address, it is possible to perform remote access to the controller's system web page to view status and diagnostics (firmware update is not supported). Also, it allows to program the controller remotely using MasterTool IEC XE. For this, the gateway must be configured with the Modem IP address like shown on the following picture:

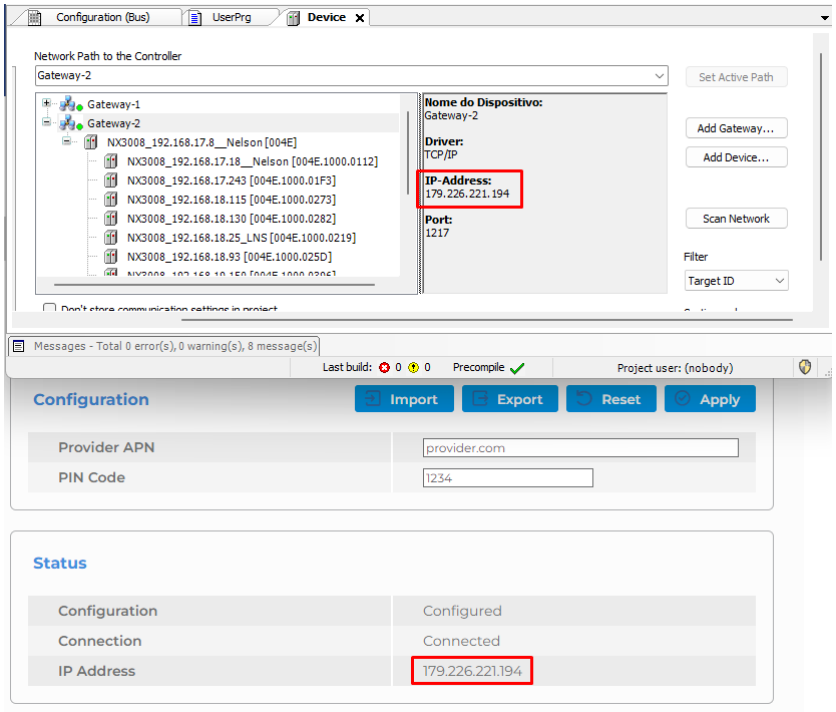


Figure 76: Configuring Gateway with Modem IP Address

ATTENTION

For bridge devices, or router devices with external access enabled (port forwarding), once connected to the internet, anyone who knows the modem IP address will be able to access the controller remotely. So, for security reasons, it is EXTREMELY important and recommended to configure the User Rights on the controller to restrict the online operations of MasterTool IEC XE with login and password.

5.8.4. WiFi Adapters

An USB WiFi adapter can be used to connect the PLC to an existing WiFi network, creating a second network adapter that can be used for programming and communication. The following table shows the list of supported chipsets.

Chipset	Manufacturer	Example of comercial products
RTL8188EU	Realtek	TP-LINK model TL-WN725N LM Technologies model LM007
RT28xx	Ralink/Mediatek	D-Link model DWA-125
AR9271	Atheros/Qualcomm	TP-LINK model TL-WN721N

Table 73: Supported chipsets for USB WiFi adapters

The management of the WiFi Adapter functionality is done through the *USB Devices* page in the *PLC Management* tab of the controller's system webpage. Once the WiFi adapter device is properly detected and mounted, the USB LED will turn on and the device information will appear as shown below:

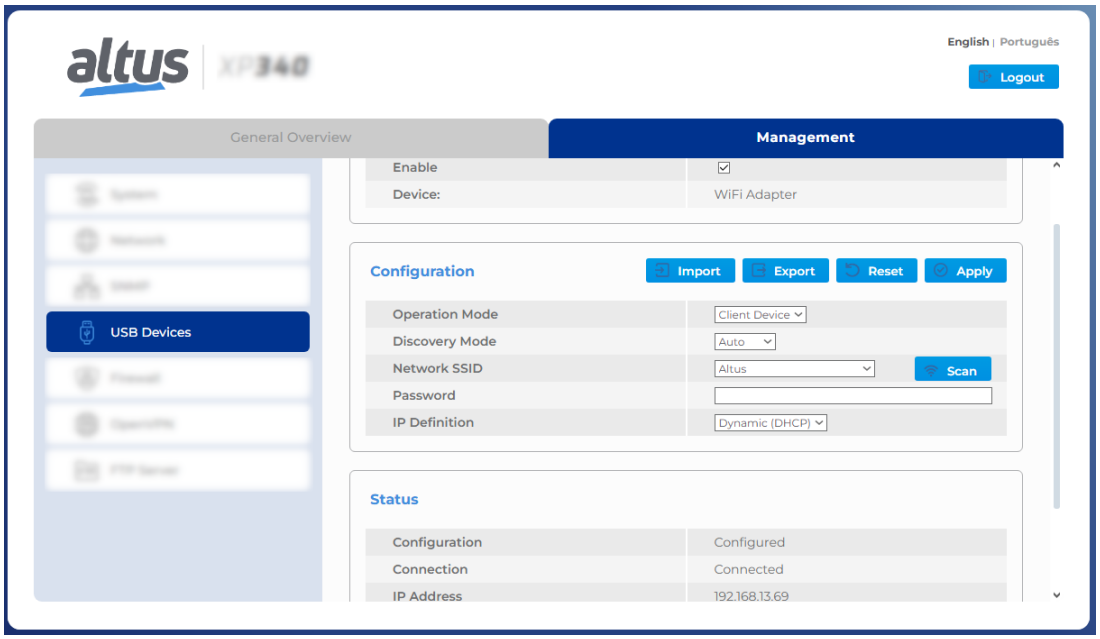


Figure 77: USB WiFi Adapter Page

This page contains basically two sections: *Status* and *Configuration*.

The *Status* section shows all diagnostics related to the WiFi adapter: configuration state, connection state, IP address, Netmask, Gateway and MAC address. These fields are updated as soon the values change. These informations are also provided in the symbolic variables diagnostics structure (see Section Diagnostics via Variables).

The *Configuration* section is composed by the following parameters:

- **Operation Mode:** defines how the WiFi adapter will operate (currently, only client mode is supported).

- **Discovery Mode:** defines what is the method to set the WiFi network. If selected as "Auto", the "Scan" button must be used to choose the wanted network. If selected as "Manual", the name of the SSID and Security Type must be entered manually.
- **Network SSID:** when discovery mode is set to "Auto", this field will show the available networks found on the scan process sorted from the best to the worst signal level (up to down). When discovery mode is set to "Manual" this field must be populated with the SSID of the wanted network.
- **Security Type:** this field is only available when the discovery mode is set to "Manual" (the "Auto" mode automatically selects the security type provided by the scanned network). This field defines the type of security used in the WiFi network, which can be "Public" or "WPA2-Personal".
- **Password:** this where you need to enter with the WiFi network password. The field will be automatically blocked if the Security Type is set as "Public" or the scanned network chosen does not use a security protocol.
- **IP Definition:** defines if the WiFi adapter will set the IP address dynamically (assigned by the network DHCP Server) or statically (where the user needs to enter the IP settings manually).
- **IP Address, Network Mask and Gateway:** only available when the IP definition is set as "Static". These fields will be used to configure the network parameters of the WiFi adapter.
- **Default Gateway:** this field defines what network interface will be used as a Gateway to access the Internet. It is possible to choose the "WiFi Adapter" or the "Local Ethernet" for this function.

ATTENTION

For proper operation, the WiFi adapter network (defined by IP and Mask) must be different from the one configured for NET1.

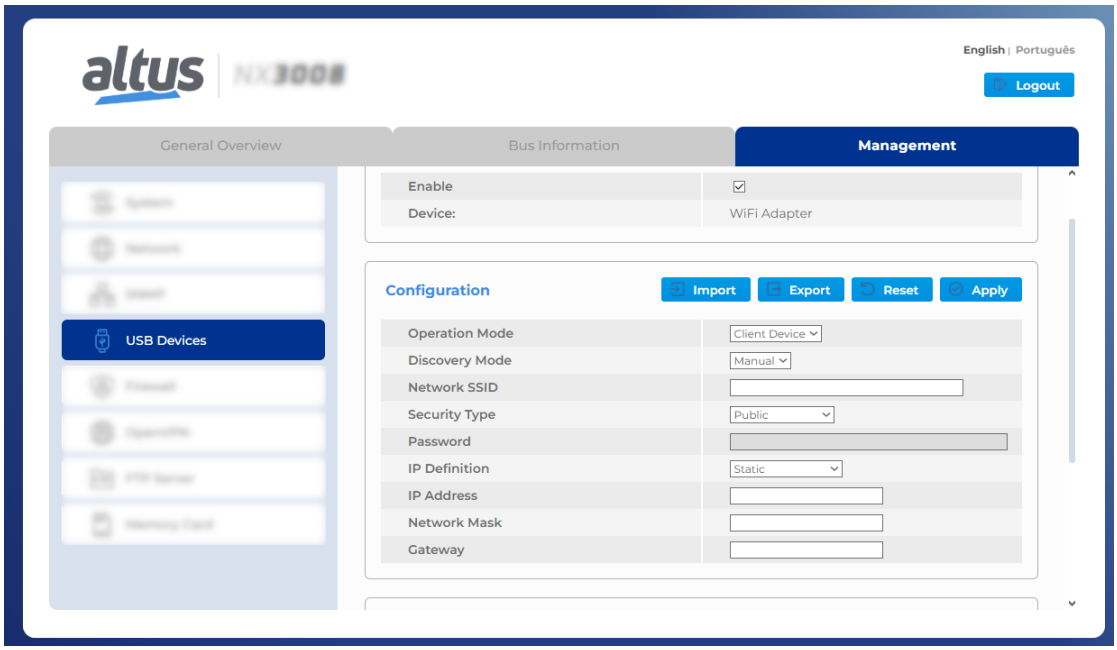


Figure 78: USB WiFi Adapter Configuration

Besides that, the *Configuration* section contains the following buttons:

- **Import:** loads a configuration file
- **Export:** download a file with the current configuration
- **Reset:** returns the configuration to the default
- **Apply:** apply the current configuration

Once the device is properly detected and the network parameters are configured, the controller will always try to keep connected to the WiFi network. The status of this process can be observed by the Connection Status field.

ATTENTION

If the *Default Gateway* was set as "*WiFi Adapter*", the Gateway of the NET1 diagnostics of the MasterTool will show zero (0.0.0.0). Otherwise, if it was set as "*Local Ethernet*", the WiFi adapter Gateway will be zero.

The configuration and operation sequence can be summarized into the following steps:

- Plug the device into the USB port. After some seconds, the *USB Devices* page will show the *Device* as "WiFi Adapter". If not, the device might be unsupported or defective.
- Set the network configuration. After clicking on Apply button, the connection process will start. Once configured, it is not necessary to set this information anymore. It will be saved on the controller's memory.
- Watch the connection status on the corresponding field. If everything goes fine, after some seconds it will inform that the adapter is connected to the WiFi network.

The following picture shows the page of a controller connected to a WiFi network:

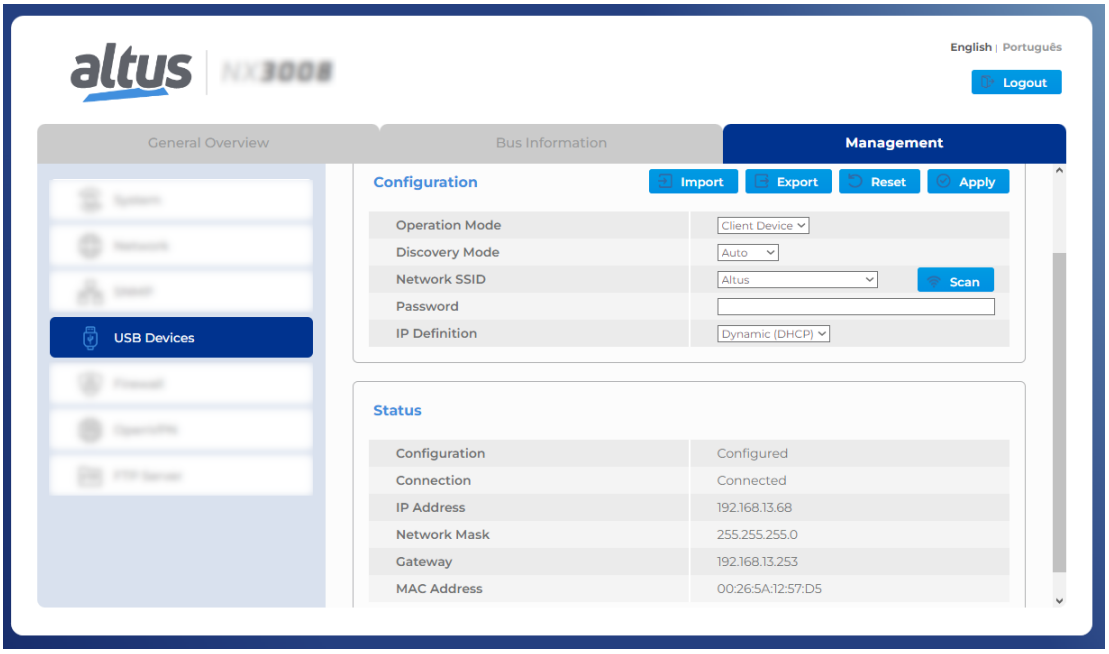


Figure 79: USB WiFi Adapter Connected to a Network

Once connected to the WiFi network, this communication channel can be used for several purposes. To program the PLC with MasterTool IEC XE, the gateway must be configured with the IP address assigned to the WiFi adapter (similar to the USB Modem, Figure 76). This IP address can also be used to access the controller's system web page, where it is possible to perform a firmware update, which is not available when using the USB Modem. Additionally, this communication channel can also be used with the MQTT client Function Block to report data to an external broker on the internet (in this case, the WiFi adapter as Default Gateway).

5.8.5. Ethernet Adapters Configuration

The Nexto Xpress series of controllers allows implementing a second Ethernet port using an USB Adapter to Ethernet. The following table shows the list of supported devices.

Adapter	Manufacturer
USB 3.0 to Gigabit SuperSpeed Ethernet Adapter UE300	TP-LINK
USB31000S USB 3.0 to Gigabit Ethernet Adapter	STAR TECH

Table 74: Supported USB to Ethernet adapters

The management of the Ethernet Adapter functionality is done through the *USB Devices* page in the *Management* tab of the controller's system web page. Once the USB Ethernet adapter device is properly detected and mounted, the USB LED will turn on and the device information will appear as shown below:

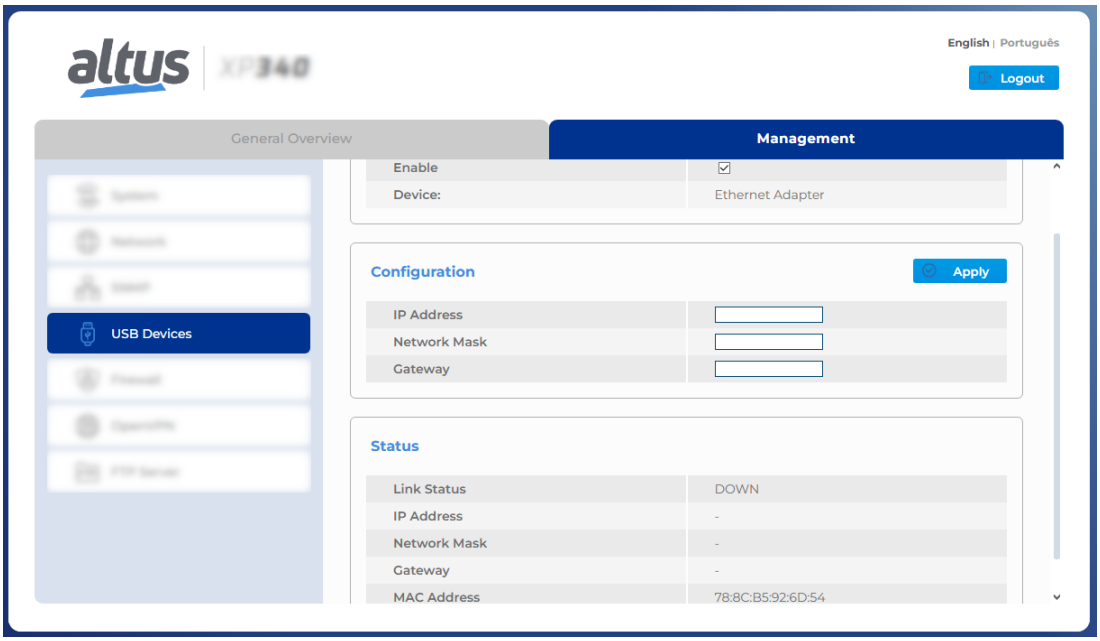


Figure 80: USB Ethernet Adapter Page without Configuration

The USB Ethernet adapter's page contains basically two sections: *Status* and *Configuration*.

The *Status* section shows all diagnostics related to the Ethernet adapter: Link state, IP address, Netmask, Gateway and MAC address. These fields are automatically updated based on the current state of the device. These informations are also provided in the symbolic variables diagnostics structure (see Section Diagnostics via Variables).

The *Configuration* section is composed by the following parameters: IP address, Netmask and Gateway. These fields will be used to configure the Ethernet adapter's network parameters.

ATTENTION

For proper operation, the Ethernet adapter network (defined by IP and Mask) must be different from the one configured for NET1.

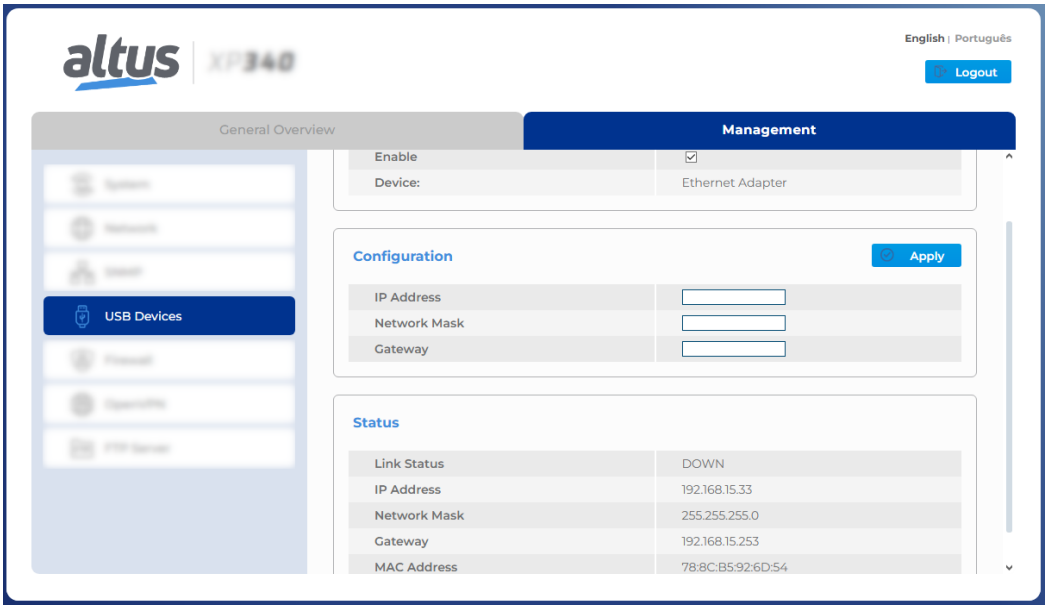


Figure 81: USB Ethernet Adapter Page with Configuration Applied

Besides that, the *Configuration* section contains the *Apply* button for that the IP, Netmask and Gateway settings are definitely applied. After the device is correctly configured with compatible network parameters, whenever the adapter is connected, the last configuration performed will be applied, regardless of whether it is a different adapter than the last configured one.

ATTENTION

The USB Ethernet adapter becomes the Default Gateway when configured.

The configuration and operation sequence can be summarized into the following steps:

- Plug the device into the USB port. After some seconds, the *USB Devices* page will show the *Device* as "Ethernet Adapter". If not, the device might be unsupported or defective.
- Set the network configuration. After filling in the fields, use the *Apply* button to carry out the settings. Once configured, it is not necessary to set this information anymore. It will be saved on the controller's memory.
- Watch the status of the network settings on the corresponding field. If everything goes fine, after some seconds it will be showed the parameters applied in the last item. After this, just it is necessary to connect an Ethernet cable to establish a connection with the network (when the cable was connected, the Link state must go to *UP*).

The following picture shows the page of a controller with an USB Ethernet Adapter configured and connected to a network:

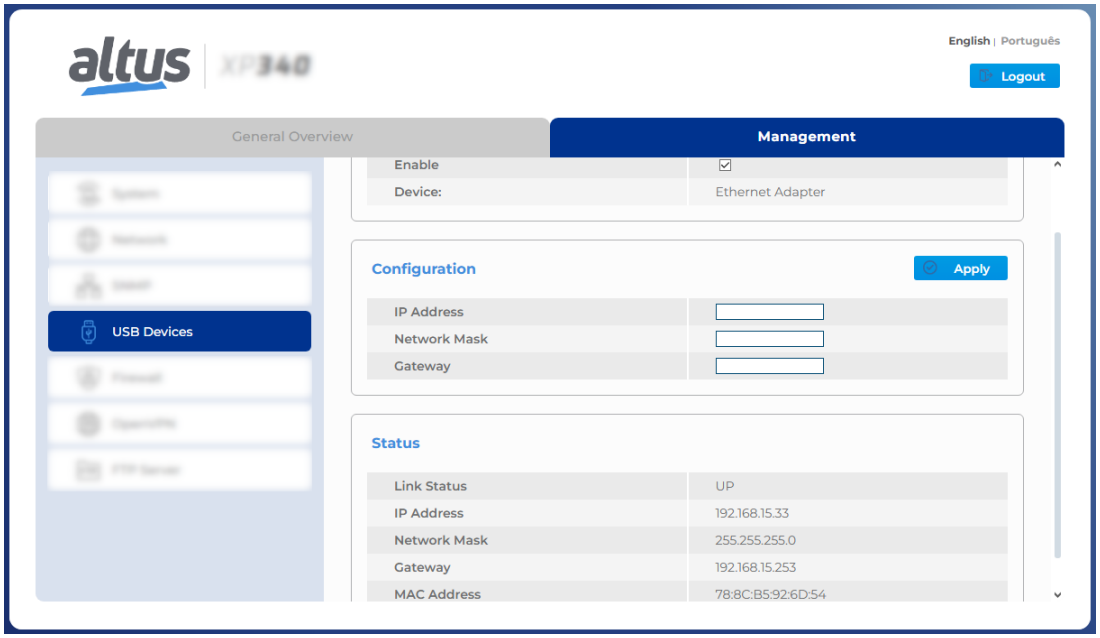


Figure 82: USB Ethernet Adapter Page with Configuration Applied and Link UP

The USB Ethernet adapter allows the controller expands its network interfaces, creating the possibility of dedicating the NET1 to run a communication protocol, for example, the protocol to SoftMotion. Therefore, the adapter can be used for other purposes, such as programming the PLC with the MasterTool IEC XE or using the IP address to access the web page of the controller system, where it is also possible to perform a firmware update.

5.8.6. 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*.

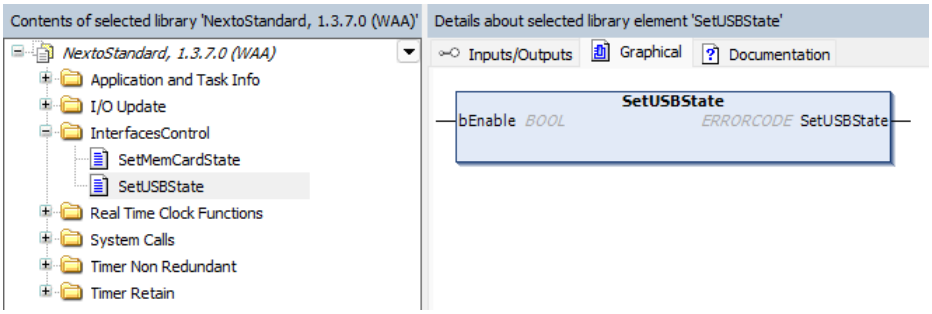


Figure 83: SetUSBState information in Library Manager

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:

	XP300	XP315	XP325	XP340	XP350	XP351
Mapped Points	20480					
Mappings (Per Instance / Total)	5120 / 20480					
Requests	512					
NETs – Client or Server instances	4					
COM (n) – Master or Slave instances	1					
Control Centers	-	-	-	3	-	-

Table 75: 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.

Control Centers: *Control Center* is all client device connected to the CPU through protocol IEC 60870-5-104. This field informs the maximum of client devices of control center type supported by the CPU. Correspond to the sum of all client devices of communication protocol Server IEC 60870-5-104 (does not include master or clients from MODBUS RTU Slave and MODBUS Server protocols)

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 76: 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.

The limitations of the IEC 60870-5-104 Server protocol can be seen in the table below.

Limitations	IEC 60870-5-104 Server
Devices per instance	3
Simultaneous requests per instance	3
Simultaneous requests per device	1

Table 77: Protocol IEC 60870-5-104 Server Limits

Notes:

Devices per instance: Quantity of client devices, of type control center, supported for each IEC 60870-5-104 Server protocol instance. The limit informed might be smaller because of the CPU total limits (check Table 75).

Simultaneous requests per instance: Quantity of requests that can be received simultaneously by each instance of Server protocol.

Simultaneous requests per device: Quantity of requests that can be received simultaneously of each IEC 60870-5-104 Client device.

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	Execution	After a breakpoint in MainPrg
MODBUS Symbol	Slave/Server	✓	✓	✓	✓	✓
	Master/Client	✗	✗	✗	✓	✓
MODBUS	Slave/Server	✗	✗	✗	✓	✗
	Master/Client	✗	✗	✗	✓	✓
SOE (DNP3)	Outstation	✓	✓	✓	✓	✓
IEC 60870-5-104	Server	✗	✗	✗	✓	✓
EtherCAT	Master	✓	✓	✗	✓	✓
OPC DA	Server	✓	✓	✓	✓	✓
OPC UA	Server	✓	✓	✓	✓	✓
SNTP	Client	✓	✓	✓	✓	✓
HTTP	Server	✓	✓	✓	✓	✓
SNMP	Agent	✓	✓	✓	✓	✓
EtherNet/IP	Scanner	✓	✓	✗	✓	✗
	Adapter	✗	✓	✗	✓	✗

Table 78: Protocol Behavior x CPU State

Notes:

Symbol ✓: Protocol remains active and operating normally.

Symbol ✗: Protocol is disabled.

EtherCAT: The tests were performed using the default value defined in *PLC Settings* (see table PLC Settings), with option *Update I/O during stop* checked and option *Configure all outputs to default*.

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. Double Points

The input and output double digital points representation is done through a special data type called DBP (defined in the *LibDataTypes* library). This type consist basically in a structure of two BOOL type elements, called OFF and ON (equivalent to TRIP and CLOSE respectively).

In Nexto, variables of this type cannot be associated to digital input and output modules, being necessary the single digital points mapping, BOOL type, and the treatment by application to conversion in double points.

To further information about the double points mapping in the input and output digital modules check the IEC 60870-5-104 Server section.

5.9.3. CPU's Events Queue

The CPU owns an events queue of type FIFO (First In, First Out) used to store temporarily the events related to communication points until they are moved to their final destiny.

All communication points events generated in the CPU are directed and stored in CPU's queue. This queue has the following features:

- Size: 1000 events
- Retentivity : it is not retentive
- Overflow policy: keep the newests

ATTENTION

In the Nexto PLC, the events queue is stored in a non-retentive memory area (volatile). This way, the events present in CPU's queue, which haven't been transmitted yet to the control center, are going to be lost in a CPU's eventual power off.

The CPU's event queue is redundant, that means it is synchronized each cycle between both CPUs, when is used CPU's redundancy. Further information can be found on the section about CPU redundancy.

The in and out of events in this queue follows the concept of producer/consumer. Producers are those system elements capable of generate events, adding events in the CPU's queue, while the consumers are those system elements which receive and use this events, taking them of the CPU's queue. The figure below describes this working, including the example of some events consumers and producers.

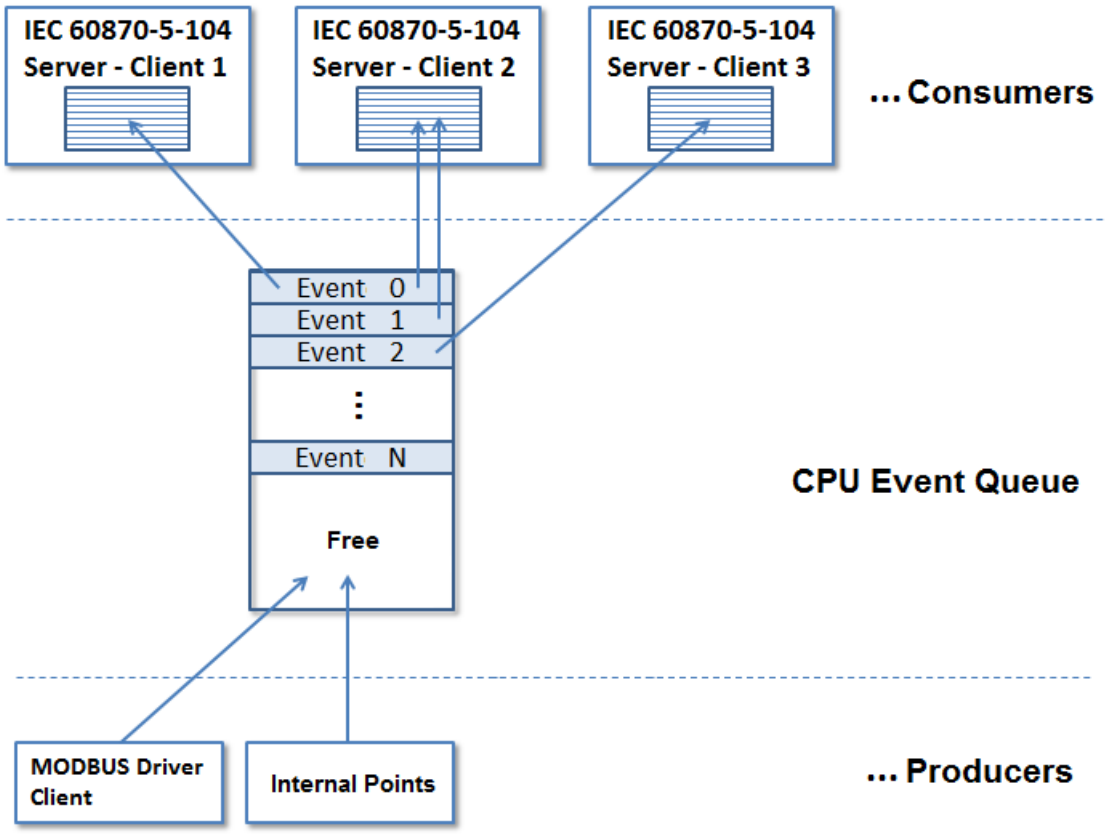


Figure 84: CPU's Event Queue

5.9.3.1. Consumers

The consumers are typically communication drivers that will communicate with SCADA or HMI. After been stored in CPU's queue, the consumers receive the events related to communication points mapped in its configuration. These events are then stored in a consumer's own events queue, which the size and working are described on the communication driver specific section.

5.9.3.2. Queue Functioning Principles

Once stored in CPU's queue, each event is transmitted to the consumer that has this communication point in its data base. On the figure above, the Event 0 is referred to a communication point mapped to two control centers IEC 60870-5-104 (Client 1 and 2). Thus the Event 1 is referred to a communication point mapped only to one control center IEC 60870-5-104 (Client 2). By its time, the Event 2 is referred to a communication point mapped to another control center IEC 60870-5-104 (Client 3).

The events remain stored in the CPU's queue until all its consumers acknowledge its receiving. The criteria used to confirm the receive is specific of each consumer. In case of the IEC 60870-5-104 Server, the acknowledge occurs when the event is transmitted to the IEC 60870-5-104 client.

In Nexto Series case, there are no diagnostics available to watch the CPU's events queue occupation, not even information about the queue overflow. However the consumers have a diagnostics group referred to its events queue. Further information can be found at the specific driver communication section.

5.9.3.2.1. Overflow Sign

The overflow sign to the consumers' events queue occurs in two situations:

- When the consumer events queue is out of space to store new events
- If the CPU aborted the event generation (because occurred to more events in a single execution cycle than the events queue total size)

5.9.3.3. Producers

The producers are typically communication drivers or PLC internal elements that are capable to generate events. The previous figure show some examples.

- **Internal Points:** This is a PLC's firmware internal element, which detects events each execution cycle (MainTask) to those communication points that don't have a defined origin and then inserts the events in the CPU's queue. The maximum number of events that can be detected in each MainTask cycle is equal to the CPU's events queue size. In case the number of generated events is bigger than this, in a single cycle, the exceeding are going to be lost.
- **MODBUS Driver (Client/Server/Master/Slave):** The variables value change caused by MODBUS read/write are detected at each MainTask cycle and then the events are inserted in CPU's queue. In Client/Master cases, are also generated quality events when there is a communication failure with the slave device.

5.9.4. Interception of Commands Coming from the Control Center

The Nexto PLC has a function block that allows selection commands and operation to the output points received by server drivers (IEC 60870-5-104 Server) been treated by the user logic. This resource allows the interlocking implementation, as well as the handling of the received command data in the user logic, or yet the command redirecting to different IEDs.

The commands interception is implemented by the *CommandReceiver* function block, defined in the *LibRtuStandard*. The input and output parameters are described on the following tables:

Parameter	Type	Description
bExec	BOOL	When TRUE, executes the command interception
bDone	BOOL	Indicates that the command output data have been already processed, releasing the function block to receive another command
dwVariableAddr	DWORD	Variable address, mapped in the server driver, which will receive the client command
eCommandResult	ENUM	Input action defined by user from the following list: SUCCESS(0) NOT_SUPPORTED(1) BLOCKED_BY_SWITCHING_HIERARCHY(2) SELECT_FAILED(3) INVALID_POSITION(4) POSITION_REACHED(5) PARAMETER_CHANGE_IN_EXECUTION(6) STEP_LIMIT(7) BLOCKED_BY_MODE(8) BLOCKED_BY_PROCESS(9) BLOCKED_BY_INTERLOCKING(10) BLOCKED_BY_SYNNCHROCHECK(11) COMMAND_ALREADY_IN_EXECUTION(12) BLOCKED_BY_HEALTH (13) ONE_OF_N_CONTROL(14) ABORTION_BY_CANCEL(15) TIME_LIMIT_OVER(16) ABORTION_BY_TRIP(17) OBJECT_NOT_SELECTED(18) OBJECT_ALREADY_SELECTED(19) NO_ACCESS_AUTHORITY(20) ENDED_WITH_OVERSHOOT(21) ABORTION_DUE_TO_DEVIATION(22) ABORTION_BY_COMMUNICATION_LOSS(23) BLOCKED_BY_COMAND(24) NONE(25) INCONSISTENT_PARAMETERS(26) LOCKED_BY_OTHER_CLIENT(27) HARDWARE_ERROR(28) UNKNOWN(29)
dwTimeout	DWORD	Time-out [ms] to the treatment by user logic

Table 79: CommandReceiver Function Block Input Parameters

Notes:

bExec: When FALSE, the command just stop being intercepted for the user application, but it keeps being treated normally by the server.

bDone: After the command interception, the user is going to be responsible for treat it. At the end of the treatment, this input must be enabled for a new command can be received. Case this input is not enabled, the block is going to wait the time defined in *dwTimeout*, to then become capable of intercept new commands.

eCommandResult: Treatment results of command intercepted by user. The result returned to the client that sent the command, which must be attributed together with the input *bDone*, is converted to the protocol's format from which was received the command. In Nexto Series it is only supported the interception of commands coming from protocol IEC 60870-5-104. In protocol interception, any return different from *SUCCESS* results in a negative Acknowledge.

ATTENTION

It is not recommended the simultaneous commands interception to one same variable by two or more *CommandReceiver* function blocks. Just one of the function blocks will intercept correctly the command, being able to suffer undesirable interference from the others function blocks if addressed to the same variable.

Parameter	Type	Description
bCommandAvailable	BOOL	Indicates that a command was intercepted and the data are available to be processed
sCommand	STRUCT	This structure stores received command data, which is composed by the following fields: eCommand sSelectParameters sOperateParameters The description of each field is in this section.
eStatus	ENUM (TYPE_RESULT)	Out of function action from obtained result, according to list: OK_SUCCESS(0) ERROR_FAILED(1)

Table 80: CommandReceiver Function Block Output Parameters

Note:

eStatus: Return of a register process of a communication point command interception. When the interception is registered with success *OK_SUCCESS* is returned, else *ERROR_FAILED* is. In case interceptor register failure, commands to the determined point are not intercepted by this function block. *TYPE_RESULT* is defined in *LibDataTypes* library.

Supported commands are described on table below:

Parameter	Type	Description
eCommand	ENUM	NO_COMMAND(0) SELECT(1) OPERATE(2)

Table 81: CommandReceiver Function Block Supported Commands

The parameters that build the *sSelectParameters*, *sOperateParameters* and *sCancelParameters* structures are described on the following table:

Parameter	Type	Description
sSelectConfig	STRUCT	Received selection command configuration. This structure parameters are described on Table 83
sValue	STRUCT	Received value in a select, when is received a selection command with value. This structure parameters are described on Table 86

Table 82: Parameters sSelectParameters

Parameter	Type	Description
bSelectWithValue	BOOL	When true indicates a selection command reception with value.

Table 83: Parameters sSelectConfig

Parameter	Type	Description
sOperateConfig	STRUCT	Received selection command configuration. This structure parameters are described on Table 85
sValue	STRUCT	Field of received operation command referred value. This structure parameters are described on Table 86

Table 84: Parameters sOperateParameters

Parameter	Type	Description
bDirectOperate	BOOL	When true indicates that an operation command without select was received.
bNoAcknowledgement	BOOL	When true indicates that a command, which doesn't require the receiving acknowledge, was received.
bTimedOperate	BOOL	When true indicates that an operation command activated by time was received.
liOperateTime	LINT	Programming of the instant in which it must be runned the command. This field is valid only when *bTimedOperate* is true.
bTest	BOOL	When true indicates that the received command was sent only for test, as so the command must not be runned.

Table 85: Parameters sOperateConfig

Parameter	Type	Description
eParamType	ENUM	Informs the type of the received command: NO_COMMAND(0) SINGLE_POINT_COMMAND(1) DOUBLE_POINT_COMMAND(2) INTEGER_STATUS_COMMAND(3) ENUMERATED_STATUS_COMMAND(4) ANALOGUE_VALUE_COMMAND(5)
sSinglePoint	STRUCT	When a command is received, in received command type function, defined by eParamType, the corresponding data structure is filled. This structures parameters are described on Tables 87 to 91
sDoublePoint	STRUCT	
sIntegerStatus	STRUCT	
sEnumeratedStatus	STRUCT	
sAnalogueValue	STRUCT	

Table 86: Parameters sValue

Parameter	Type	Description
bValue	BOOL	Point operation value.
sPulseConfig	STRUCT	The pulsed command configuration parameters are stored in this structure. This structure parameters are described on Table 92.

Table 87: Parameters sSinglePoint

Parameter	Type	Description
bValue	BOOL	Point operation value.
sPulseConfig	STRUCT	The pulsed command configuration parameters are stored in this structure. This structure parameters are described on Table 92.

Table 88: Parameters sDoublePoint

Parameter	Type	Description
diValue	DINT	Point operation value.

Table 89: Parameters sIntegerStatus

Parameter	Type	Description
dwValue	DWORD	Point operation value.

Table 90: Parameters sEnumeratedStatus

Parameter	Type	Description
eType	ENUM	Informs the data type of the received analog value. INTEGER (0) FLOAT (1)
diValue	DINT	Point operation value, integer format.
fValue	REAL	Point operation value, float format.

Table 91: Parameters sAnalogueValue

Parameter	Type	Description
bPulseCommand	BOOL	When true indicates that received command is pulsed.
dwOnDuration	DWORD	This is time, in milliseconds, that the output must remain on.
dwOffDuration	DWORD	This is time, in milliseconds, that the output must remain off.
dwPulseCount	DWORD	Number of times the command must be executed.

Table 92: Parameters sPulseConfig

To intercept commands to a specific point, first it is need to load in the *dwVariableAddr* parameter the variable address correspondent to the point wanted to intercept the commands and then execute a pulse in the *bExec* parameter. Once the command

was intercepted, the function block informs that a command was intercepted through *bCommandAvailable* parameter. The intercepted command information are then filled in the *sCommand* and *eStatus* output parameters, according to the received command type. This operation depends only of the received command type, don't matter the variable's data type to which is being intercepted the command. The interception is finished and then the function block can be released to intercept a new command when *bDone* parameter is *true*. Yet must be pointed the command processing result in *eCommandResult*.

5.9.5. MODBUS RTU Master

This protocol is available for the Nexto Series CPUs in its serial channels. By selecting this option at MasterTool IEC XE, 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).

There are two configuration modes for this protocol. One makes use of Direct Representation (%Q), in which the variables are defined by its address. The other, called Symbolic Mapping has the variables defined by its name.

Regardless of the configuration mode, the steps to insert a protocol instance and configure the serial interface are the same. The procedure to insert a protocol instance is found in detail in the MasterTool IEC XE User Manual - MU299609 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 Configuration section.

5.9.5.1. 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 85.
- 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 86.
- Add and configure the MODBUS mappings on Mappings tab as Figure 87, 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 88, 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.

5.9.5.1.1. MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration

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

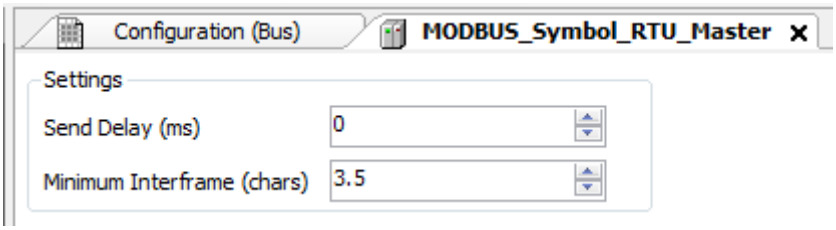


Figure 85: MODBUS RTU Master Configuration Screen

Configuration	Description	Default	Options
Send Delay (ms)	Delay for the answer transmission.	0	0 to 65535
Minimum Interframe (chars)	Minimum silence time between different frames.	3.5	3.5 to 100.0

Table 93: 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, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_RTU_MASTER_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words 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 1: invalid serial port 2: invalid serial port mode 3: invalid baud rate 4: invalid data bits 5: invalid parity 6: invalid stop bits 7: invalid modem signal parameter 8: invalid UART RX Threshold parameter 9: invalid time-out parameter 10: busy serial port 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid signal modem method 22: CTS time-out = true 23: CTS time-out = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from manual 33: invalid flow control for the configured serial port 34: data reception not allowed in normal mode 35: data reception not allowed in extended mode 36: DCD interruption not allowed 37: CTS interruption not allowed 38: DSR interruption not allowed 39: serial port not configured 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).

T_DIAG_MODBUS_RTU_MASTER_1.*	Size	Description
Communication Statistics:		
tStat.wTXRequests	WORD	Counter of request transmitted by the master (0 to 65535).
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 94: MODBUS RTU Master Diagnostics

Note:

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

5.9.5.1.2. Devices Configuration – Symbolic Mapping configuration

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

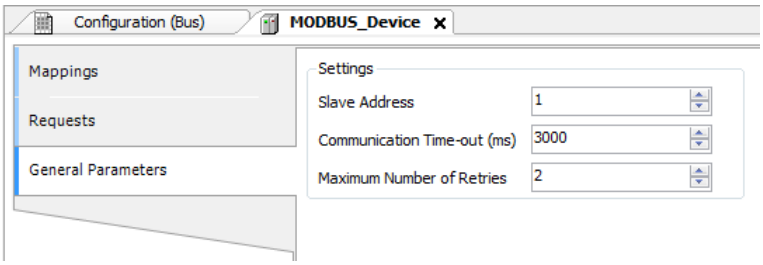


Figure 86: Device General Parameters Settings

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 95: 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.

5.9.5.1.3. Mappings Configuration – Symbolic Mapping Settings

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

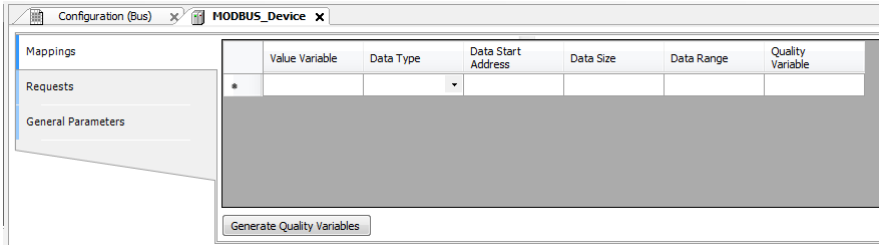


Figure 87: MODBUS Data Mappings Screen

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) Coil - Read (1 bit) Holding Register - Write (16 bits) Holding Register - Read (16 bits) Holding Register – Mask AND (16 bits) Holding Register – Mask OR (16 bits) Input Register (16 bits) 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 96: 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 97: 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 – Symbolic Mapping Settings

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

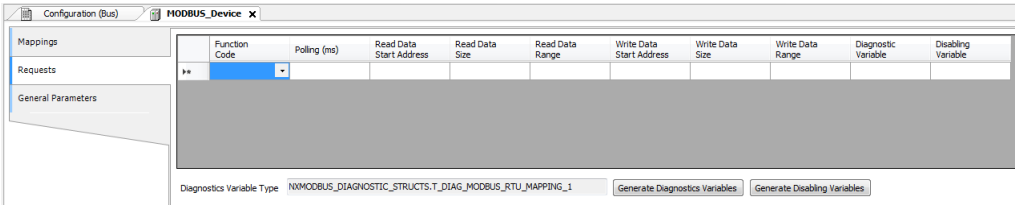


Figure 88: Data Requests Screen MODBUS Master

Configuration	Description	Default Value	Options
Function Code	MODBUS function type	-	01 – Read Coils 02 – Read Input Status 03 – Read Holding Registers 04 – Read Input Registers 05 – Write Single Coil 06 – Write Single Register 15 – Write Multiple Coils 16 – Write Multiple Registers 22 – Mask Write Register 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 98: 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 99: 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 or by direct representation, 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 and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 100.

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 118 for further details.
Last exception code received by master:		
eLastExceptionCode	MODBUS_EXCEPTION (BYTE)	NO_EXCEPTION (0) FUNCTION_NOT_SUPPORTED (1) MAPPING_NOT_FOUND (2) ILLEGAL_VALUE (3) ACCESS_DENIED (128)* MAPPING_DISABLED (129)* 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.

T_DIAG_MODBUS_RTU_MAPPING_1.*	Size	Description
wCommErrorCounter	WORD	Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero.

Table 100: 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 101: 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.

5.9.6. MODBUS RTU Slave

This protocol is available for the Nexto Series on its serial channels. At selecting this option in MasterTool IEC XE, 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 the MasterTool IEC XE User Manual - MU299609. 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 RTS/CTS signals behavior, the parity, the stop bits channel, among others. See Serial Interface Configuration section.

5.9.6.1. 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.

5.9.6.1.1. MODBUS Slave Protocol General Parameters – Configuration via Symbolic Mapping

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

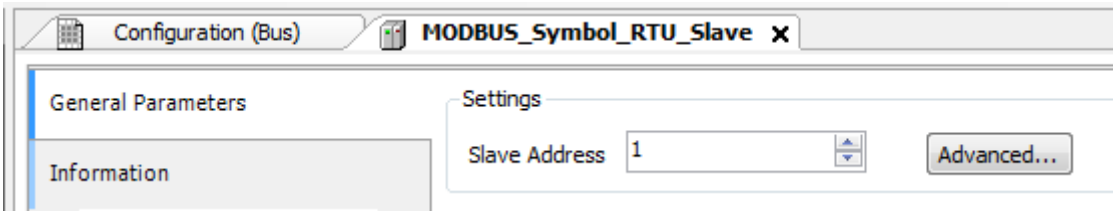


Figure 89: MODBUS RTU Slave Configuration Screen

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

Table 102: 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.

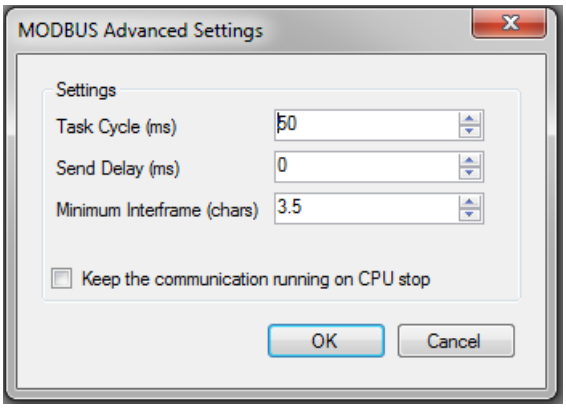


Figure 90: Modbus Slave Advanced Configurations

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 103: 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, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_RTU_SLAVE_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below:

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 1: invalid serial port 2: invalid serial port mode 3: invalid baud rate 4: invalid data bits 5: invalid parity 6: invalid stop bits 7: invalid modem signal parameter 8: invalid UART RX Threshold parameter 9: invalid time-out parameter 10: busy serial port 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid signal modem method 22: CTS time-out = true 23: CTS time-out = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from manual 33: invalid flow control for the configured serial port 34: data reception not allowed in normal mode 35: data reception not allowed in extended mode 36: DCD interruption not allowed 37: CTS interruption not allowed 38: DSR interruption not allowed 39: serial port not configured 50: internal error in the serial port
Command bits, automatically initialized:		

T_DIAG_MODBUS_RTU_SLAVE_1.*	Size	Description
tCommand.bStop	BIT	Stop slave.
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 104: MODBUS RTU Slave Diagnostic

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

5.9.6.1.2. Configuration of the Relations – Symbolic Mapping Setting

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

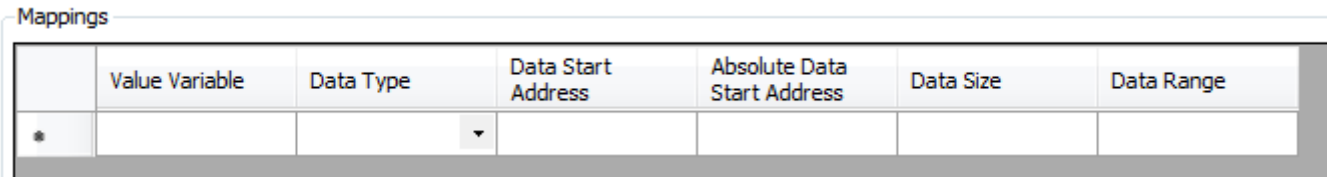


Figure 91: MODBUS Data Mappings Screen

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 Input Status Holding Register 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 105: 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 106: 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.

5.9.7. 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 75.

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

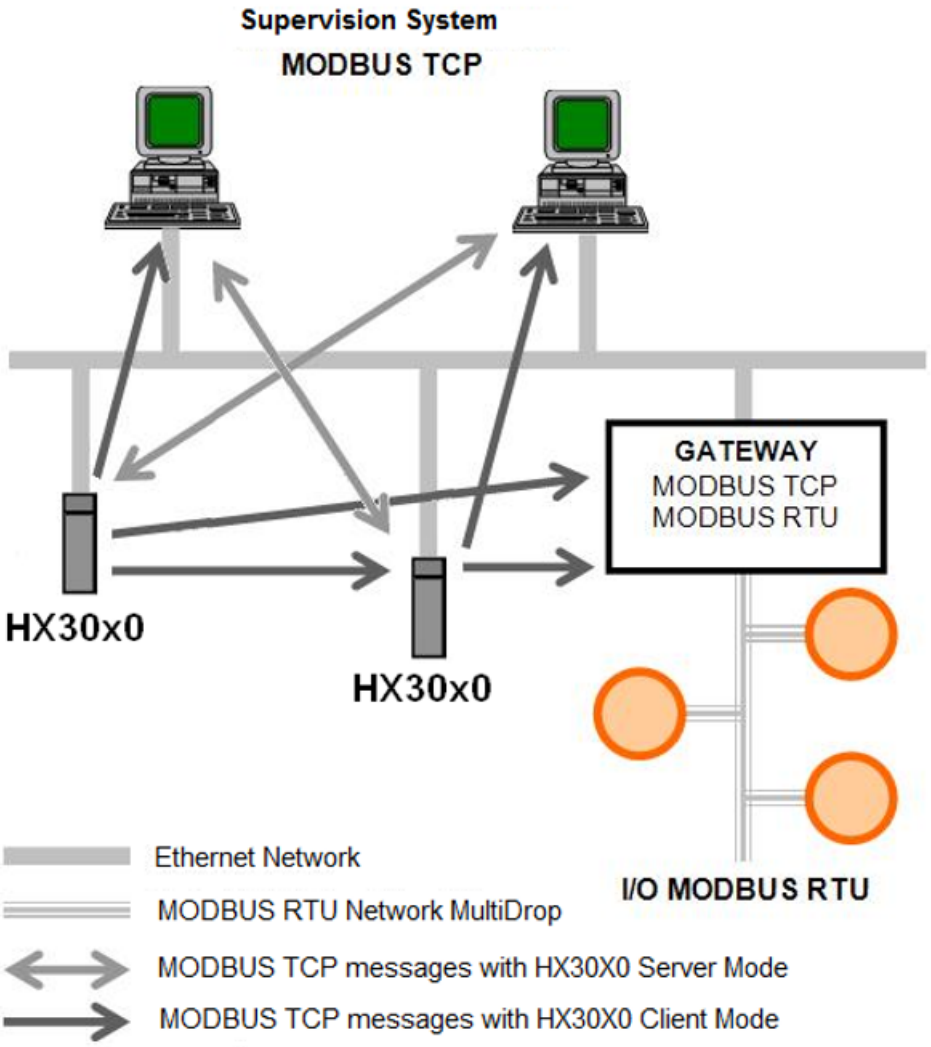


Figure 92: MODBUS TCP Communication Network

The association of MODBUS variables with CPU symbolic variables is made by the user through relations definition via MasterTool IEC XE 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 IEC XE. 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).

Table 107: 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 108: 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 Configuration.

5.9.8. MODBUS Ethernet Client

This protocol is available for all Nexto Series CPUs on its Ethernet channels. When selecting this option at MasterTool IEC XE, 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 the MasterTool IEC XE User Manual – MU299609 or on Inserting a Protocol Instance section.

5.9.8.1. 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.

5.9.8.1.1. 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:

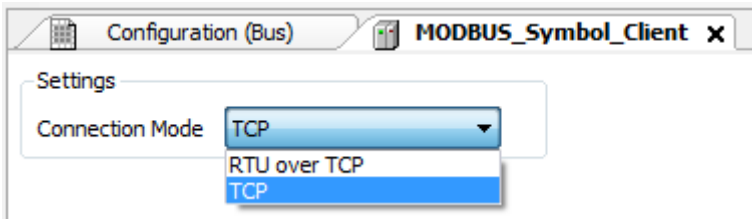


Figure 93: MODBUS Client General Parameters Configuration Screen

Configuration	Description	Default	Options
Connection Mode	Protocol selection	TCP	RTU via TCP TCP

Table 109: MODBUS Client General Configurations

The MODBUS Client protocol diagnostics and commands configured, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_ETH_CLIENT_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below:

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 110: MODBUS Client Protocol Diagnostics

Note:

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

5.9.8.1.2. Device Configuration – Configuration via Symbolic Mapping

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

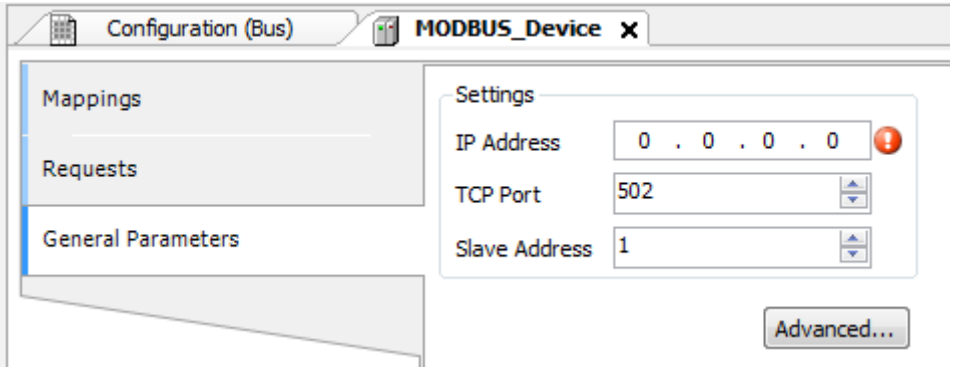


Figure 94: Device General Parameters Settings

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 111: 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. Connection is closed at the end of each communication. Connection is closed after an inactive time of (s): 10 to 3600.
Inactive Time (s)	Inactivity time	10	10 to 3600

Table 112: 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.

5.9.8.1.3. Mappings Configuration – Configuration via Symbolic Mapping

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

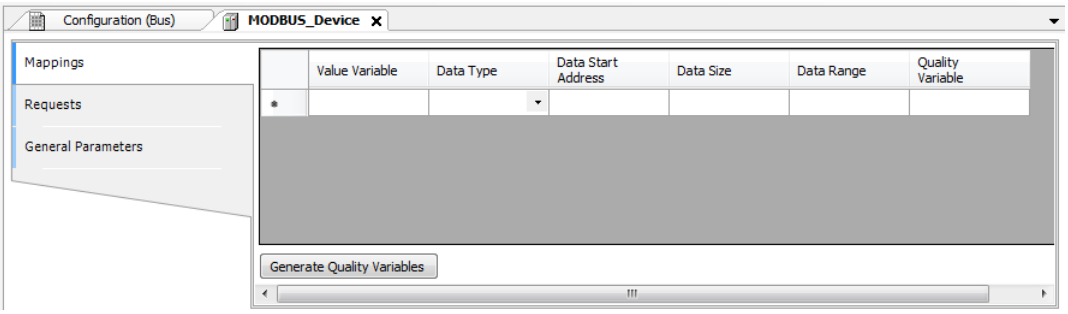


Figure 95: MODBUS Data Type

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) Coil - Read (1 bit) Holding Register - Write (16 bits) Holding Register - Read (16 bits) Holding Register – Mask AND (16 bits) Holding Register – Mask OR (16 bits) Input Register (16 bits) 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 113: 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 114: 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.

5.9.8.1.4. Requests Configuration – Configuration via Symbolic Mapping

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

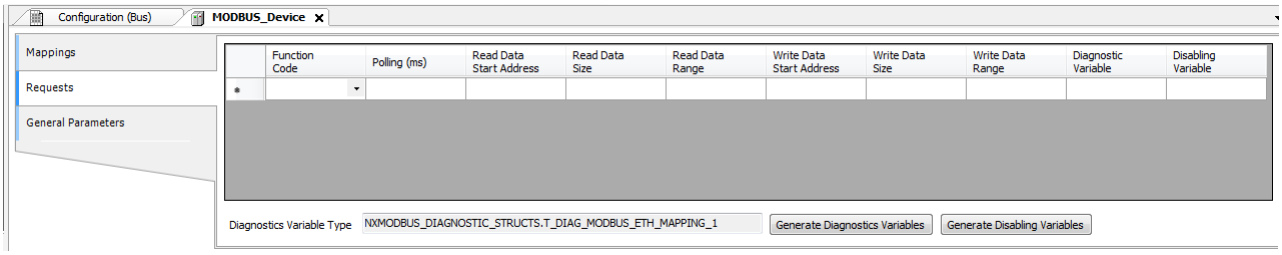


Figure 96: MODBUS Data Request Screen

Configuration	Description	Default Value	Options
Function Code	MODBUS function type	-	01 – Read Coils 02 – Read Input Status 03 – Read Holding Registers 04 – Read Input Registers 05 – Write Single Coil 06 – Write Single Register 15 – Write Multiple Coils 16 – Write Multiple Registers 22 – Mask Write Register 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 115: 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 116: 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 or by direct representation, 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 and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 100.

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 118 for further details.
Last exception code received by master:		
eLastExceptionCode	MODBUS_EXCEPTION (BYTE)	NO_EXCEPTION (0) FUNCTION_NOT_SUPPORTED (1) MAPPING_NOT_FOUND (2) ILLEGAL_VALUE (3) ACCESS_DENIED (128)* MAPPING_DISABLED (129)* 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.

T_DIAG_MODBUS_ETH_MAPPING_1.*	Size	Description
wCommErrorCounter	WORD	Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero.

Table 117: 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.
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 118: 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.

5.9.8.2. 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.

5.9.9. MODBUS Ethernet Server

This protocol is available for all Nexto Series CPUs on its Ethernet channels. When selecting this option at MasterTool IEC XE, 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 the MasterTool IEC XE User Manual – MU299609.

5.9.9.1. 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.

5.9.9.1.1. MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping

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

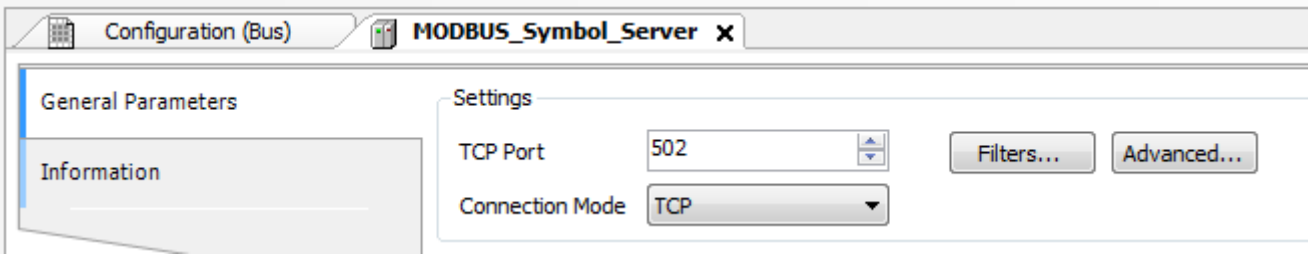


Figure 97: MODBUS Server General Parameters Configuration Screen

Configuration	Description	Default	Options
TCP Port	TCP port	502	2 to 65534
Connection Mode	Protocol selection	TCP	RTU via TCP TCP

Table 119: 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

Configuration	Description	Default Value	Options
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 120: 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*.

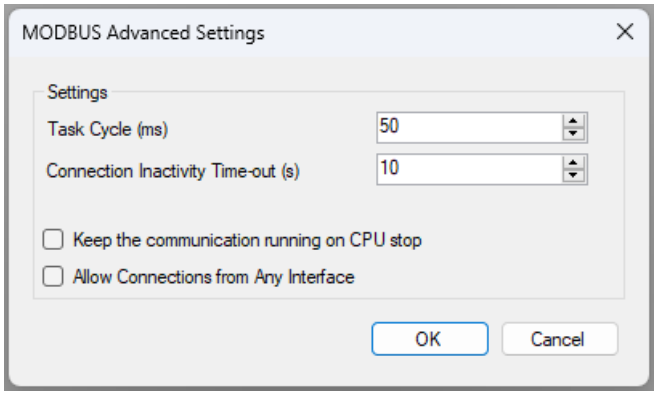


Figure 98: MODBUS Server Advanced Settings Configuration Screen

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
Allow Connections from Any Interface	Enables connections through any network interface, including VPN	Unmarked	Marked or Unmarked

Table 121: 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.

Allow Connections from Any Interface: When enabled, the MODBUS server will accept connections from any available interface on the device, including VPN interfaces. When disabled, it will only accept connections on the interface where it is instantiated. When enabling this option, ensure that no other protocol in the project uses the same port as the MODBUS server to avoid communication conflicts.

5.9.9.1.2. MODBUS Server Diagnostics – Configuration via Symbolic Mapping

The diagnostics and commands of the MODBUS server protocol configured, either by symbolic mapping or by direct representation, are stored in variables of type *T_DIAG_MODBUS_ETH_SERVER_1* and the mapping by direct representation are in 4-byte and 8-word, which are described in table below:

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 bInterruptedBy-Command).
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:		
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 122: 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.

5.9.9.1.3. Mapping Configuration – Configuration via Symbolic Mapping

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

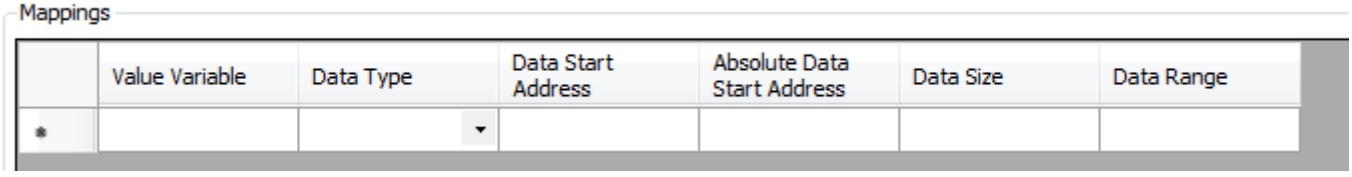


Figure 99: MODBUS Server Data Mappings Screen

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 Input Status Holding Register 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 123: 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.

5.9.10. 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).

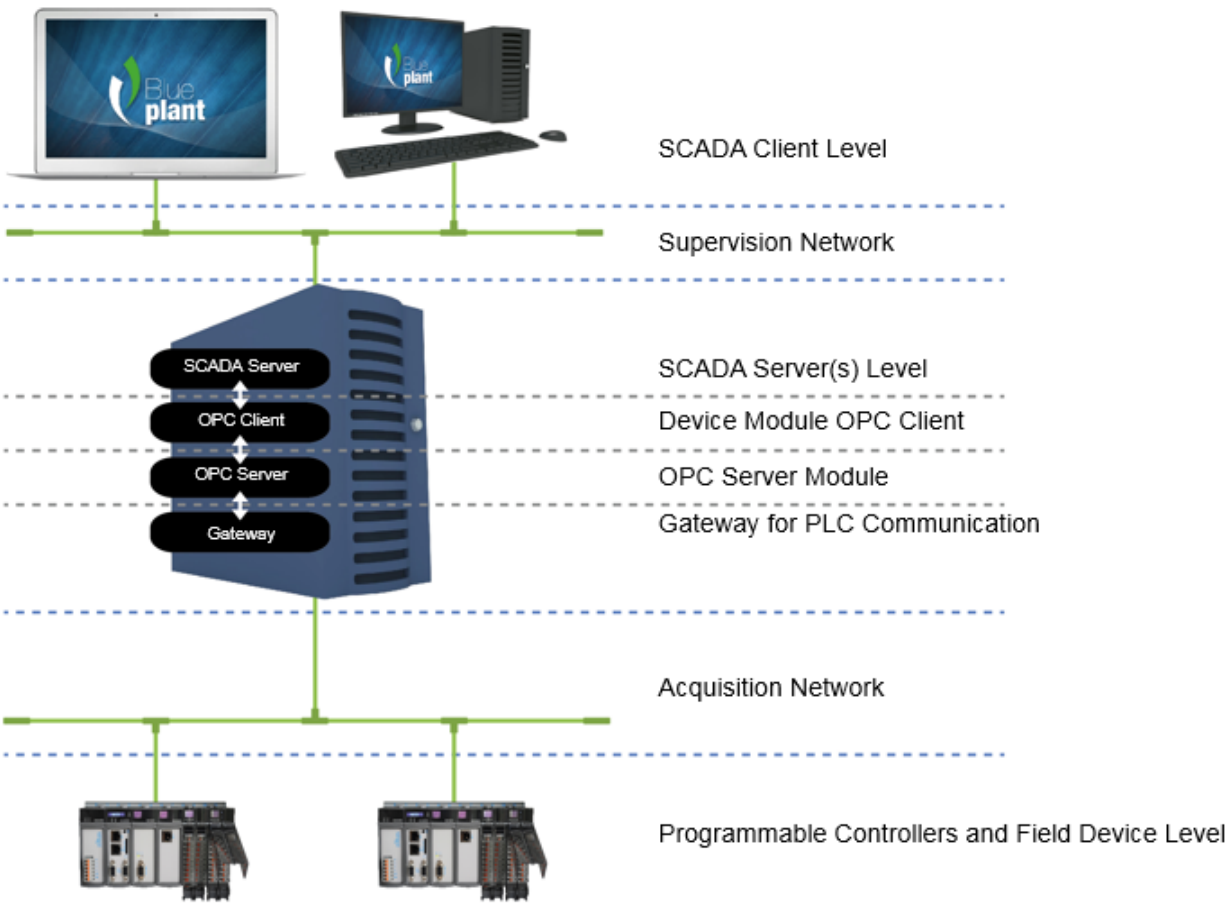


Figure 100: OPC DA Architecture

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 IEC XE User Manual – MU299609.
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.

Role	Description
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.
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 124: 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 IEC XE 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.

5.9.10.1. 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.

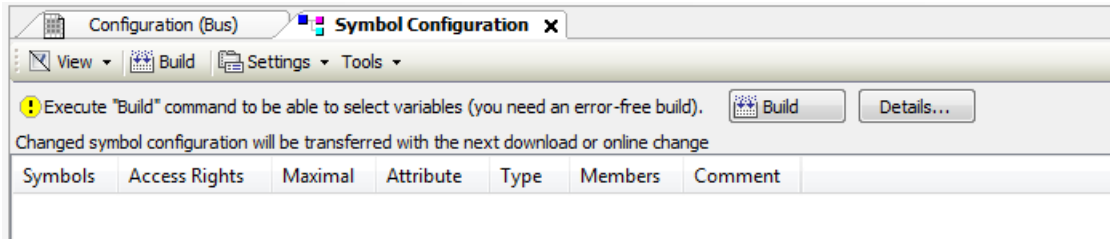


Figure 101: Symbol Configuration Object

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: Read only Write only 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: attribute 'symbol' := 'read' the column shows attribute 'symbol' := 'write' the column shows 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 125: 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 101 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.

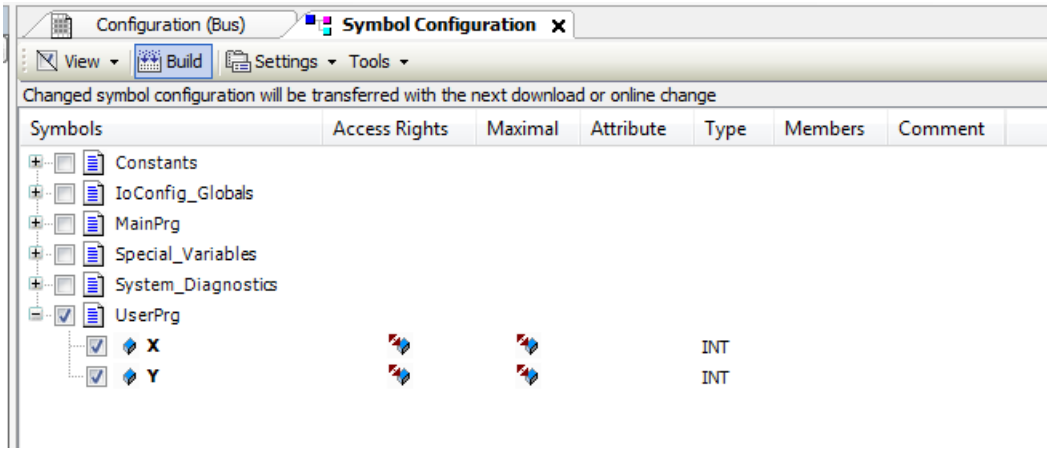


Figure 102: Selecting Variables on the Symbol Configuration

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.

5.9.10.2. Configuring a PLC on the OPC DA Server

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

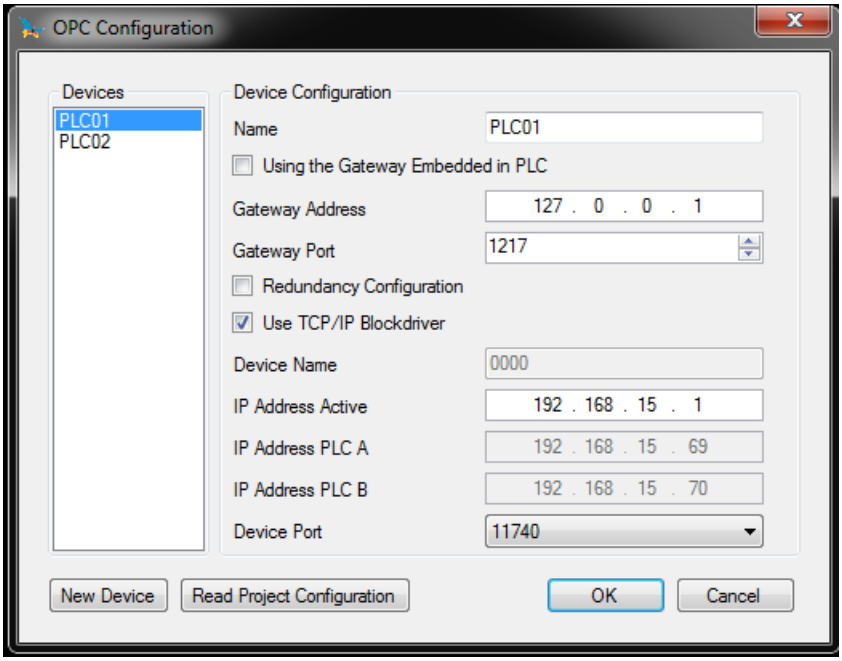


Figure 103: OPC DA Server Settings

The *Gateway Configuration* is the same set in the Gateway used for the communication between the MasterTool IEC XE and the PLC and described in Communication Settings, present in the MasterTool IEC XE User Manual – MU299609. 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 IEC XE.

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 126: 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 IEC XE, 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, the MasterTool IEC XE 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 the MasterTool IEC XE 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 IEC XE. 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.

5.9.10.2.1. Importing a Project Configuration

Using the button *Read Project Configuration*, as shown in Figure 103, 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 the MasterTool IEC XE User Manual – MU299609. 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.

5.9.10.3. 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 the MasterTool IEC XE maintains this configuration valid.

Table 127: 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 128: Description of the OPC Quality value

Table 128 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.

5.9.10.4. 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	-
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 129: OPC DA Server Communication Limits

Note:

Maximum number of variables communicating with a single PLC: There is no configuration limit. The maximum possible number of variables depends on the processing capacity of the device.

ATTENTION

The Maximum number of simultaneous connections of an OPC DA Server in a single PLC is shared with connections made with the MasterTool IEC XE. I.e. the sum of connections of OPC DA Server and MasterTool IEC XE should not exceed the maximum quantity defined in Table 129.

The communication between the OPC DA Server and the PLC uses the same protocol used in the MasterTool IEC XE 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.

5.9.10.5. 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 IEC XE, 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 IEC XE icon and choose *Run as Administrator*.

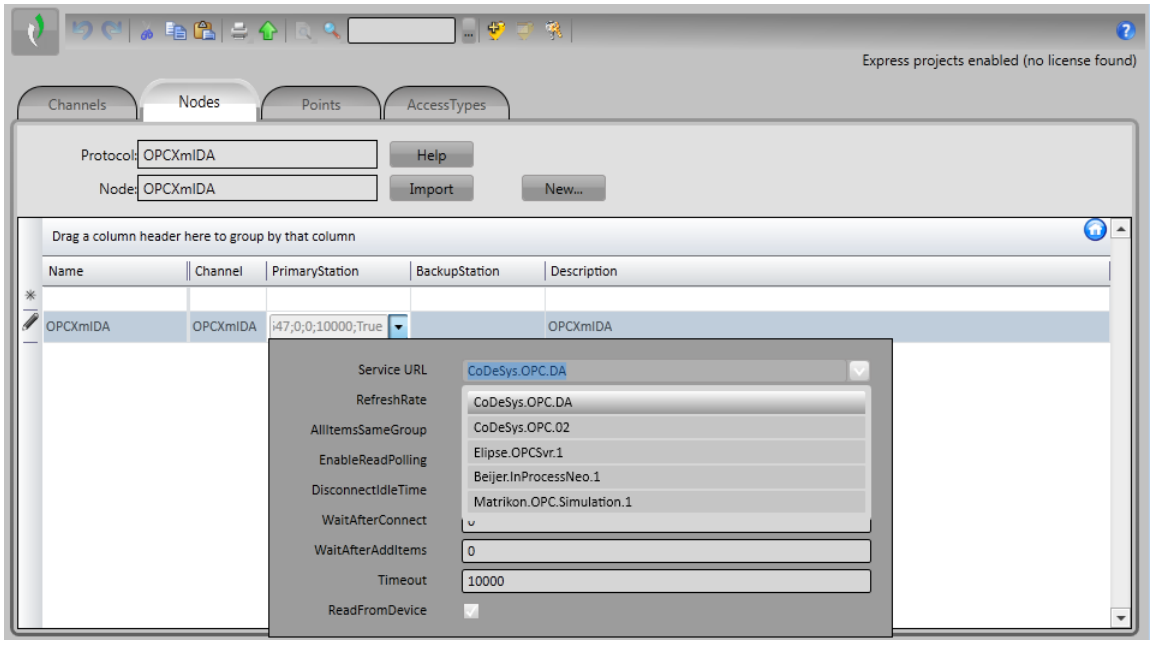


Figure 104: Selecting the OPC DA Server in the Client Configuration

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.

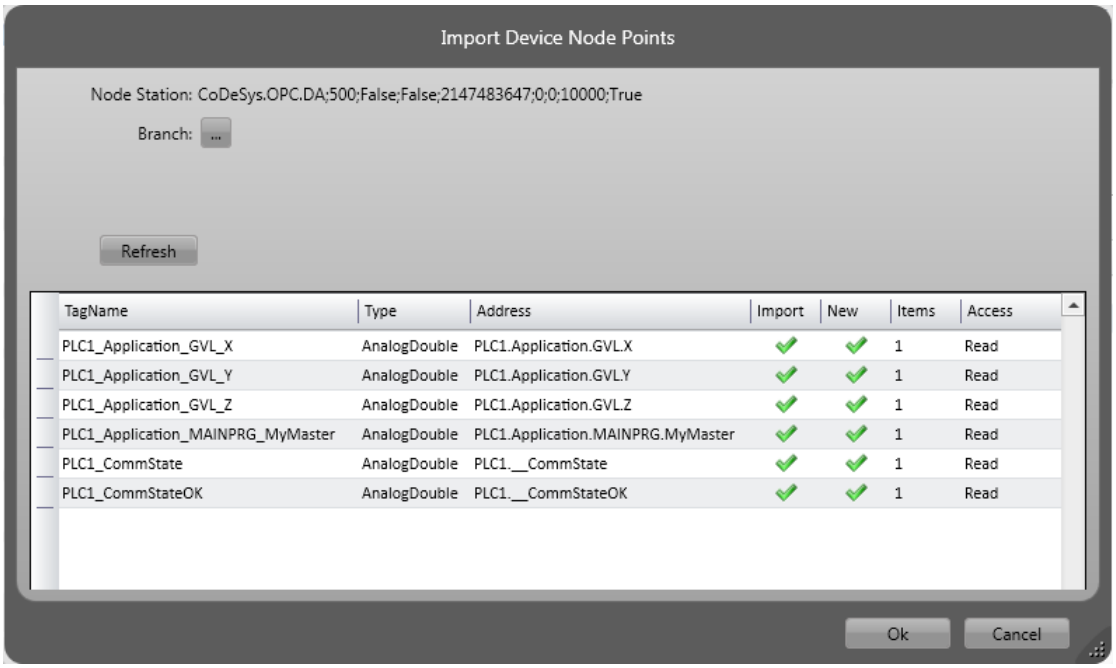


Figure 105: Symbols list consulted by the OPC Client

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 IEC XE software 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 IEC XE User Manual – MU299609.

5.9.11. 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.

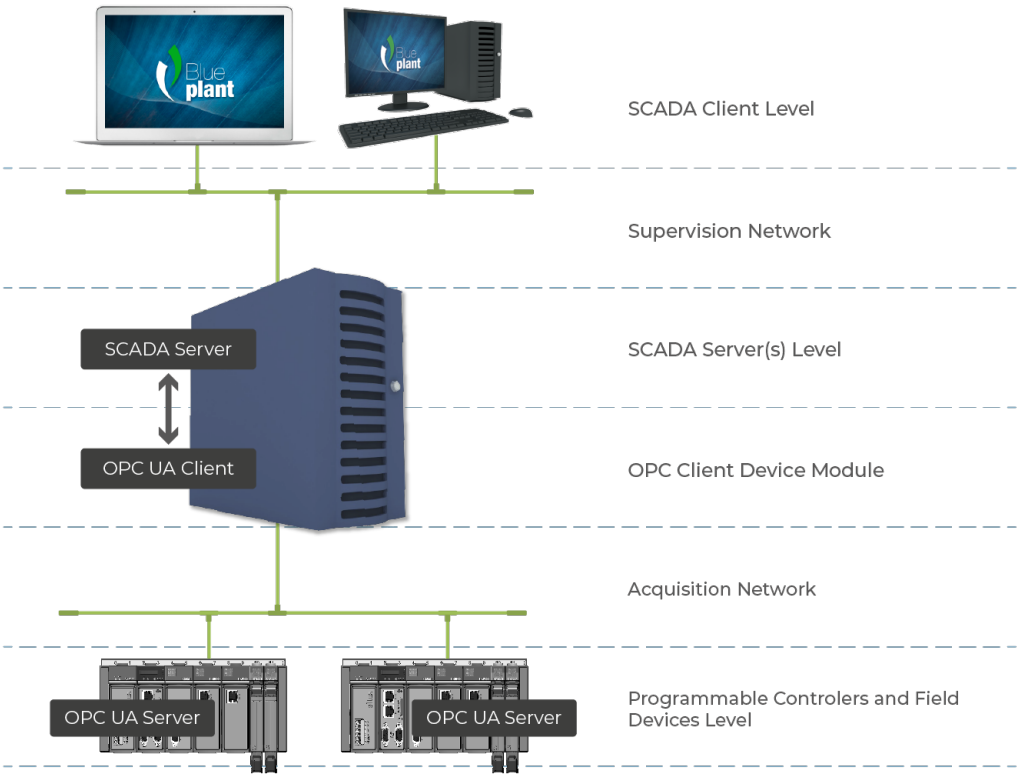


Figure 106: OPC UA Architecture

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 130: 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.

5.9.11.1. 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.

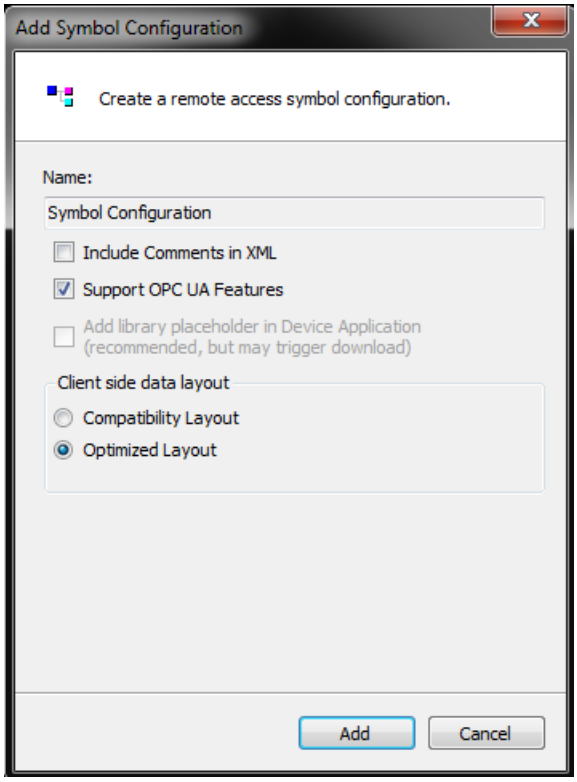


Figure 107: Symbol Configuration Object

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.

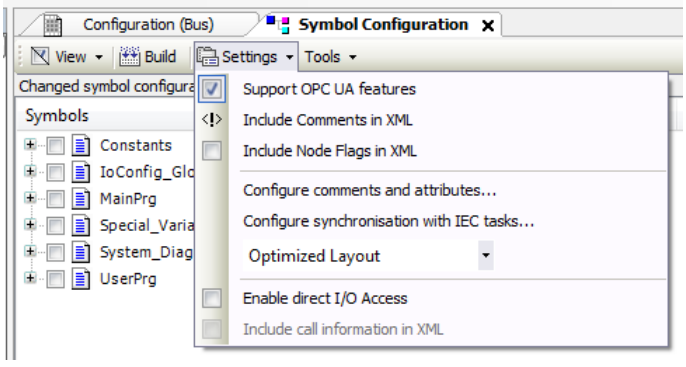


Figure 108: Enabling OPC UA in Object Symbol Configuration

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.

5.9.11.2. 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.

5.9.11.3. Limit Connected Clients on the OPC UA Server

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

5.9.11.4. 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.

5.9.11.5. 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.

5.9.11.6. 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*.

5.9.11.6.1. 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.

5.9.11.6.2. Publishing Interval (ms) and 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.

5.9.11.6.3. Lifetime Count and 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*.

5.9.11.6.4. Queue Size and 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.

5.9.11.6.5. Filter Type and 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.

5.9.11.6.6. PublishingEnabled, MaxNotificationsPerPublish and 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.

5.9.11.7. 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

Like MasterTool IEC XE, some tools need to be run with administrator rights 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*.

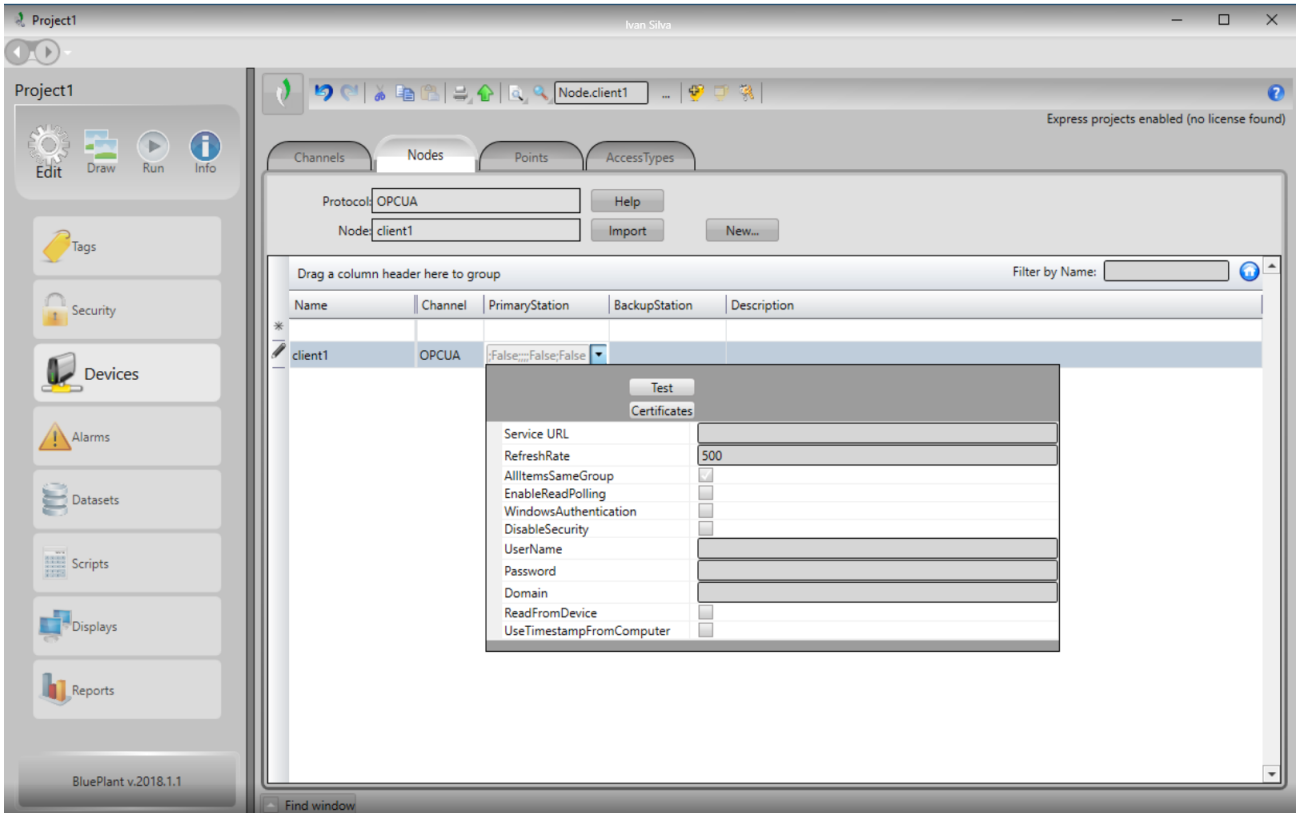


Figure 109: Selecting OPC UA Server in Client Configuration

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.

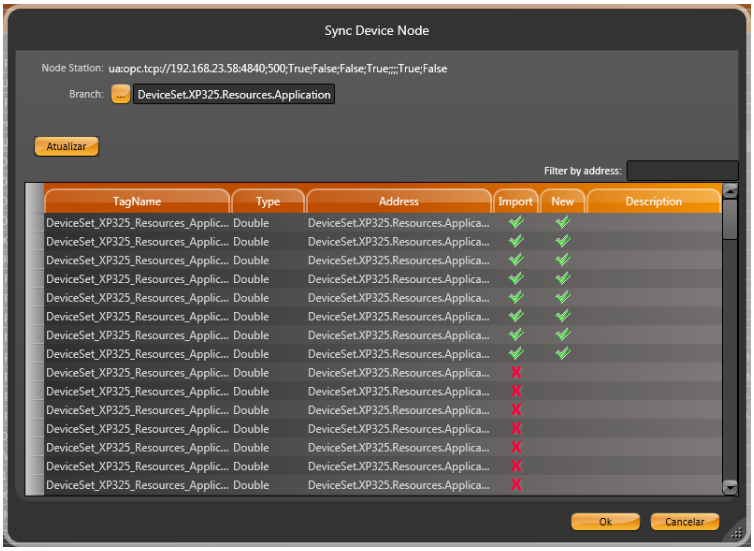


Figure 110: List of Symbols Browsed by OPC UA

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.

5.9.12. 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 μS or 100 servomotors axis each 100 μS 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.

5.9.12.1. 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 IEC XE 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.

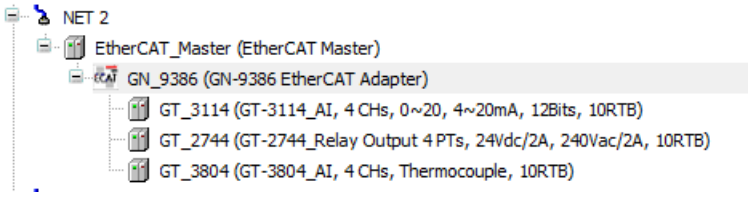


Figure 111: EtherCAT Configuration Example

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.

5.9.12.1.1. 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. In case of future additions of Slave devices, in order to run this command, it's necessary that the EtherCAT network is stopped. To do this, put to TRUE the *bStopBus* bit, present in the variables of the EtherCAT Master Diagnostics.

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.

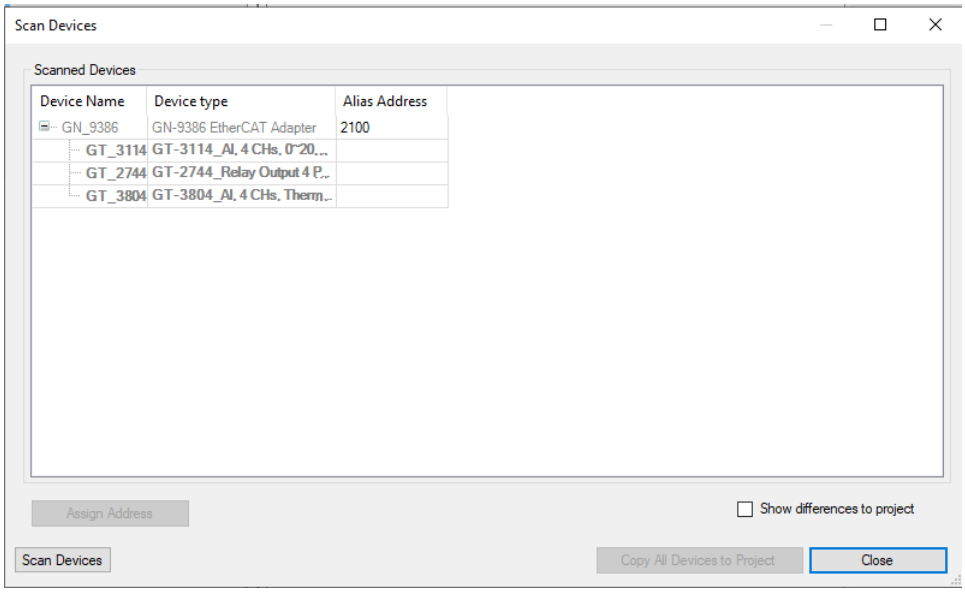


Figure 112: EtherCAT Devices Search Dialog

5.9.12.2. EtherCAT Master Settings

Below are listed the options to carry out the EtherCAT Master configuration, such as defined in *Device Description File*.

5.9.12.2.1. EtherCAT Master - General

Below are the general parameters found in the initial screen of the EtherCAT Master configuration, according figure below.

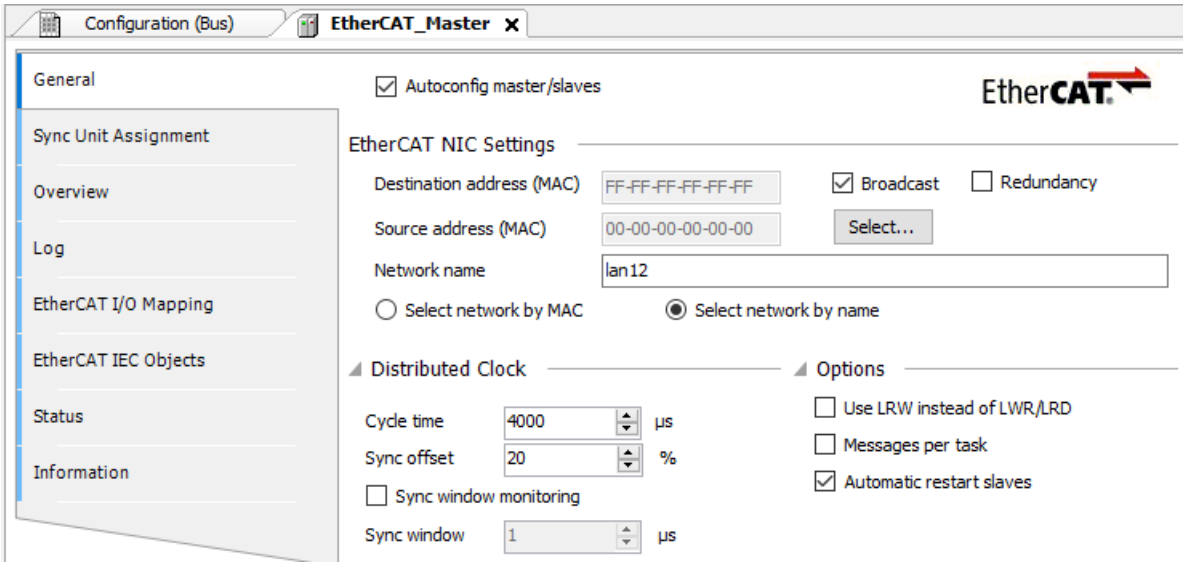


Figure 113: EtherCAT Master Configuration Dialog

Device Configuration	Description	Factory Default	Possible Values
Autoconfig master/slaves	Enable the Master and Slave automatic configuration.	Marked	Marked Unmarked
Cycle time [μs]	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 Unmarked
Sync window [μs]	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 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 Unmarked

Device Configuration	Description	Factory Default	Possible Values
Automatic restart slaves	Restart the devices when the communication is aborted.	Marked	Marked Unmarked

Table 131: 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.

5.9.12.2.2. 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.

5.9.12.2.3. 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.

5.9.12.2.4. 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.

5.9.12.2.5. *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.

5.9.12.2.6. *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*.

5.9.12.3. EtherCAT Slave Configuration

Below are listed the main EtherCAT Slave configuration options, as defined in the *Device Description File*.

5.9.12.3.1. *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.

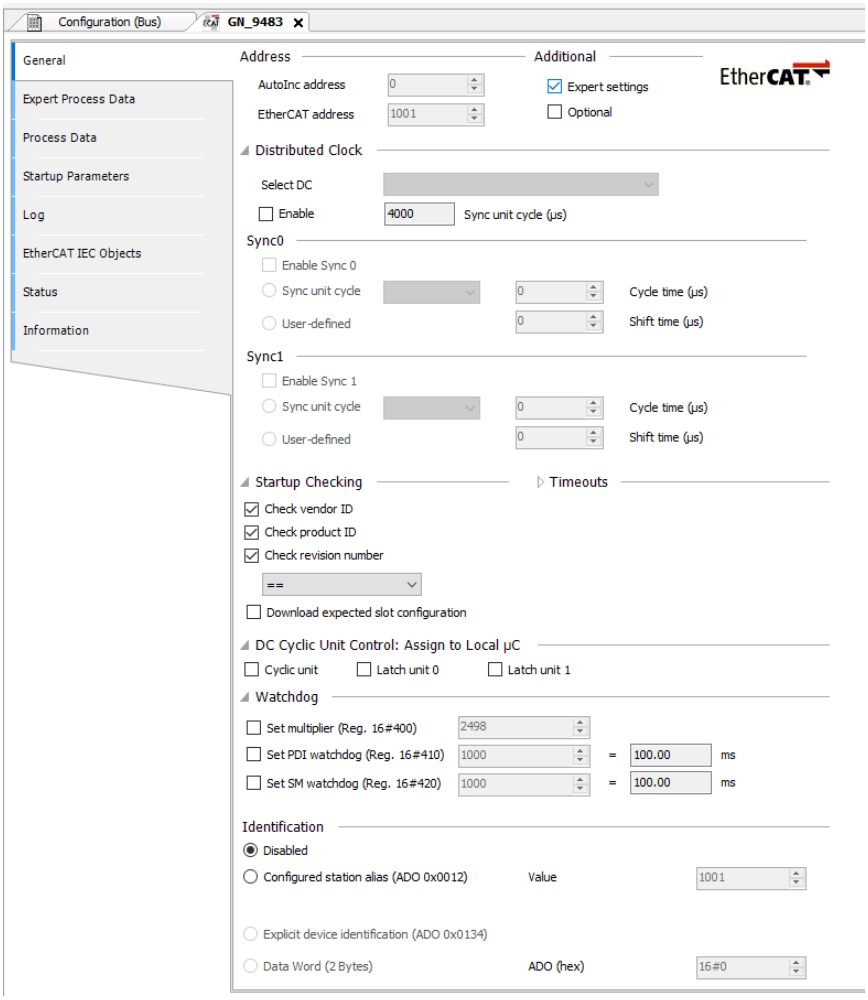


Figure 114: EtherCAT Slave Configuration Dialog

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 Unmarked
Optional	Declare the Slave as Optional.	Unmarked	Marked 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 Unmarked
Sync Unit Cycle [μs]	Show the Cycle Time set in Master.	100000	2000 to 1000000
Enable (Sync 0)	Enable the Sync 0 synchronization unit configurations.	Unmarked	Marked 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 Unmarked
User Defined (Sync 0)	If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time (μs) field.	Unmarked	Marked Unmarked
Cycle Time [μs] (Sync 0)	Show the cycle time currently set.	100000	1 to 2147483647
Shift Time [μs] (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 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 Unmarked
User Defined (Sync 1)	If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time (μs) field.	Unmarked	Marked Unmarked
Cycle Time [μs] (Sync 1)	Show the cycle time currently set.	100000	1 to 2147483647

Device Configuration	Description	Default Value	Options
Shift Time [μs] (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 Unmarked
Check Product ID	If unmarked, it will disable the Product ID Check.	Marked	Marked 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 Unmarked
Latch Unit 0	Set the Latch Unit 0 to the local microprocessor.	Unmarked	Marked Unmarked
Latch Unit 1	Set the Latch Unit 1 to the local microprocessor.	Unmarked	Marked Unmarked

Table 132: 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 (μs)* 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 (μs)* 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 (μs)* 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.

5.9.12.3.2. 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.

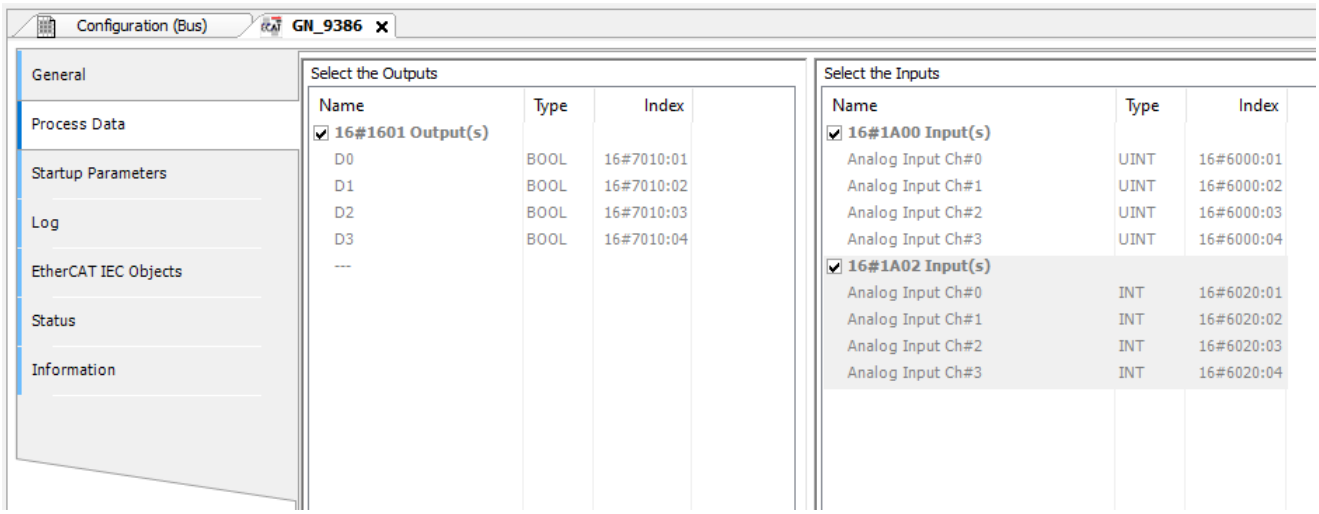


Figure 115: Process Data Dialog

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.

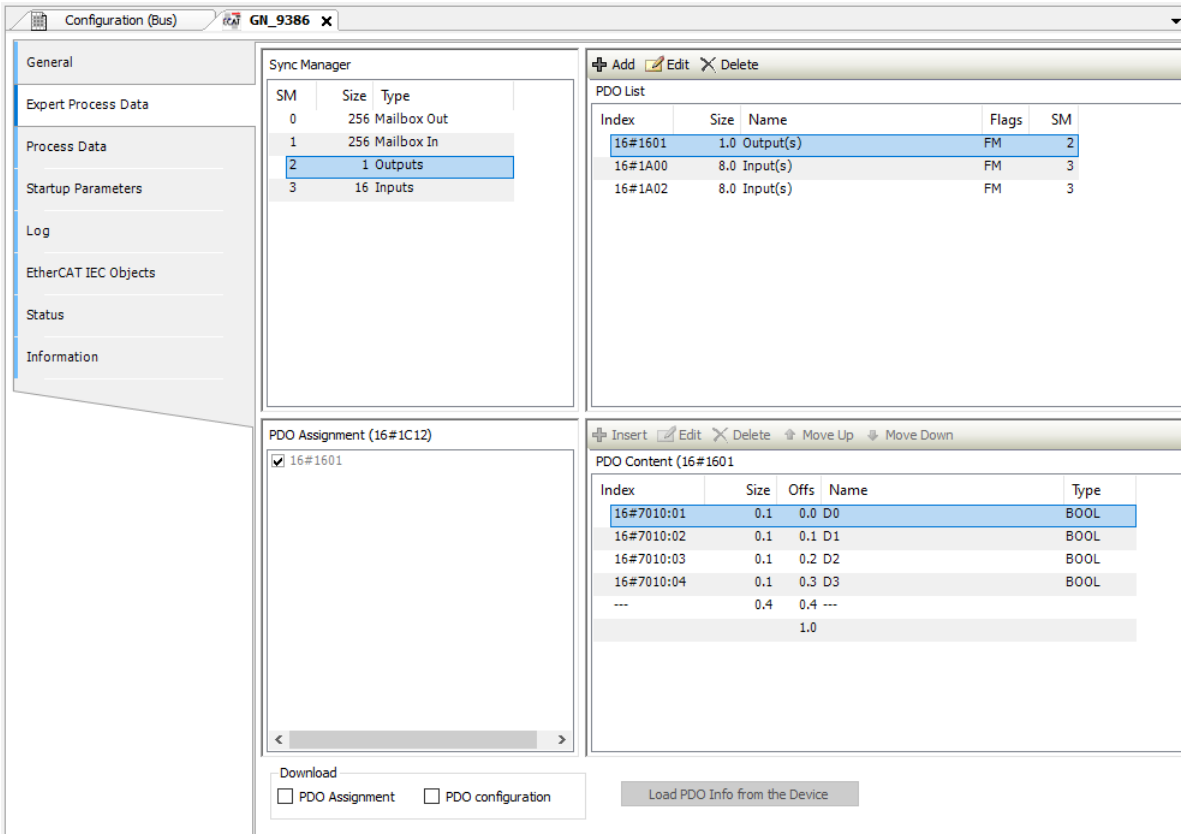


Figure 116: Expert Process Data Dialog

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.

5.9.12.3.3. EtherCAT Slave - Edit PDO List

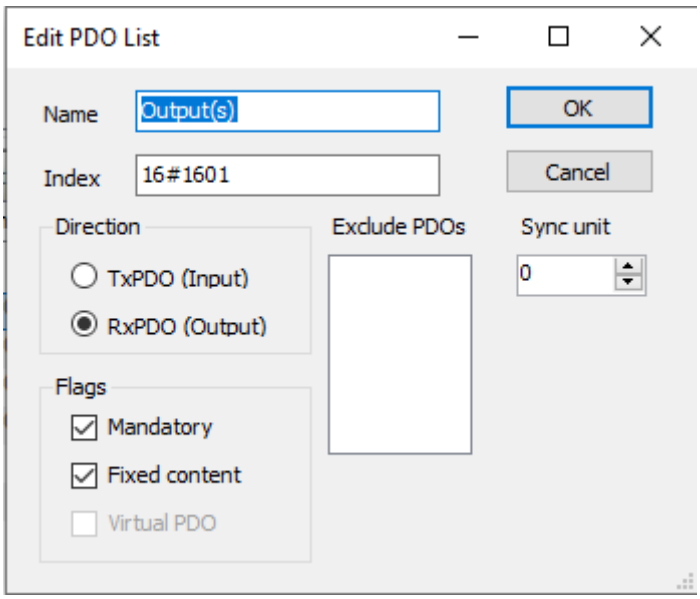


Figure 117: Edit PDO List Dialog

This dialog is opened through the context menu from the PDO List area, presented in Figure 116. 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.

5.9.12.3.4. 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*.

5.9.12.3.5. 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*.

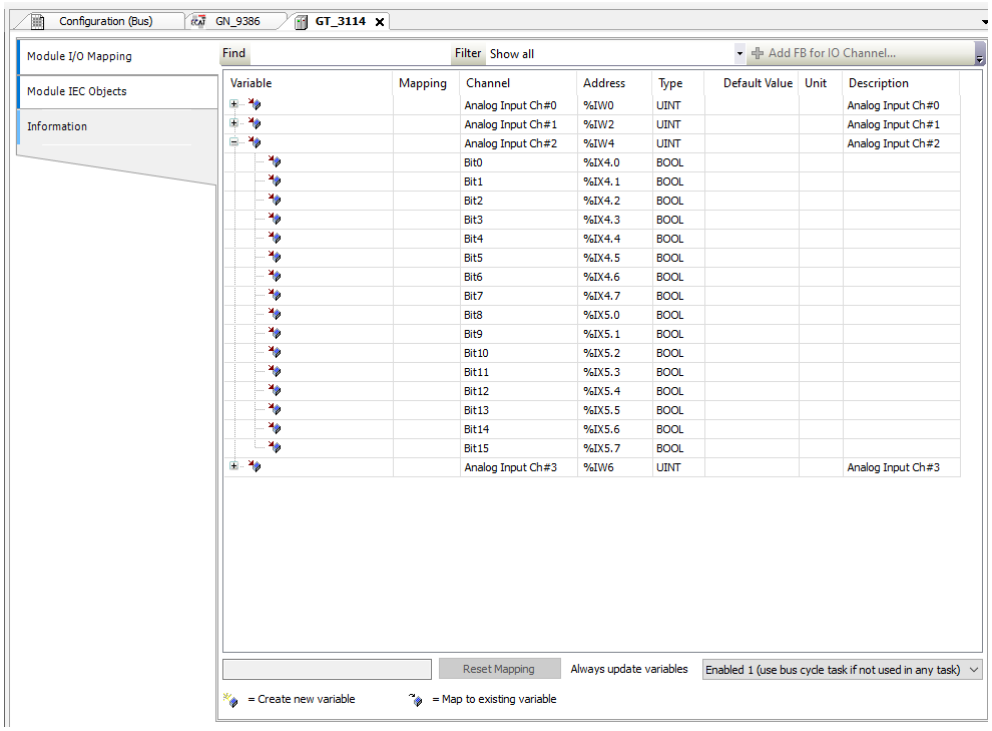


Figure 118: Slave I/O Mapping Dialog

5.9.12.3.6. 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*.

5.9.13. 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.

5.9.13.1. EtherNet/IP

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*.

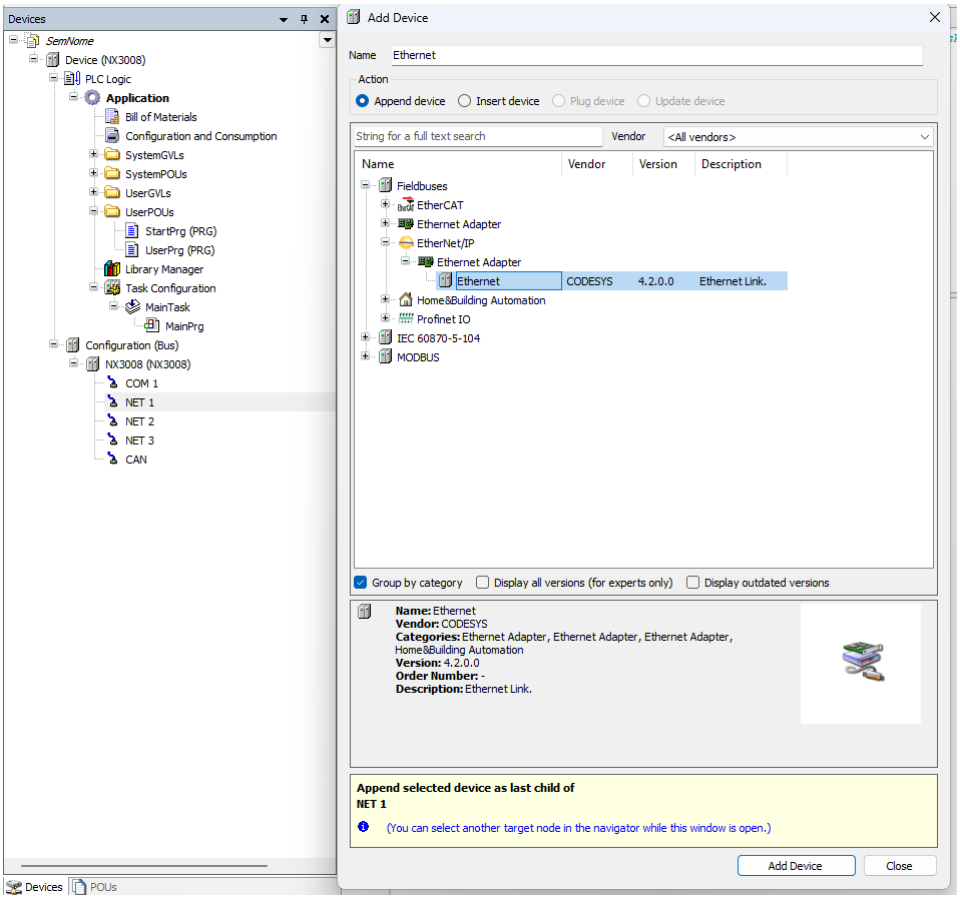


Figure 119: Adding an Ethernet Adapter

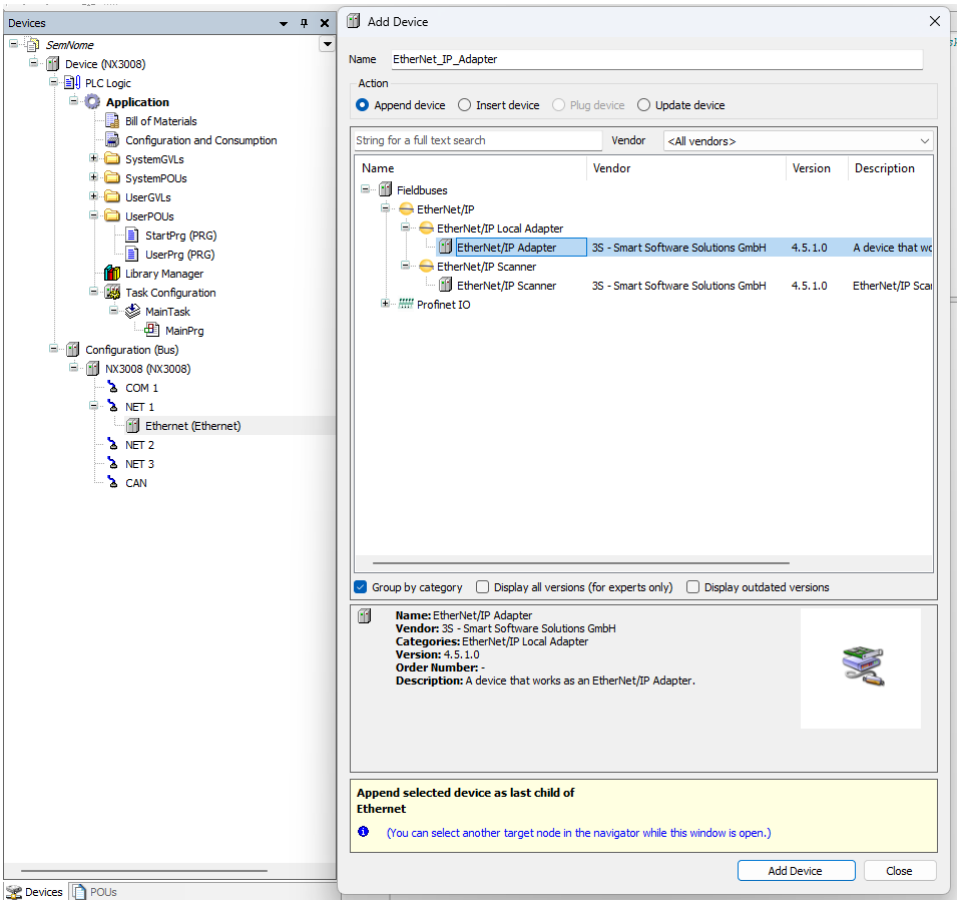


Figure 120: Adding an EtherNet/IP Adapter or Scanner

5.9.13.2. EtherNet/IP Scanner Configuration

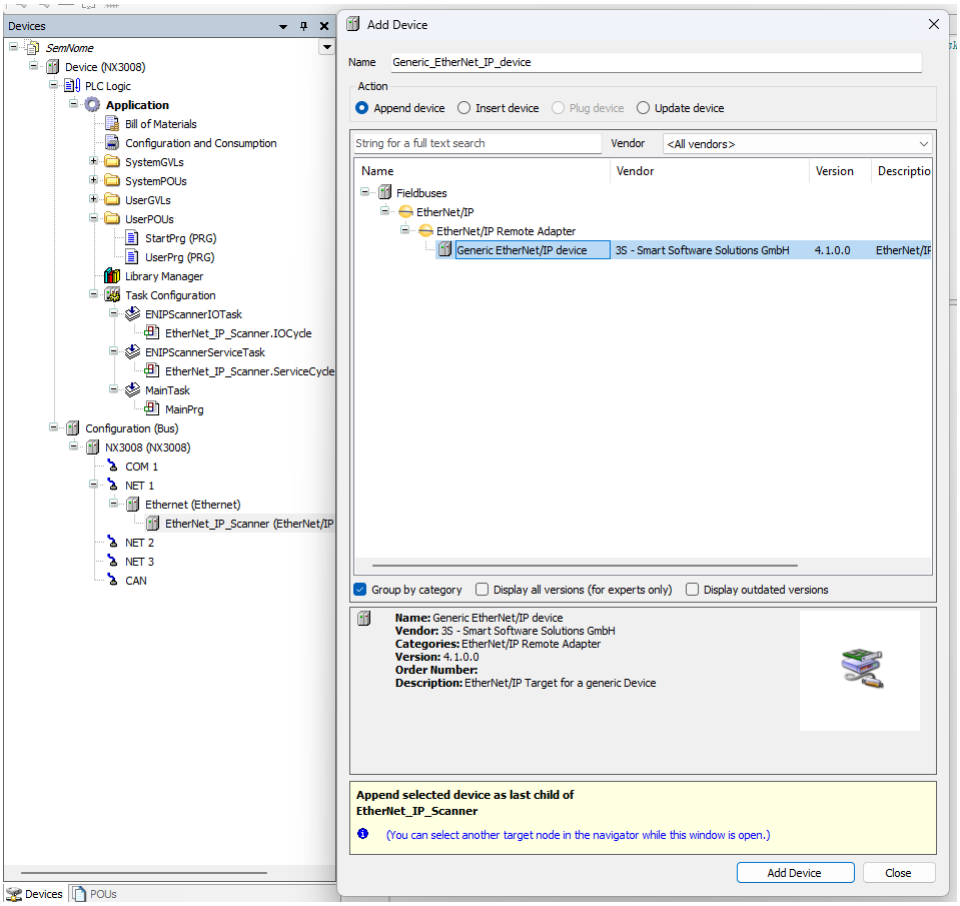


Figure 121: Adding an EtherNet/IP Adapter Under the Scanner

5.9.13.2.1. 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".

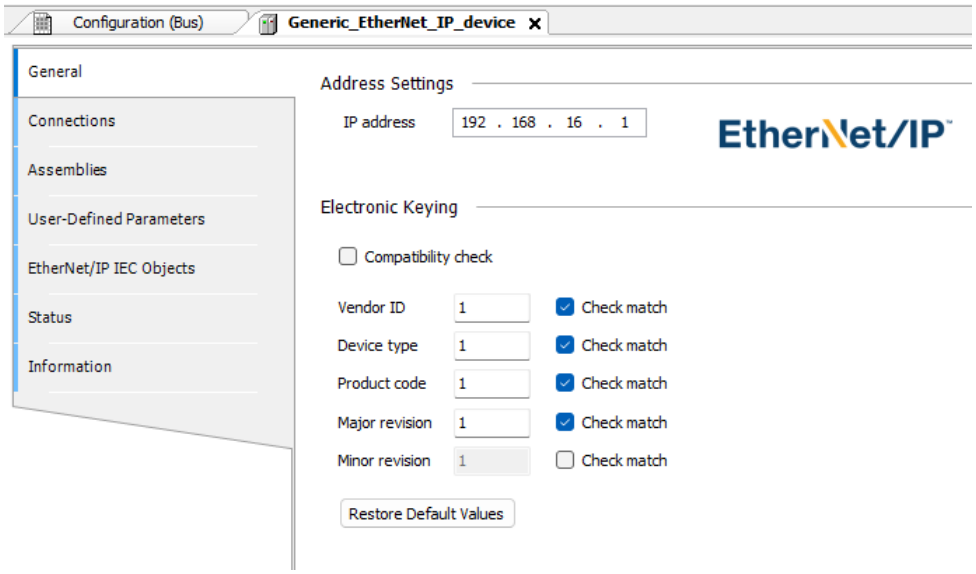


Figure 122: EtherNet/IP General Tab

5.9.13.2.2. 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.

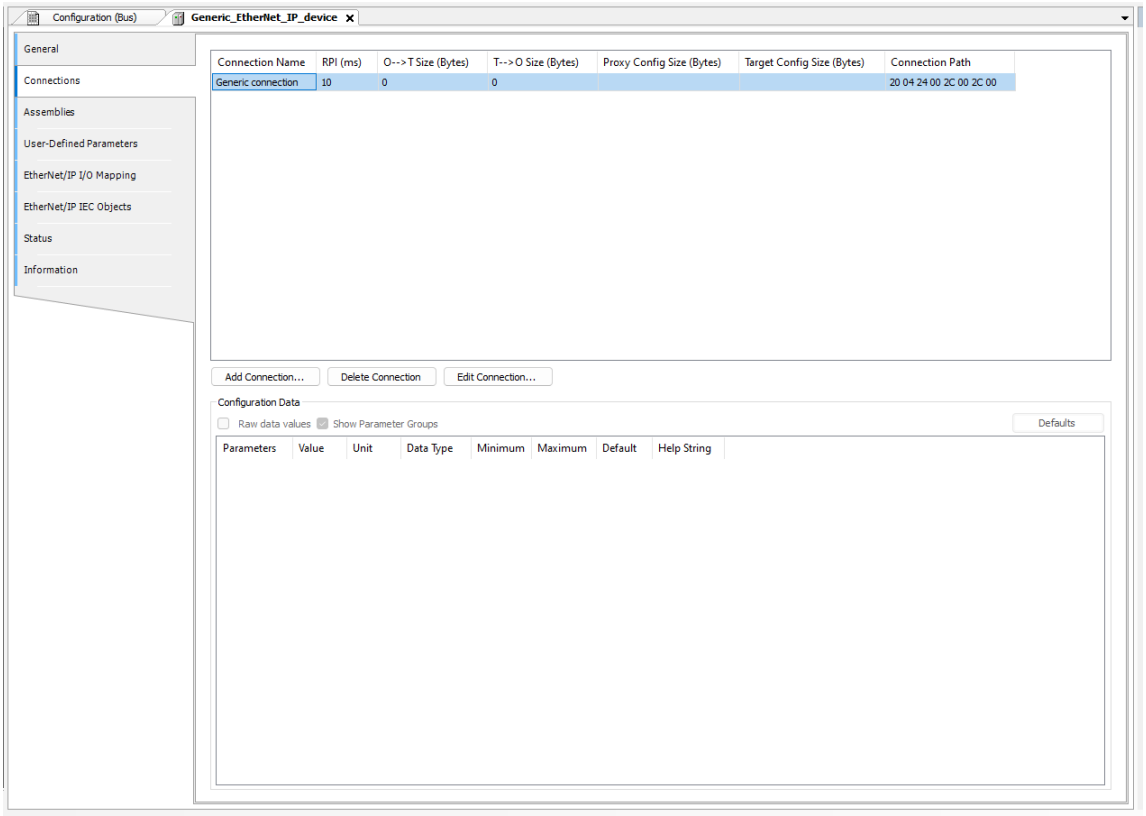


Figure 123: EtherNet/IP Connection Tab

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 133: 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.

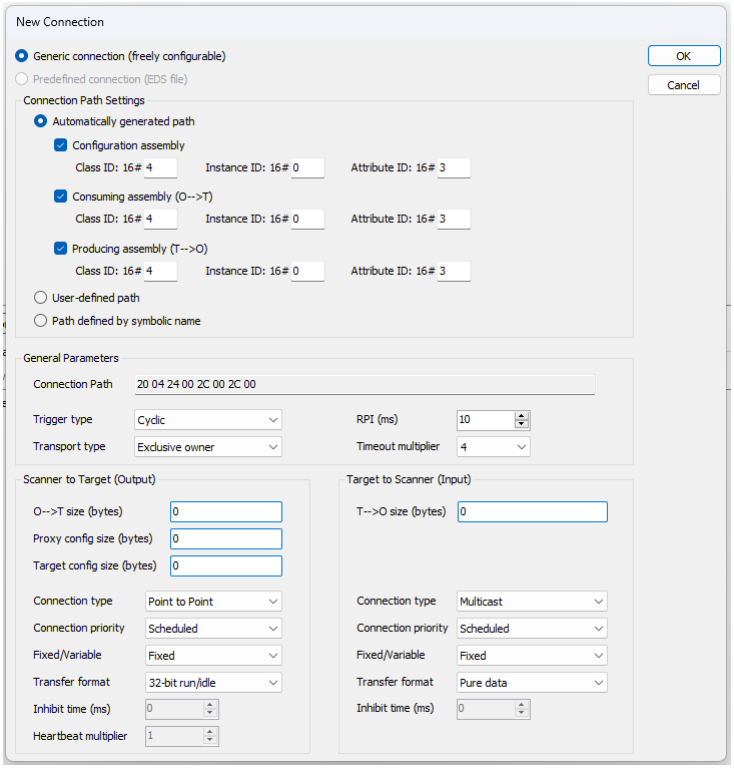


Figure 124: EtherNet/IP New Connection's Window

5.9.13.2.3. *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.

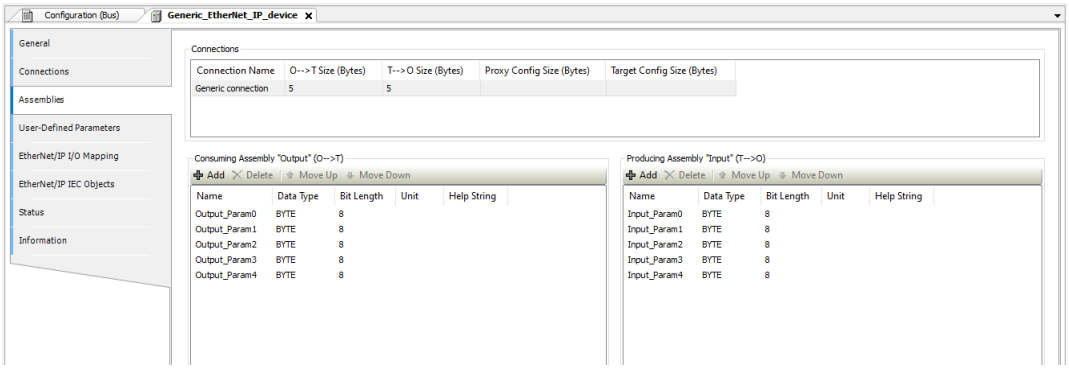


Figure 125: EtherNet/IP Assemblies

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 134: 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 135: EtherNet/IP "Add Input/Output" window

5.9.13.2.4. 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.

5.9.13.3. 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.

5.9.13.3.1. 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.

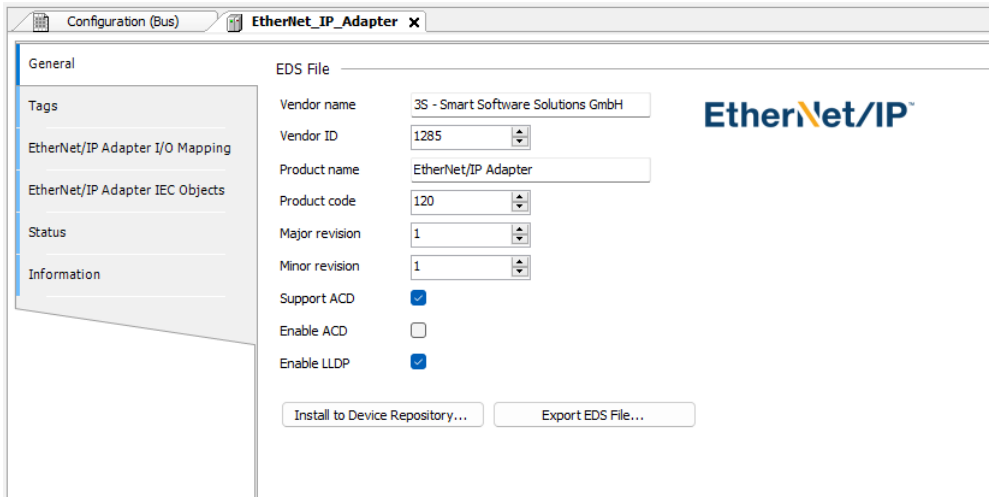


Figure 126: EtherNet/IP General Tab

5.9.13.3.2. 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.

5.9.13.4. EtherNet/IP Module Configuration

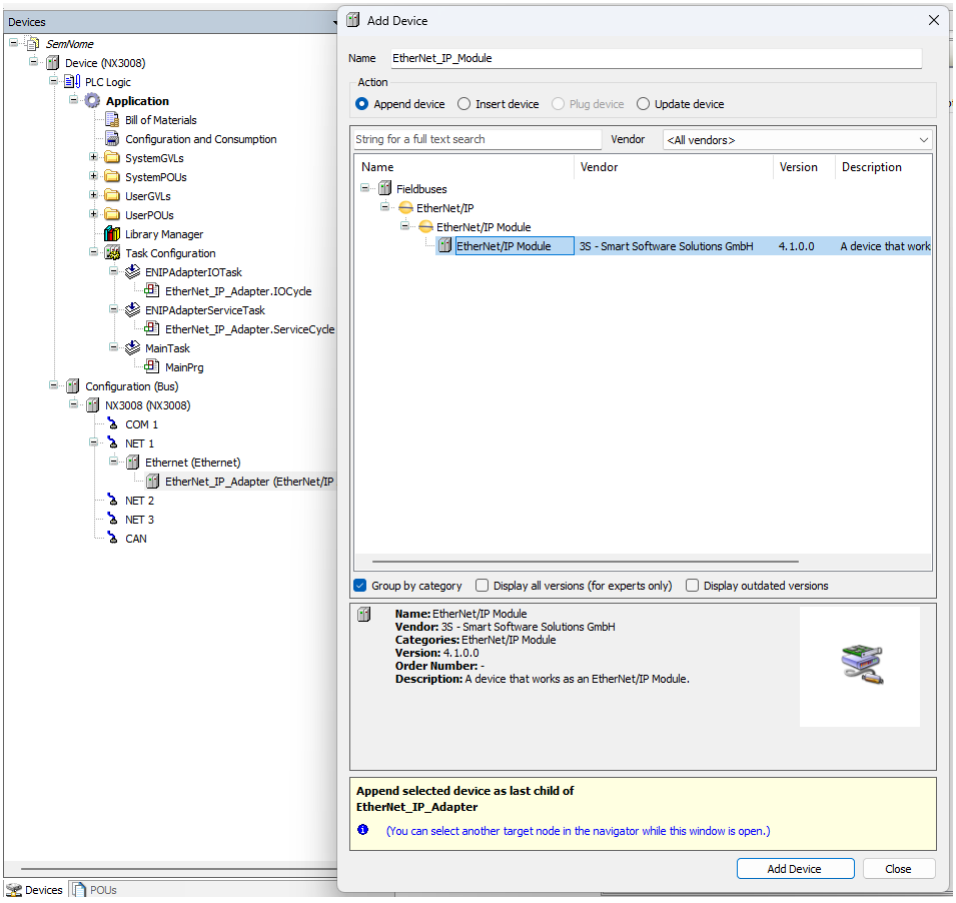


Figure 127: Adding an EtherNet/IP Module under the Adapter

5.9.13.4.1. Assemblies

The parameters of the module's *General* tab follow the same rules as described in the 134 and 135 tables.

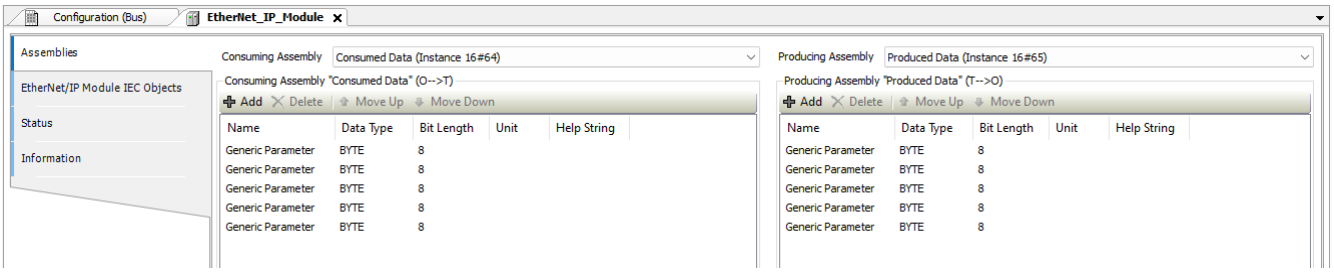


Figure 128: EtherNet/IP Module Assemblies tab

5.9.13.4.2. 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.

5.9.14. IEC 60870-5-104 Server

As select this option at MasterTool, the CPU starts to be an IEC 60870-5-104 communication server, allowing connection with up to three client devices. To each client the driver owns one exclusive event queue with the following features:

- Size: 1000 events
- Retentivity: non retentive
- Overflow policy: keep the newest

To configure this protocol, it is needed to do the following steps:

- Add a protocol IEC 60870-5-104 Server instance to one of the available Ethernet channel. To realize this procedure consult the section Inserting a Protocol Instance
- Configure the Ethernet interface. To realize this procedure consult the section Ethernet Interface Configuration
- Configure the general parameters of protocol IEC 60870-5-104 Server with connection mode Port or IP, and the TCP port number when the selected connection mode is IP
- Add and configure devices, defining the proper parameters
- Add and configure the IEC 60870-5-104 mappings, specifying the variable name, type of object, object address, size, range, dead band and type of dead band
- Configure the link layer parameters, specifying the addresses, communication time-outs and communication parameters
- Configure the application layer parameters, synchronism configuration, commands, as well as transmission mode of *Integrated Totals* objects

The descriptions of each configuration are related below, in this section.

5.9.14.1. Type of data

The table below shows the supported variable type by the Nexto Series CPU for each protocol IEC 60870-5-104 data type.

Object Type	IEC Variables Type
Single Point Information (M_SP_NA)	BOOL
	BIT
Double Point Information (M_DP_NA)	DBP
Step Position Information (M_ST_NA)	USINT
Measured Value, normalized value (M_ME_NA)	INT
Measured Value, scaled value (M_ME_NB)	INT
Measured Value, short floating point value (M_ME_NC)	INT
	UINT
	DINT
	UDINT
	REAL
Integrated Totals (M_IT_NA)	INT
	DINT
Bitstring Information (M_BO_NA)	DWORD
Single Command (C_SC_NA)	BOOL
	BIT
Double Command (C_DC_NA)	DBP
Regulating Step Command (C_RC_NA)	DBP
Setting Point Command, normalized Value (C_SE_NA)	INT
Setting Point Command, scaled Value (C_SE_NB)	INT
Setting Point Command, short floating point Value (C_SE_NC)	REAL
Bitstring Command (C_BO_NA)	DWORD

Table 136: Variables Declaration to IEC 60870-5-104

Notes:

Regulating Step Command: The *Lower* and *Higher* object states of the C_RC_NA are associated respectively to *OFF* and *ON* internal *DBP* type states.

Step Position Information: According to item 7.3.1.5 of Standard IEC 60870-5-101, this 8 bit variable is compose by two fields: value (defined by the 7 bits less significant) and transient (defined as the most significant bit, which indicates when the measured device is transitioning).

Below, there is a code example for fields manipulation in an USINT type variable. Attention, because this code doesn't consist if the value is inside the range, therefore this consistency remains at user's charge.

```
PROGRAM UserPrg
VAR
usiVTI: USINT;  // Value with transient state indication, mapped for the Client
siValue: SINT;  // Value to be converted to VTI. Must be between -64 and +63
bTransient: BOOL; // Transient to be converted to VTI
END_VAR

usiVTI := SINT_TO_USINT(siValue) AND 16#3F;
IF siValue < 0 THEN
usiVTI := usiVTI OR 16#40;
END_IF
IF bTransient THEN
usiVTI := usiVTI OR 16#80;
END_IF
```

```
PROGRAM UserPrg
VAR
  iAnalogIn:    INT;
  iAnalogOut:   INT;
  diCounter:    DINT;
END_VAR

// Analog input conversion from WORD (PROFIBUS) to INT (IEC104)
iAnalogIn:= WORD_TO_INT(wNX6000in00);

// Analog output conversion from INT(IEC104) to WORD (PROFIBUS)
wNX6100out00:= INT_TO_WORD(iAnalogOut);

// Counter conversion from WORDs high+low (PROFIBUS) to DINT (IEC104)
diCounter:= DWORD_TO_DINT(ROL(WORD_TO_DWORD(wNX1005cnt00H), 16) OR wNX1005cnt00L
   );
```

5.9.14.2. Double Points

The double digital points are used to indicate equipment position, such as valves, circuit breakers and secctioners, where the transition between open and close states demand a determined time. Can thus indicate an intermediary transition state between both final states.

Double digital points are also used as outputs and, in an analogous way, it is necessary to keep one of the outputs enabled for a certain time to complete the transition. Such actuation is done through pulses, also known by trip/close commands, with determined duration (enough to the switching of the device under control).

Consult the Double Points section of Utilization Manual for information about double digital points through DBP data type.

Once the Nexto Series digital input and output modules don't support DBP points mapping, some application trickery are needed to make it possible. Remembering that is also not possible to use the *PulsedCommand* function, defined at the *LibRtuStandard* library, to operate the Nexto Series digital double points.

5.9.14.2.1. Digital Input Double Points

For the digital input modules it is needed two auxiliary variables' declaration, to be mapped on the digital input module, besides the double point that is wished to map on the server:

- The double point value variable: type DBP
- The simple point OFF/TRIP value variable: type BOOL
- The simple point ON/CLOSE value variable: type BOOL

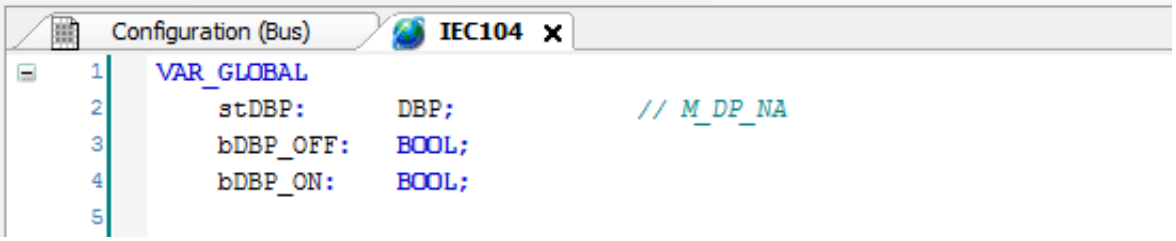


Figure 129: Double Point Variables Declaration Example

Done the variables declaration, it is necessary to create a link between the value variables and the digital input module quality, through the CPU's *Internal Points* tab:

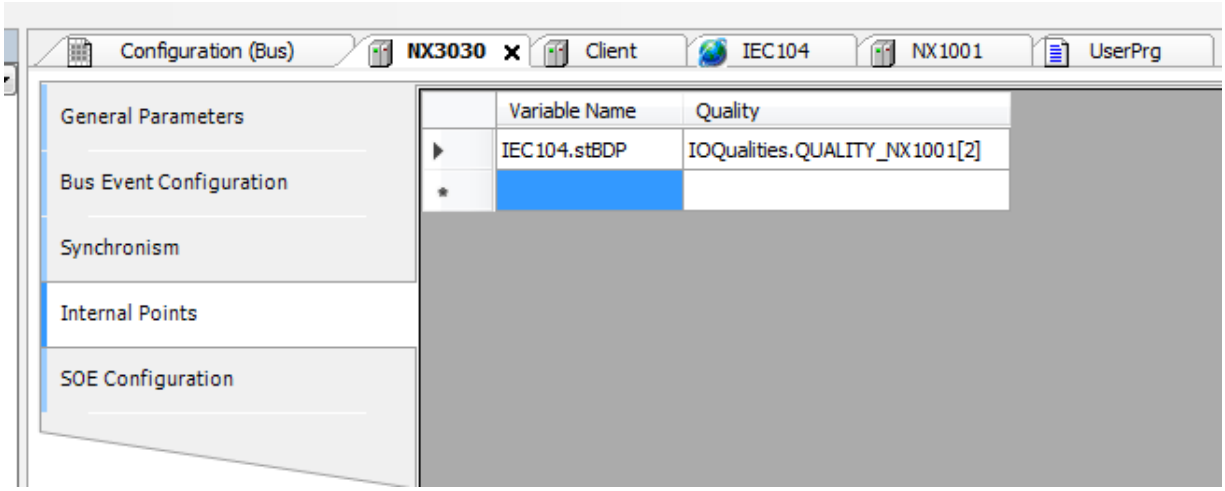


Figure 130: Double Point Variables Attribution to Internal Points

The double point value variable must be mapped at the server IEC 60870-5-104 driver, and both simple variables at the Nexto Series digital input module (in that example, a NX1001). Typically the OFF (TRIP) state is mapped to the even input and the ON (CLOSE) state to the odd input.

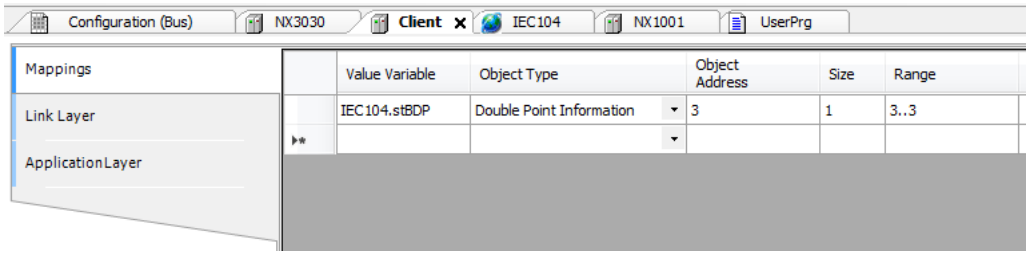


Figure 131: Double Point Variables Mapping on the Client IEC 60870-5-104

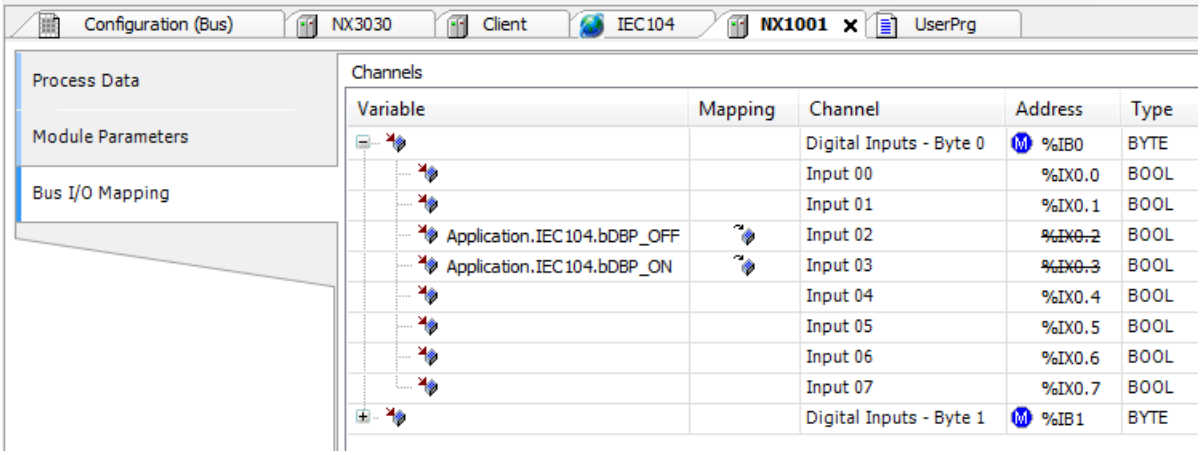


Figure 132: Variables Mapping at the Module Inputs

At last, the user must insert two code lines in its application, to be cyclically executed, to simple variables value attribution to double point:

- DBP value variable, index ON, receive simple point ON value
- DBP value variable, index OFF, receive simple point OFF value

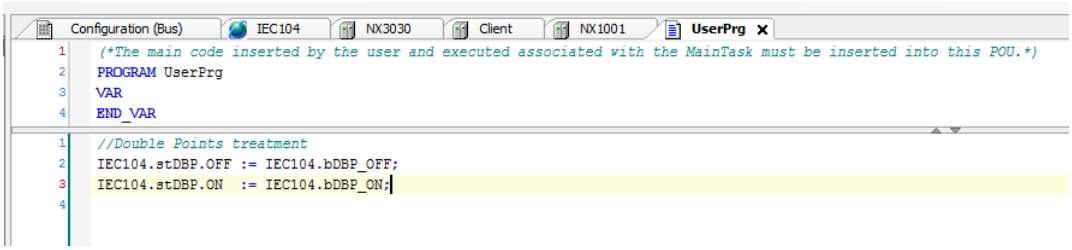


Figure 133: Variables' Values Attribution to the Double Point

5.9.14.2.2. *Digital Output Double Points*

For the digital output modules it must be used the *CommandReceiver* function block to intercept double points actuation commands originated from the clients IEC 60870-5-104. Consult the section Interception of Commands Coming from the Control Center for further information.

The example code below, POU *CmdRcv*, treats pulsed commands received from clients for a digital double point, mapped in a NX2020 module. Besides the following ST code it is need to map a DBP point in Nexto's IEC 60870-5-104 server.

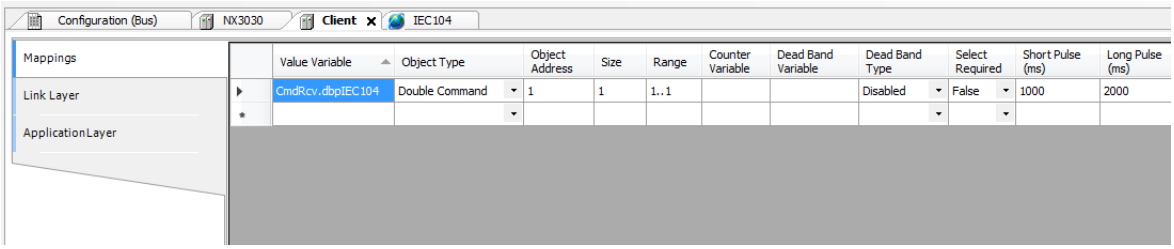


Figure 134: Mapping of Digital Output Double Point variables on IEC 60870-5-104 Client

```
PROGRAM CmdRcv
VAR
CmdReceive:    CommandReceiver; // Interceptor Instance
fbPulsedCmd:   PulsedCommandNexto; // Pulsed Command Instance
byResult:    BYTE; // Pulsed command result
dbpIEC104:     DBP; // Variable mapped in the IEC 104
bSetup:    BOOL:= TRUE; // Interceptor initial setup
END_VAR

// Executes the function configuration in the first cycle
IF bSetup THEN
CmdReceive.dwVariableAddr:= ADR(dbpIEC104);
CmdReceive.bExec:= TRUE;
CmdReceive.eCommandResult:= COMMAND_RESULT.NONE;
CmdReceive.dwTimeout:= 256 * 10;
bSetup:= FALSE;
END_IF

// In case a command is captured:
IF CmdReceive.bCommandAvailable THEN

// Treats each one of the possible commands
CASE CmdReceive.sCommand.eCommand OF

COMMAND_TYPE.NO_COMMAND:

  // Inform that there is an invalid command.
```

```
    // Does nothing and must move on by time-out.

  COMMAND_TYPE.SELECT:

    // Treats only commands for double points
    IF CmdReceive.sCommand.sSelectParameters.sValue.eParamType =
     DOUBLE_POINT_COMMAND THEN
      // Returns command finished with success
      // (controlled by IEC104 protocol)
      byResult:= 7;
    ELSE
      // Returns command not supported
      byResult:= 1;
    END_IF

  COMMAND_TYPE.OPERATE:

    // Treats only commands for double points
    IF CmdReceive.sCommand.sOperateParameters.sValue.eParamType =
     DOUBLE_POINT_COMMAND THEN
      // Pulse generation in outputs
      IF CmdReceive.sCommand.sOperateParameters.sValue.sDoublePoint.bValue THEN
        // Executes TRIP function
        fbPulsedCmd(
          byCmdType:= 101,
          byPulseTime:= DWORD_TO_BYTE(CmdReceive.sCommand.sOperateParameters.
      sValue.sDoublePoint.sPulseConfig.dwOnDuration/10),
          ptDbpVarAdr:= ADR(dbpIEC104),
          stQuality:= IOQualities.QUALITY_NX2020[4],
          byStatus=> byResult);
      ELSE
        // Executes CLOSE function
        fbPulsedCmd(
          byCmdType:= 102,
          byPulseTime:= DWORD_TO_BYTE(CmdReceive.sCommand.sOperateParameters.
      sValue.sDoublePoint.sPulseConfig.dwOffDuration/10),
          ptDbpVarAdr:= ADR(dbpIEC104),
          stQuality:= IOQualities.QUALITY_NX2020[5],
          byStatus=> byResult);
      END_IF
    ELSE
      // Returns command not supported
      byResult:= 1;
    END_IF

  COMMAND_TYPE.CANCEL:

    // Returns command finished with success
    // (controlled by IEC104 protocol)
    byResult:= 7;

  END_CASE

  // Treats the pulsed command function result
  // and generates the answer to the intercepted command
  CASE byResult OF
```

```
1:  // Invalid type of command
  CmdReceive.eCommandResult:= COMMAND_RESULT.NOT_SUPPORTED;
  CmdReceive.bDone:= TRUE;
2:  // Invalid input parameters
  CmdReceive.eCommandResult:= COMMAND_RESULT.INCONSISTENT_PARAMETERS;
  CmdReceive.bDone:= TRUE;
3:  // Parameter change in running
  CmdReceive.eCommandResult:= COMMAND_RESULT.PARAMETER_CHANGE_IN_EXECUTION;
  CmdReceive.bDone:= TRUE;
4:  // Module did not answered the command(absent)
  CmdReceive.eCommandResult:= COMMAND_RESULT.HARDWARE_ERROR;
  CmdReceive.bDone:= TRUE;
5:  // Command started and in running (does not returns nothing)
6:  // Another command has been sent to this point and it is running
  CmdReceive.eCommandResult:= COMMAND_RESULT.LOCKED_BY_OTHER_CLIENT;
  CmdReceive.bDone:= TRUE;
7:  // Command finished with success
  CmdReceive.eCommandResult:= COMMAND_RESULT.SUCCESS;
  CmdReceive.bDone:= TRUE;
END_CASE

END_IF

CmdReceive();

IF CmdReceive.bDone THEN
CmdReceive.bDone:= FALSE;
END_IF
```

As can be observed in the previous code, to help in the pulse generation in Nexto's digital double outputs, it was created and used a function block equivalent to *PulsedCommand* function of library *LibRtuStandard*. The *PulsedCommandNexto()* function block shows up coded in ST language.

```
FUNCTION_BLOCK PulsedCommandNexto
VAR_INPUT
byCmdType:    BYTE;     // command type:
            // 100 = status
            // 101 = close/on
            // 102 = trip/off
byPulseTime:  BYTE;     // Pulse duration (in hundredths of second)
ptDbpVarAdr:  POINTER TO DBP; // DBP variable address (can be mapped)
stQuality:    QUALITY;    // DBP point quality(digital module)
END_VAR
VAR_OUTPUT
bON:      BOOL;   // Odd output mapped on Nexto DO module
bOFF:     BOOL;   // Even output mapped on Nexto DO module
byStatus:   BYTE:= 7; // Function return:
          // 1 = invalid command
          // 2 = Time out of valid range (2..255)
          // 3 = command changed in running time
          // 4 = module did not answer to the command (absent)
          // 5 = command started or running
          // 6 = There is already an active command on this point
          // 7 = pulse command finished with success
```

```
END_VAR
VAR
byState:    BYTE;   // Function block state
udiPulseEnd:  UDINT;  // Pulse end instant
END_VAR

// PulsedCommandNexto state machine
CASE byState OF

0: // Init state, ready to receive commands:
CASE byCmdType OF

  100:// Just returns the last status

  101:  // Execute pulse ON:
    // Valids the pulse duration
    IF byPulseTime > 1 THEN
      // Check if there is already an active command on this point
      IF ptDbpVarAdr^.ON OR ptDbpVarAdr^.OFF THEN
        // Returns that there is already an active command
        byStatus:= 6;
      ELSE
        // Enables CLOSE output
        ptDbpVarAdr^.ON:= TRUE;
        ptDbpVarAdr^.OFF:= FALSE;
        // Next state: execute pulse ON
        byState:= byCmdType;
        // Returns started command
        byStatus:= 5;
      END_IF
    ELSE
      // Returns the out of range pulse
      byStatus:= 2;
    END_IF

  102:  // Execute pulse OFF
    // Valids the pulse duration
    IF byPulseTime > 1 THEN
      // Check if there is already an active command on this point
      IF ptDbpVarAdr^.ON OR ptDbpVarAdr^.OFF THEN
        // Returns that there is already an active
        byStatus:= 6;
      ELSE
        // Enables TRIP output
        ptDbpVarAdr^.ON:= FALSE;
        ptDbpVarAdr^.OFF:= TRUE;
        // Next step: execute pulse OFF
        byState:= byCmdType;
        // Returns started command
        byStatus:= 5;
      END_IF
    ELSE
      // Returns the out of range pulse
      byStatus:= 2;
    END_IF
ELSE
```

```
    // Returns invalid command
    byStatus:= 1;
  END_CASE

  // Memorizes the instant of the pulse end
  udiPulseEnd:= SysTimeGetMs() + BYTE_TO_UDINT(byPulseTime) * 10;

  101, 102:// Continues the pulse execution ON/OFF
  // It returns that the command is running
  byStatus:= 5;
  // Checks the running parameter change
  IF byCmdType <> 100 AND byCmdType <> byState THEN
    // Returns the running parameter change
    byStatus:= 3;
  END_IF
  // Checks pulse end
  IF SysTimeGetMs() >= udiPulseEnd THEN
    // Disable TRIP and CLOSE outputs
    ptDbpVarAdr^.ON:= FALSE;
    ptDbpVarAdr^.OFF:= FALSE;
    // Returns finished command, only if the command has not changed
    IF byCmdType = 100 OR byCmdType = byState THEN
      byStatus:= 7;
    END_IF
    // Next state: initial
    byState:= 0;
  END_IF

  END_CASE

  // Checks digital module (DBP point) quality
  IF stQuality.VALIDITY <> QUALITY_VALIDITY.VALIDITY_GOOD THEN
  // Disable TRIP and CLOSE outputs
  ptDbpVarAdr^.ON:= FALSE;
  ptDbpVarAdr^.OFF:= FALSE;
  // Returns absent module
  byStatus:= 4;
  // Next state: initial
  byState:= 0;
  END_IF

  // Copy DBP output states to the simple outputs
  bON:= ptDbpVarAdr^.ON;
  bOFF:= ptDbpVarAdr^.OFF;
```

5.9.14.3. General Parameters

To the *General Parameters* configuration of an IEC 60870-5-104 Server according to figure below follow the table below parameters:

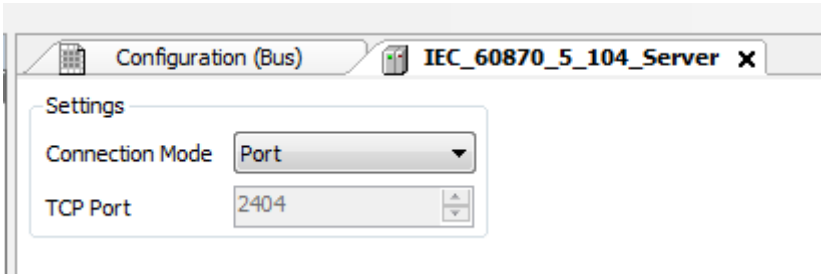


Figure 135: Server IEC 60870-5-104 General Parameters Screen

Parameter	Description	Factory Default	Possibilities
Connection Mode	Set the connection mode with the Connected Client modules.	Port	Port IP
TCP Port	Defines which PLC's TCP port number will be used to communicate with the Connected Client modules. In case the "Connection Mode" field is set as "IP".	2404	1 to 65535

Table 137: IEC 60870-5-104 Server General Parameters Configuration

5.9.14.4. Data Mapping

To configure the IEC 60870-5-104 Server data relation, viewed on figure below follow the parameters described on table below:

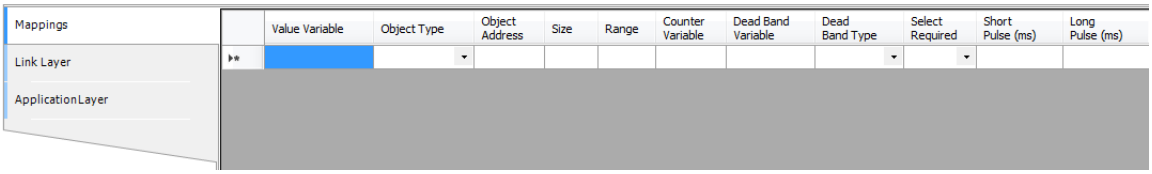


Figure 136: IEC 60870-5-104 Server Mappings Screen

Parameter	Description	Factory Default	Possibilities
Value Variable	Symbolic variable name	-	Name of a variable declared in a POU or GVL

Parameter	Description	Factory Default	Possibilities
Object Type	IEC 60870-5-104 object type configuration	-	Single Point Information Double Point Information Step Position Information Measured Value (Normalized) Measured Value (Scaled) Measured Value (Short Floating Point) Integrated Totals Bitstring Information (32 Bits) Single Command Double Command Regulating Step Command Setting Point Command (Normalized) Setting Point Command (Scaled) Setting Point Command (Short Floating Point) Bitstring Command (32 Bits)
Object Address	IEC 60870-5-104 mapping first point's index	-	1 to 65535
Size	Specifies the maximum data quantity that an IEC 60870-5-104 mapping will can access	-	1 to 86400000
Range	Configured data address range	-	-
Counter Variable	Name of the symbolic variable which will hold the counter variable's value	-	Name of a variable declared in a POU, GVL or counter module
Dead Band Variable	Name of the symbolic variable which will hold the dead band's value	-	Name of a variable declared in a POU or GVL
Dead Band Type	Defines the dead band type to be used in the mapping	Disabled	Absolute Disabled Integrated
Select Required	Defines if it is required a previous select to run a command	False	True False
Short Pulse (ms)	Defines the short pulse time to an IEC 60870-5-104 digital command	1000	1 to 86400000
Long Pulse (ms)	Defines the long pulse time to an IEC 60870-5-104 digital command	2000	1 to 86400000

Table 138: IEC 60870-5-104 Server Mappings Configuration

Notes:

Value Variable: When a read command is sent, the return received in the answer is stored in this variable. When it is a write command, the written value is going to be stored in that variable. The variable can be simple, array, array element or can be at structures.

Counter Variable: This field applies only on mapping of *Integrated Totals* type objects, being this the controller variable to be managed on process. It must has same type and size of the variable declared on *Value Variable* column, which value is going to be read, or reported to, the client in case of events.

ATTENTION

When the *Counter Variable* has a quality variable associated, to the quality to be transferred to the frozen variable at freeze command, it must be associated a quality variable to the frozen one. This procedure must be done through *Internal Points* tab.

Dead Band Variable: This field applies only to input analog variables (*Measured Value* type objects) mappings. It must has same type and size of the variable declared on *Value Variable* column. New dead band variable values are going to be considered only when the input analog variable change its value.

Dead Band Type: The configuration types available to dead band are:

Function type	Configuration	Description
Dead Band Type	Disabled	In this option, any value change in a group's point, as smaller it is, generates an event to this point.
	Absolute	In this option, if the group's point value absolute change is bigger than the value in "Dead Band" field, an event is going to be generated to this point.
	Integrated	In this option, if the absolute of the integration of the group's point value change is bigger than the value in "Dead Band" field, an event is going to be generated to this point. The integration interval is one second.

Table 139: IEC 60870-5-104 Server Mappings Dead Band Types

Short Pulse and Long Pulse: At the define of short and long pulses duration time it must be considered the limits supported by the device which will treat the command. For example, case the destiny is an output card, which is not supported in native by Nexto Series. It must be checked at the module's Datasheet what the minimum and maximum times, as well as the resolution, to running the pulsed commands.

5.9.14.5. Link Layer

To the IEC 60870-5-104 Server link layer parameters configuration, shown on figure below, follow the described parameters on table below:

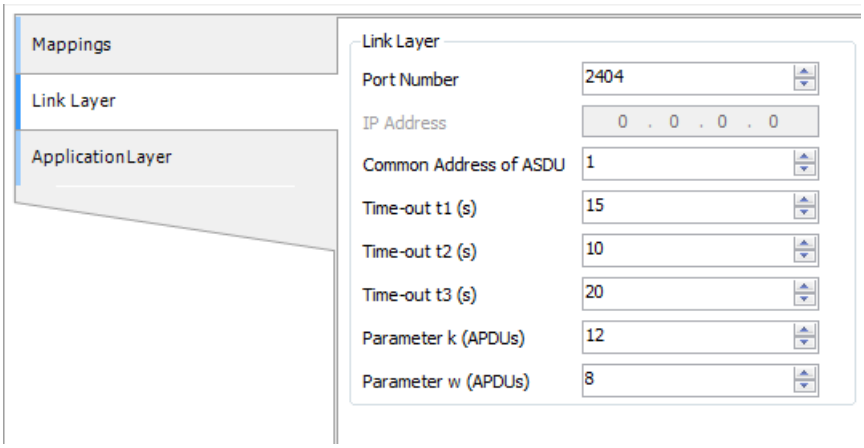


Figure 137: Server IEC 60870-5-104 Link Layer Configuration Screen

Parameter	Description	Factory Default	Possibilities
Port Number	Listened port address to client connection. Used when the client connection isn't through IP	2404	1 to 65535
IP Address	Connected client IP, used when the client connection is through IP	0.0.0.0	1.0.0.1 to 223.255.255.254
Common Address of ASDU	IEC 60870-5-104 address, if the connected client is through IP	1	1 to 65534
Time-out t1 (s)	Time period (in seconds) that the device waits the receiving of an acknowledge message after sent an APDU message type I or U (data), before close the connection	15	1 to 180
Time-out t2 (s)	Time period (in seconds) that the device waits to send a watch message (S-Frame) acknowledging the data frame receiving	10	1 to 180
Time-out t3 (s)	Time period (in seconds) in what is going to be sent a message to link test in case there is no transmission by both sides	20	1 to 180
Parameter k (APDUs)	Maximum number of data messages (I-Frame) transmitted and not acknowledged	12	1 to 12
Parameter w (APDUs)	Maximum number of data messages (I-Frame) received and not acknowledged	8	1 to 8

Table 140: IEC 60870-5-104 Server Link Layer Configuration

Note:

The fields *Time-out t1 (s)*, *Time-out t2 (s)* and *Time-out t3 (s)* are dependents between themselves and must be configured in a way that *Time-out t1 (s)* be bigger than *Time-out t2 (s)* and *Time-out t3 (s)* be bigger than *Time-out t1 (s)*. If any of these rules be not respected, error messages are going to be generated during the project compilation.

ATTENTION

For slow communication links (example: satellite communication), the parameters *Time-out t1 (s)*, *Time-out t2 (s)* and *Time-out t3 (s)* must be properly adjusted, such as doubling the default values of these fields.

5.9.14.6. Application Layer

To configure the IEC 60870-5-104 Server application layer, shown on figure below, follow the parameters described on table below:

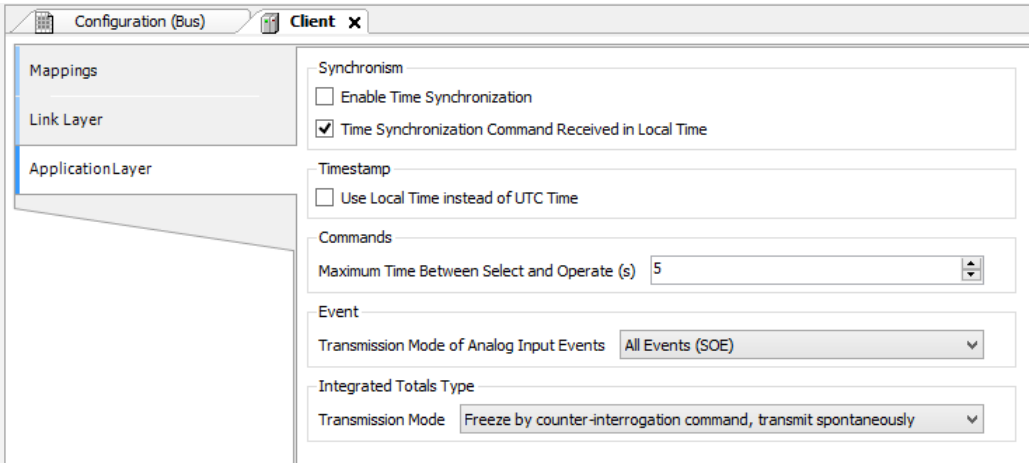


Figure 138: Server IEC 60870-5-104 Application Layer Configuration Screen

Parameter	Description	Factory Default	Possibilities
Enable Time Synchronization	Option to Enable/Disable time sync request	Disabled	Disabled Enabled
Time Synchronization Command Received in Local Time	Option to Enable/Disable the treatment of the synchronization command in local time	Enabled	Disabled Enabled
Use Local Time instead of UTC Time	Option to Enable/Disable the time stamp in local time for events	Disabled	Disabled Enabled
Maximum Time Between Select and Operate (s)	Time period in which the selection command will remain active (the count starts from the received selection command acknowledge) waiting the Operate command	5	1 to 180

Parameter	Description	Factory Default	Possibilities
Transmission Mode of Analog Input Events	Analog input events transmission mode	All Events (SOE)	All Events (SOE) Most Recent Event
Transmission Mode	Frozen counters transmission mode (Integrated Totals)	Freeze by counter-interrogation command, transmit spontaneously	Freeze by counter-interrogation command, transmit spontaneously Freeze and transmit by counter-interrogation command

Table 141: IEC 60870-5-104 Server Application Layer Configuration

Notes:

Enable Time Synchronization: Once enabled, allow the IEC 60870-5-104 Server adjust the CPU's clock when a sync command is received.

Time Synchronization Command Received in Local Time: When enabled, the IEC 60870-5-104 Server adjusts the CPU clock by treating the time received in the synchronization command as local time. Otherwise, this time is considered UTC.

Use Local Time instead of UTC Time: Once enabled, the time stamp of the events generated by IEC 60870-5-104 Server will be sent according to the CPU's local time.

ATTENTION

When the time sync option is checked in more than one server, the received times from different servers will be overwritten in the system clock in a short time period, being able to cause undesirable behaviors due to delays on messages propagation time and system load.

Transmission Mode of Analog Inputs Events: The Analog Inputs Events transmission modes available are the following:

Function Type	Configuration	Description
Transmission Mode of Analog Input Events	All Events (SOE)	All analog events generated are going to be sent.
	Most Recent Event	It is sent only the most recent analog event.

Table 142: IEC 60870-5-104 Server Transmission Modes of Analog Inputs Events

Transmission Mode: The available transmission modes of the frozen counters (*Integrated Totals*) are the following:

Function Type	Configuration	Description
Transmission Mode	Freeze by counter-interrogation command, transmit spontaneously	Equivalent to the counters acquisition D Mode (Integrated Totals) defined by Standard IEC 60870-5-101. In this mode, the control station's counters interrogation commands, freeze the counters. Case the frozen values have been modified, they are reported through events.
	Freeze and transmit by counter-interrogation command	Equivalent to the counters acquisition C Mode (Integrated Totals) defined by Standard IEC 60870-5-101. In this mode, the control station's counters interrogation commands, freeze the counters. The subsequent counters interrogation commands (read) are sent by the control station to receive the frozen values.

Table 143: IEC 60870-5-104 Server Transmission Modes of the Frozen Counters

ATTENTION

The Standard IEC 60870-5-104, section *Transmission control using Start/Stop*, foresee the commands *STARTDT* and *STOPDT* utilization to data traffic control between client and server, using simple or multiple connections. Despite Nexto supports such commands, its utilization isn't recommended to control data transmission, mainly with redundant CPUs, because such commands aren't synchronized between both CPUs. Instead of using multiple connections between client and Nexto server, it's suggested the use of NIC Teaming resources to supply (physically) redundant Ethernet channels and preserve the CPU resources (CPU control centers).

5.9.14.7. Server Diagnostic

The IEC 60870-5-104 Server protocol diagnostics are stored in *T_DIAG_IEC104_SERVER_1* type variables, which are described in table below:

Diagnostic variable of type T_DIAG_IEC104_SERVER_1.*	Size	Description
Command bits, automatically reset:		
tCommand.bStop	BOOL	Disable Driver
tCommand.bStart	BOOL	Enable Driver
tCommand.bDiag_01_Reserved	BOOL	Reserved
tCommand.bDiag_02_Reserved	BOOL	Reserved
tCommand.bDiag_03_Reserved	BOOL	Reserved
tCommand.bDiag_04_Reserved	BOOL	Reserved
tCommand.bDiag_05_Reserved	BOOL	Reserved
tCommand.bDiag_06_Reserved	BOOL	Reserved
Diagnostics:		
tClient_X.bRunning	BOOL	IEC 60870-5-104 Server is running

Diagnostic variable of type T_DIAG_IEC104_SERVER_1.*	Size	Description
tClient_X.eConnectionStatus. CLOSED	ENUM(BYTE)	Communication channel closed. Server won't accept connection request. ENUM value (0)
tClient_X.eConnectionStatus. LISTENING		Server is listening to the configured port and there is no connected clients. ENUM value (1)
tClient_X.eConnectionStatus. CONNECTED		Connected client. ENUM value (2)
tClient_X.tQueueDiags. bOverflow	BOOL	Client queue is overflowed
tClient_X.tQueueDiags. wSize	WORD	Configured queue size
tClient_X.tQueueDiags. wUsage	WORD	Events number in the queue
tClient_X.tQueueDiags. dwReserved_0	DWORD	Reserved
tClient_X.tQueueDiags. dwReserved_1	DWORD	Reserved
tClient_X.tStats.wRXFrames	WORD	Number of received frames
tClient_X.tStats.wTXFrames	WORD	Number of sent frames
tClient_X.tStats.wCommErrors	WORD	Communication errors counter, including physical layer, link layer and transport layer errors.
tClient_X.tStats.dwReserved_0	DWORD	Reserved
tClient_X.tStats.dwReserved_1	DWORD	Reserved

Table 144: IEC 60870-5-104 Server Diagnostics

5.9.14.8. Commands Qualifier

The standard IEC 60870-5-104 foresee four different command qualifiers for the objects *Single Command*, *Double Command* and *Regulating Step Command*, all supported by the Nexto Server.

Each object type has a specific behavior to each command qualifier, as can be seen on the table below.

Qualifier	Protocol IEC 60870-5-104 object type		
	Single Command	**Double Command**	**Regulating Step Command**
No additional definition (default)	Same behavior of persistent qualifier.	Same behavior of short pulse qualifier.	Same behavior of short pulse qualifier.
Short pulse duration	Requires command interception to application treatment. Other way it will return a negative acknowledge message (fail).	Requires command interception to application treatment. Other way it will return a negative acknowledge message (fail).	Requires command interception to application treatment. Other way it will return a negative acknowledge message (fail).
Long pulse duration			

Qualifier	Protocol IEC 60870-5-104 object type		
	Single Command	**Double Command**	**Regulating Step Command**
Persistent output	The output is going to be on or off and that will remain until new command, according to value (ON or OFF) commanded by the client.		

Table 145: IEC 60870-5-104 Server Commands Qualifier

Note:

Command Interception: For further information about commands interception of IEC 608705-104 clients, consult section Interception of Commands Coming from the Control Center, implemented through *CommandReceiver* function block.

5.9.15. 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 IEC XE.

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 lenght
1000 kbit/s	25 m
800 kbit/s	50 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 146: Baudrate vs Bus Lenght

5.9.15.1. Installing and inserting CANopen Devices

The configuration of a CANopen network uses the same standard procedure of other fieldbuses configuration on MasterTool IEC XE.

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:

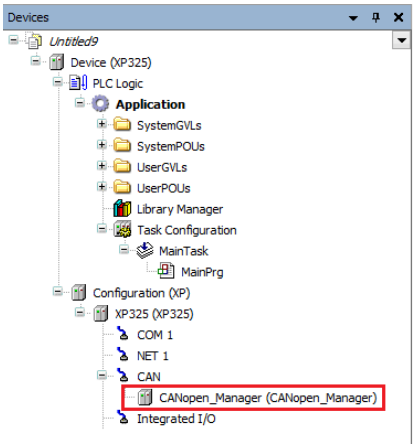


Figure 139: Adding CANopen Manager

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:

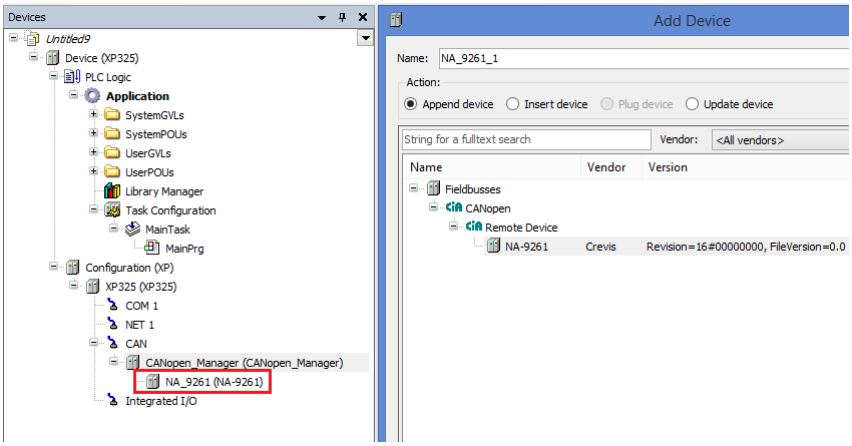


Figure 140: Adding CANopen Slave Device

5.9.15.2. 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:

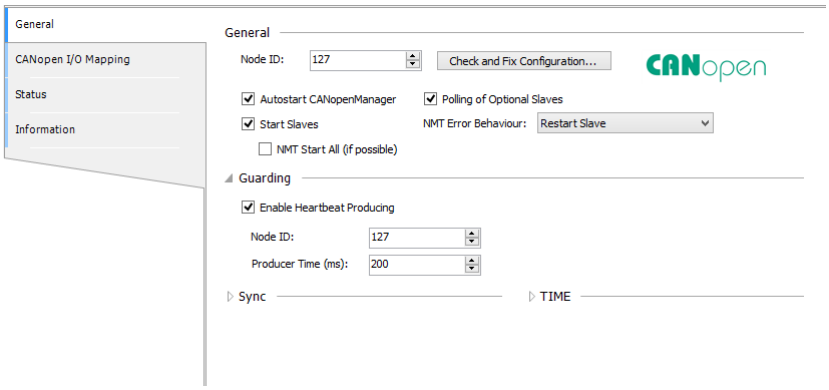


Figure 141: CANopen Manager general parameters

The detailed description of CANopen Manager general parameters can be found on section *Device Editors -> CANopen* of MasterTool IEC XE Online Help (F1).

Additionally, the tab *CANopen I/O Mapping* allows to change the bus cycle task:

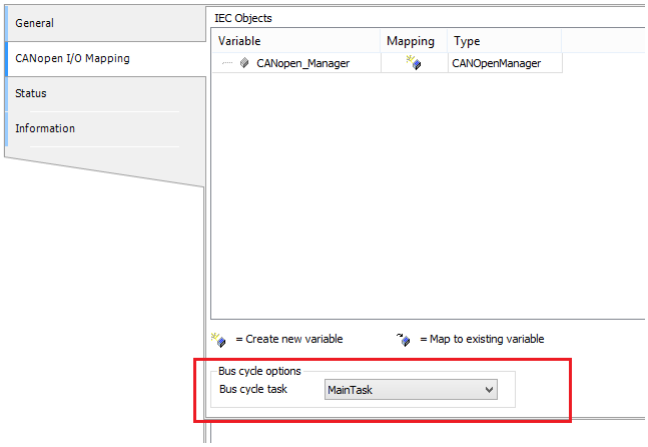


Figure 142: CANopen Manager bus cycle task setting

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.

5.9.15.3. CANopen Slave Configuration

The configuration of CANopen Remote Devices (Slaves) is separated in the first four tabs shown on the following picture:

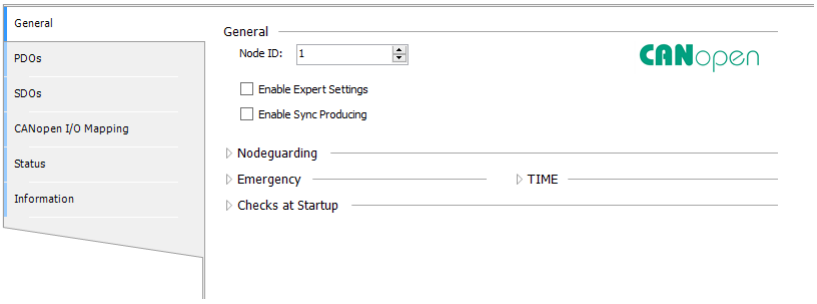


Figure 143: CANopen Slave parameters

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 IEC XE Online Help (F1).

5.10. Remote I/O Mode

Nexto Xpress controllers have a remote operation mode, which is used as I/O expansion. This expansion is based in CANopen protocol. When the controller is in remote mode, it isn't a standard PLC, operating only as a remote slave. To configure your Xpress as a remote I/O expansion, access the *Operation Mode* section, in the *Management* tab of the Controller's System Web Page.

Figure 144: Remote I/O Configuration Screen

In this tab, it's possible to choose the controller operation mode through the *Operation Mode* parameter. This is available only when the controller is in STOP. Use the *Apply* button to change to the desired mode. The Xpress will reboot and configure the new operation mode. The available options are:

- **Programmable Controller:** default controller function, which can be programmed according to user needs.
- **CANopen Slave:** remote I/O expansion function, where the controller becomes a CANopen slave, which can communicate to other controllers through CANopen Manager.
- **EtherNet/IP Adapter:** remote I/O expansion function, in which the controller becomes an EtherNet/IP Adapter, and can be controlled by other controllers with EtherNet/IP Scanner.

ATTENTION

The remote operation mode uses an application developed only for I/O expansion purposes. For the correct operation of the controller in CANopen slave mode, the minimum bus cycle of the CAN network must be higher than 10 ms. When operating in EtherNet/IP Adapter mode, the minimum RPI must be 20 ms. These settings cannot be changed, nor is it possible to load a new application during execution.

When in remote operation mode, some features of the controller will be modified. The controller can't be found by the MasterTool. However, it's possible to find the device via *Easy Connection*, even change its IP. Besides that, in the *Firmware Update* tab on the Web page, the *Erase Application* option is unavailable.

5.10.1. CANopen Slave

To use the expansion mode as a CANopen Slave, first, change the *Operation Mode* to *CANopen Slave*, in *Operation Mode* on the Web page. Next, make the configurations of the *CANopen Slave*: configure the network (*IP Address*, *Network Mask* and *Gateway*); configure the CANopen parameters (*Node ID*, *Baudrate* and *Termination*); and the I/O configuration (according to the controller). These settings are similar to a typical application.

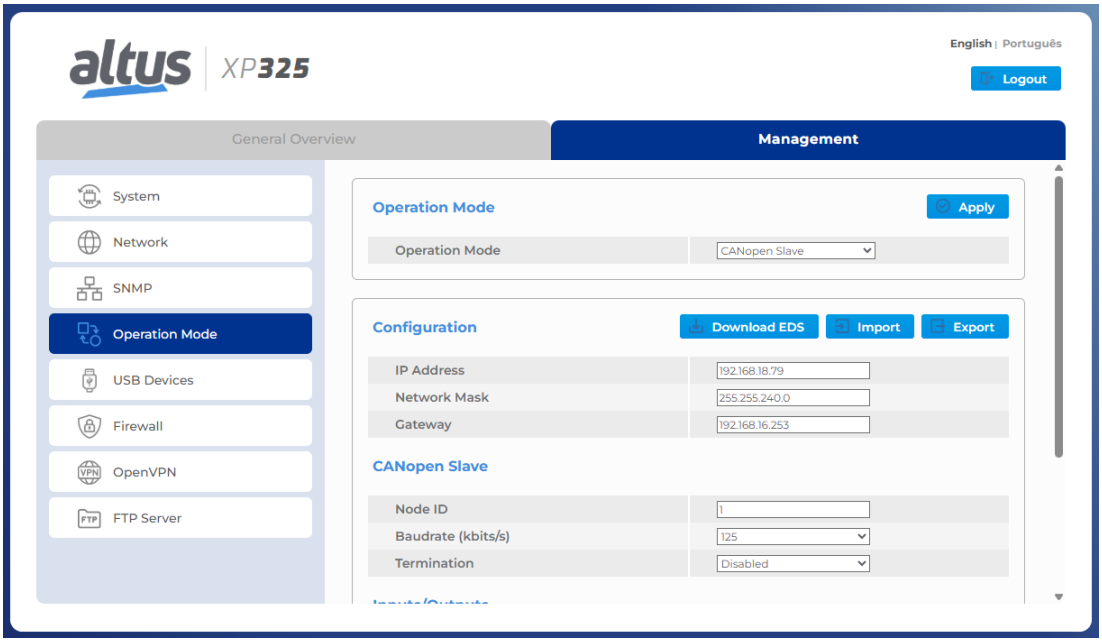


Figure 145: CANopen Slave Remote Configuration Screen

The configuration of inputs and outputs in CANopen Slave mode is done directly on the web page, in the Input and Output Configuration section. The settings are presented in the Remote I/O Mode Configuration section.

ATTENTION

PDOs can't be edited or removed from the CANopen Slave. It's not possible to create your own CANopen slave device.

How the CANopen Slave is not accessible by the user via MasterTool, the RUN and STOP state of the application are controlled by the CANopen slave operation state. To put the CANopen Slave in RUN, it's necessary to set the state to *Operational* (green symbol next to the device). To put it in STOP, you need to use the NMT Function Block of the CiA405 library - see Online Help (F1) - to change the CANopen slave operation state (recommended). Or, you can remove the CAN connector of the remote controller. The CAN LED can keep blinking once it indicates the message transmission and reception, not the operation state of the CANopen protocol.

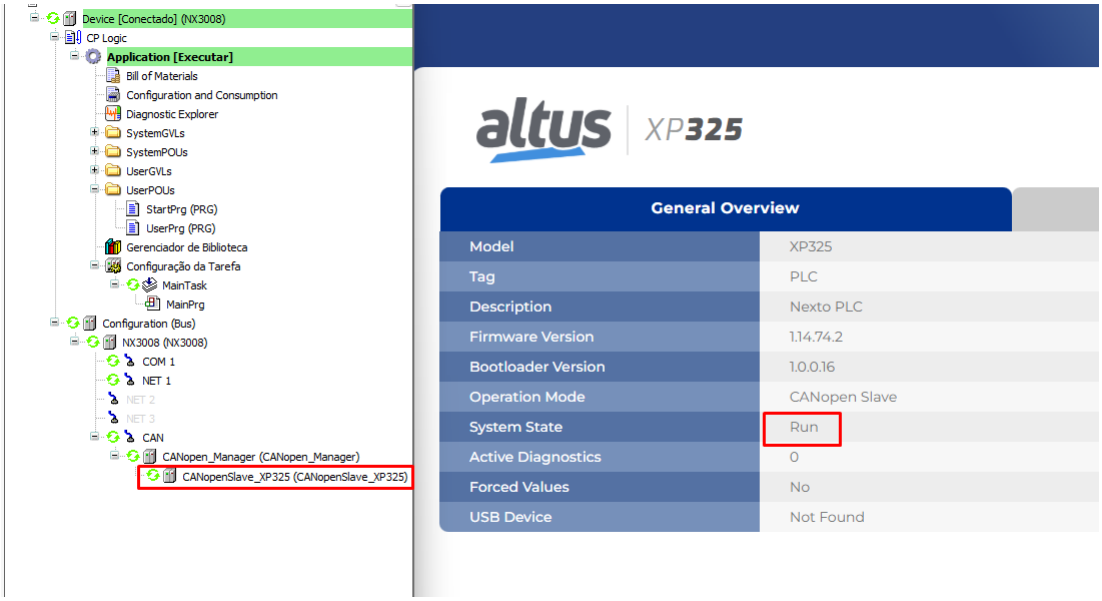


Figure 146: CANopen Slave in Operational - RUN

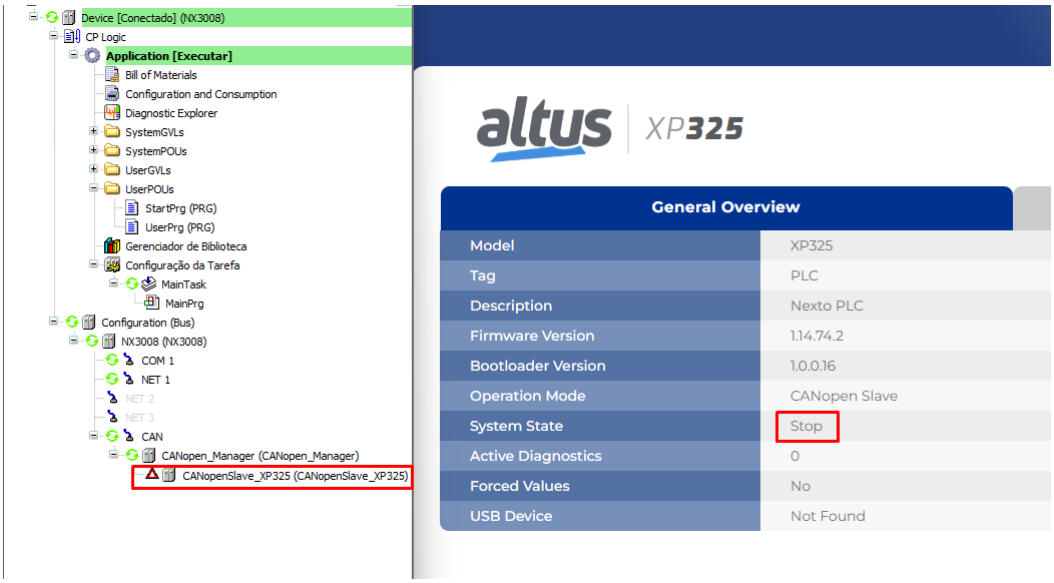


Figure 147: CANopen Slave in Pre-Operational - STOP

5.10.2. EtherNet/IP Adapter

To use expansion mode as an EtherNet/IP Adapter, first change the *Operation Mode* option to *EtherNet/IP Adapter* in the *Operation Mode* tab on the web page. Then, perform the appropriate configurations for *EtherNet/IP Adapter* mode: configure the network data (*IP Address*, *Netmask*, and *Gateway*) and the *I/O Configuration* parameters (according to the controller's availability). These configurations are similar to those performed in a traditional application.

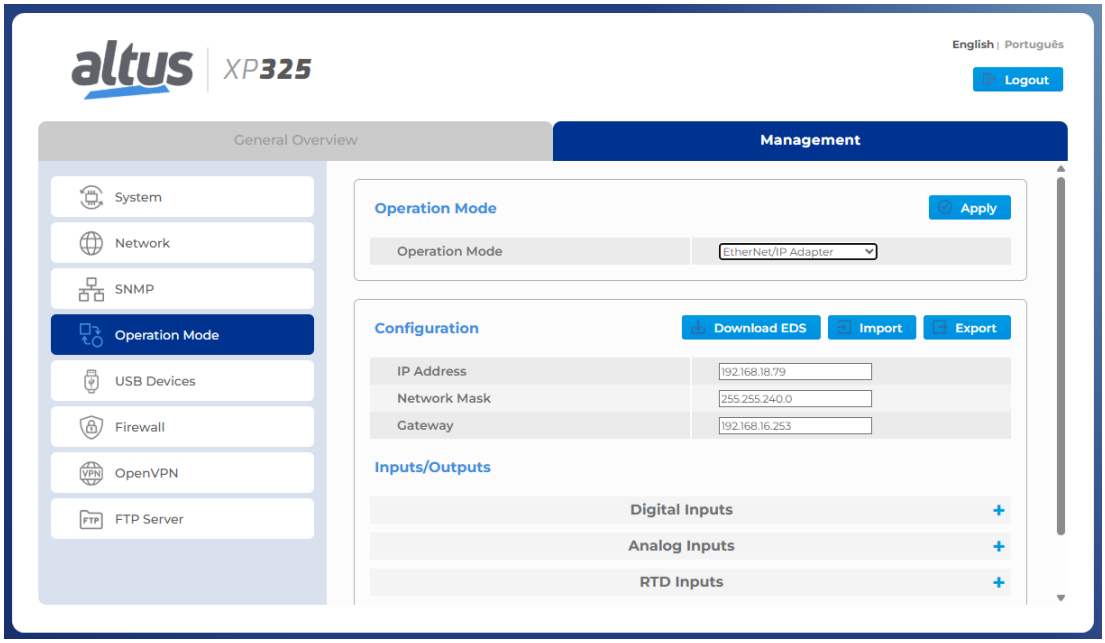


Figure 148: EtherNet/IP Adapter Configuration Screen

The configuration of inputs and outputs in EtherNet/IP Adapter mode is done directly on the web page, in the Input and Output Configuration section. The settings are presented in the Remote I/O Mode Configuration section.

ATTENTION

Data and information cannot be edited or deleted from the EtherNet/IP Remote, and the user cannot create their own EtherNet/IP Adapter.

When running the remote EtherNet/IP Adapter application it is not possible to access the controller via MasterTool. Therefore, the RUN and STOP application states are controlled by the EtherNet/IP Adapter operation state. To assume the RUN state, it's necessary to a EtherNet/IP Scanner connects to the controller. If no EtherNet/IP Scanner is connected, the controller stays in STOP state.

5.10.3. Remote I/O Mode Configuration

The input and output settings are the same for any configuration mode. They apply to CANopen slave mode and EtherNet/IP Adapter mode. To do this, click the items with a + on the right to expand the configuration window. The parameters presented in *I/O Configuration* are the same as those mentioned in the Integrated I/O Configuration section. The configuration screens for each possible input and output type are presented below.

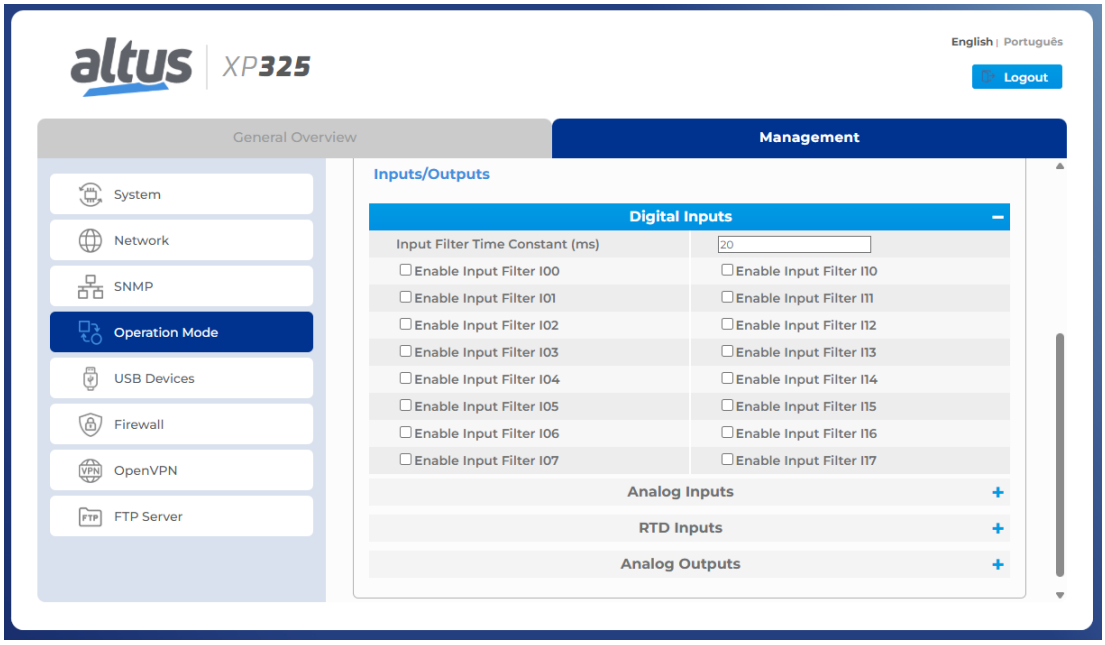


Figure 149: Expanded Digital Inputs Configuration Screen

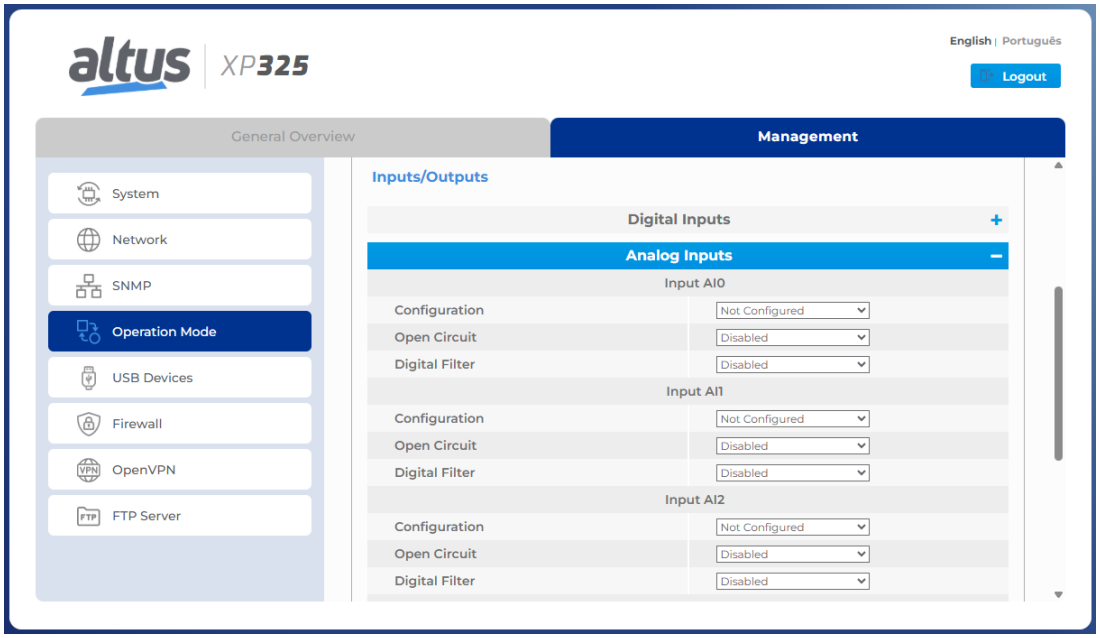


Figure 150: Expanded Analog Inputs Configuration Screen

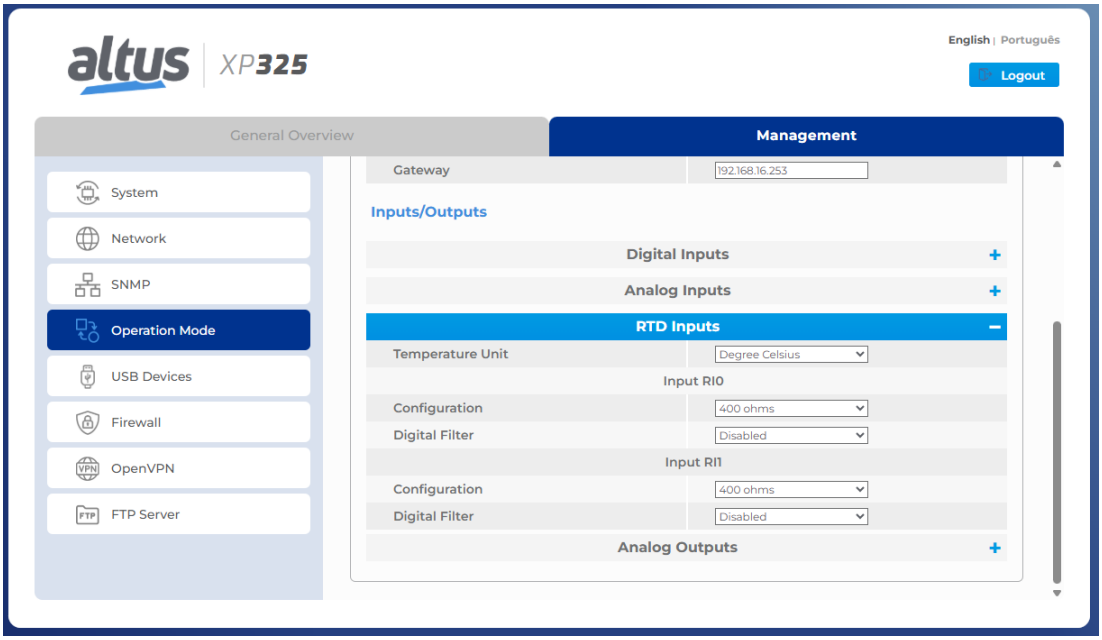


Figure 151: Expanded RTD Inputs Configuration Screen

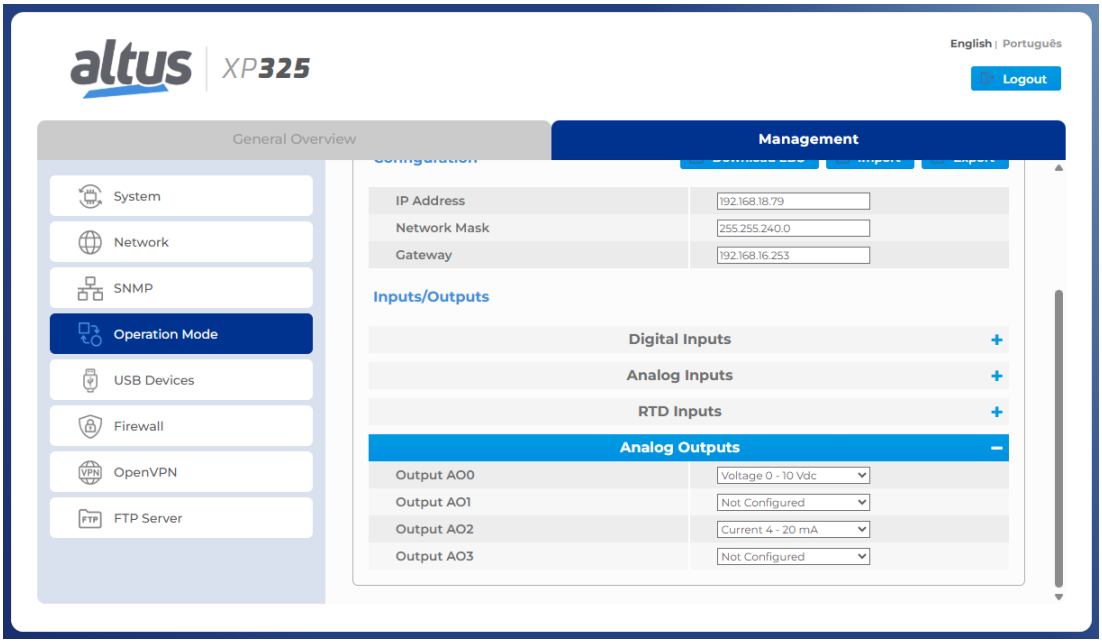


Figure 152: Expanded Analog Outputs Configuration Screen

After completing the configuration, you can use the *Export Configuration* button to download a configuration file, called *WebRemoteConfigurationCAN.config* for CANOpen Slave mode and *WebRemoteConfigurationENIP.config* for ETherNet/IP Adapter mode, containing the parameters configured on the screen. This file can then be loaded using the *Import Configuration* button. Additionally, you can download the Electronic Data Sheet (EDS) file for the respective remote operation mode directly from the web using the *Download EDS* button.

Once the configuration step is complete, click *Apply Configuration*, which will restart the controller. The web page will automatically update to the configured IP address. In the *PLC Information* tab, you can check the PLC's operating mode, confirming the change to the selected operating mode.

Figure 153: Operation Mode in the PLC Overview Screen

This allows you to use a controller with CANopen Manager or EtherNet/IP Scanner functionality (for example, the XP340) to access the PLC's I/O points in CANopen Slave or EtherNet/IP Adapter mode. To do this, perform the same steps described in the CANOpen Manager section of this document when selecting CANopen Slave mode, or the steps described in the EtherNet/IP Scanner Configuration section for EtherNet/IP Adapter mode. The data structure containing the input and output information available on the PLC is organized according to the table below:

Variable Name	Representation	Variable Type
Digital_Outputs_1	Q0 Group	USINT - 8 bits
Digital_Outputs_2	Q1 Group	USINT - 8 bits
Analog_Outputs_1	AO0	INT - 16 bits
Analog_Outputs_2	AO1	INT - 16 bits
Analog_Outputs_3	AO2	INT - 16 bits
Analog_Outputs_4	AO3	INT - 16 bits
Digital_Inputs_1	I0 Group	USINT - 8 bits
Digital_Inputs_2	I1 Group	USINT - 8 bits
Analog_Inputs_1	AI0	INT - 16 bits
Analog_Inputs_2	AI1	INT - 16 bits
Analog_Inputs_3	AI2	INT - 16 bits
Analog_Inputs_4	AI3	INT - 16 bits
Analog_Inputs_5	AI4	INT - 16 bits
RTD_Inputs_1	RI0	INT - 16 bits
RTD_Inputs_2	RI1	INT - 16 bits
Diagnostics_Analog_Inputs_1	AI0 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Inputs_2	AI1 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Inputs_3	AI2 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Inputs_4	AI3 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Inputs_5	AI4 Diagnostics	USINT - 8 bits
Diagnostics_RTD_Inputs_1	RI0 Diagnostics	USINT - 8 bits
Diagnostics_RTD_Inputs_2	RI1 Diagnostics	USINT - 8 bits

Variable Name	Representation	Variable Type
Diagnostics_Analog_Outputs_1	AO0 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Outputs_2	AO1 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Outputs_3	AO2 Diagnostics	USINT - 8 bits
Diagnostics_Analog_Outputs_4	AO3 Diagnostics	USINT - 8 bits

Table 147: CANopen Slave Remote PDOs Organization

Digital I/O is accessed in groups via a single-byte variable, where each bit represents a digital input or output. For example, I00 is the least significant bit and I07 is the most significant. Analog I/O is transmitted/received directly via an integer. Diagnostics for each analog I/O is received in a single byte, as shown in the tables below.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	bOpenLoop	bOverRange	bInputNotEnable

Table 148: AIx Diagnostics

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	bUnderRange	bOverRange	bInputNotEnable

Table 149: RIx Diagnostics

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	bShortCircuit	bOpenLoop	bOutputNotEnable

Table 150: AOx Diagnostics

5.10.4. PROFINET Controller

For correct use of the PROFINET Controller protocol, it is necessary to consult the manual MU214621 - Nexto Series PROFINET Manual .

5.11. Communication Performance

5.11.1. 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.

5.11.1.1. CPU's Integrated 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
4 Connections	760	395	200
7 Connections	1354	695	350
10 Connections	1933	976	500

Table 151: 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.

5.11.2. 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.

5.12. 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 MasterTool IEC XE. 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 on Help of MasterTool IEC XE.

5.13. SNMP

5.13.1. 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 in which each object of this tree is called OID (Object Identifier). For all equipments with SNMP, it is mandatory to support MIB-II, which have key information for managing Ethernet networks.

5.13.2. SNMP in Nexto Xpress Controllers

The Nexto Xpress controllers behaves as agents in SNMP communication, with support for protocols SNMPv1, SNMPv2c, SNMPv3 and support the MIB-II, where required objects are described in RFC-1213. The information provided by the SNMP cannot be manipulated or accessed through the user application, requiring an external SNMP manager to perform access. The following table describes the objects available in Nexto Xpress controllers.

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 6 and 7 available, which can be viewed by the network interfaces statistics NET 1 and NET 2, respectively, of the Nexto Series 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 152: MIB II Objects – Nexto Series SNMP Agent

By default, the SNMP agent is activated, i.e., the service is initialized at the time the controller is started. The access to the agent information is via the Ethernet interface, TCP port 161. The following figure shows an example of an SNMP manager reading some values.

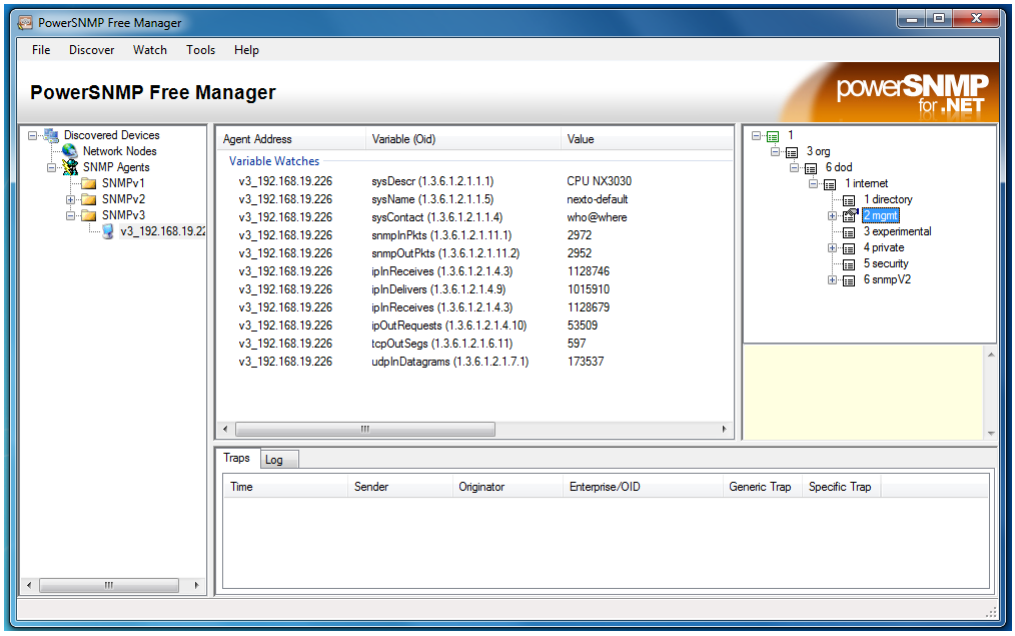


Figure 154: SNMP Manager Example

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 controller's web page as described on the following section.

5.13.3. 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.

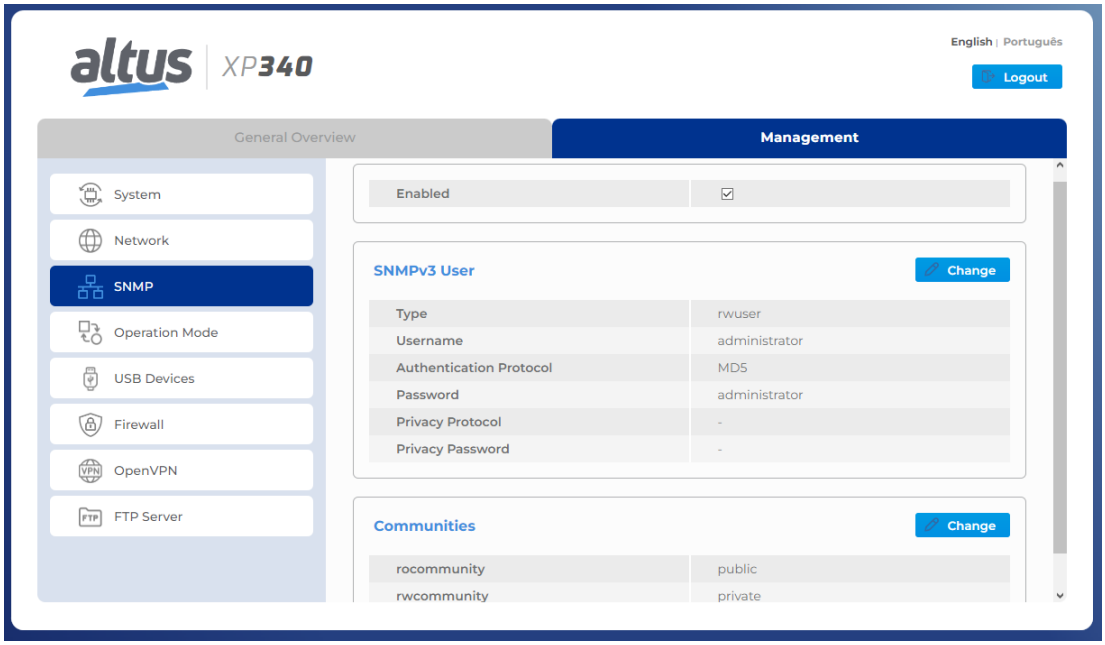


Figure 155: SNMP status configuration screen

ATTENTION

The *Username* and *Password* to access the agent via SNMP protocol are the same used to login on the SNMP Settings web page.

5.13.4. User and SNMP Communities

To access the SNMPv1 / v2c of the Nexto Xpress controllers, there are two communities, according to following table.

Communities	Default String	Type
rocommunity	Public	Only read
rwcommunity	Private	Read and Write

Table 153: SNMP v1/v2c Default Communities info

It's possible to access SNMP v3 using default user, see table below:

User	Type	Authentication Protocol	Authentication Password	Private Protocol	Private Password
administrator	rwuser	MD5	administrator	-	-

Table 154: SNMP v3 User info

For all settings of communities, 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	-	30	[0-9][a-z][A-Z]@$*_.
Password v3	8	30	[0-9][a-z][A-Z]@$*_.

Table 155: SNMP settings limits

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:

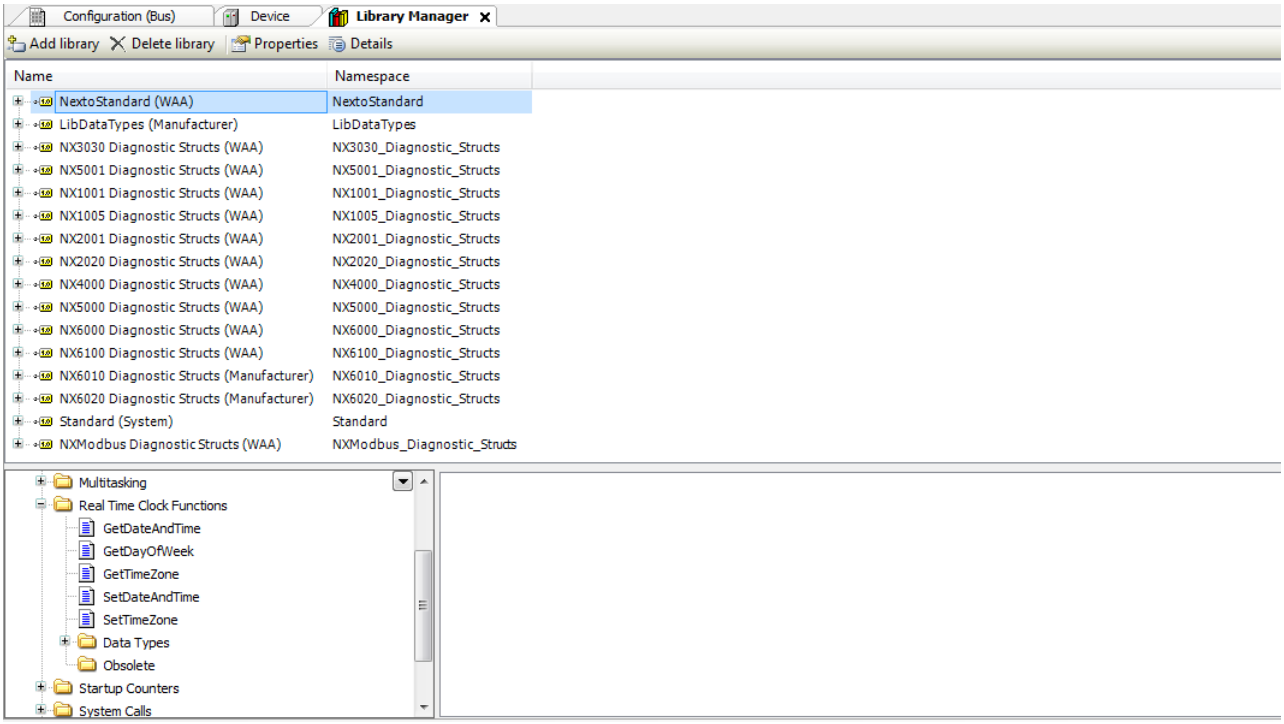


Figure 156: Clock Reading and Writing Blocks

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

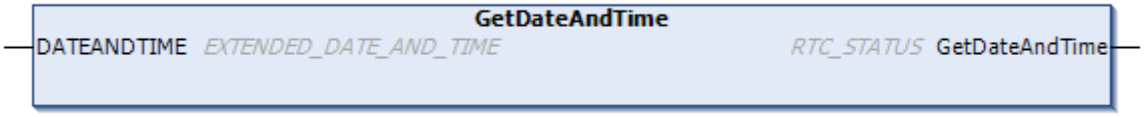


Figure 157: Date and Hour Reading

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 165.

Table 156: Input Parameters of GetDateAndTime

Output Parameters	Type	Description
GETDATEANDTIME	RTC_STATUS	Returns the function error state, see Table 167.

Table 157: 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:

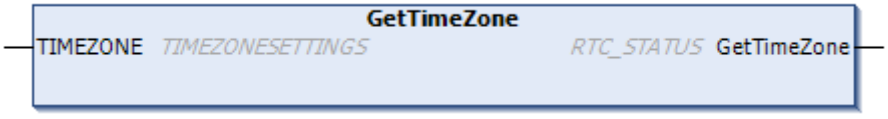


Figure 158: Configuration Reading of Time Zone

Input Parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	This variable presents the reading of Time Zone configuration.

Table 158: Input Parameters of GetTimeZone

Output Parameters	Type	Description
GetTimeZone	RTC_STATUS	Returns the function error state, see Table 167.

Table 159: 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 159: Day of Week Reading

Output Parameters	Type	Description
GetDayOfWeek	DAYS_OF_WEEK	Returns the day of the week. See Section 166.

Table 160: 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.

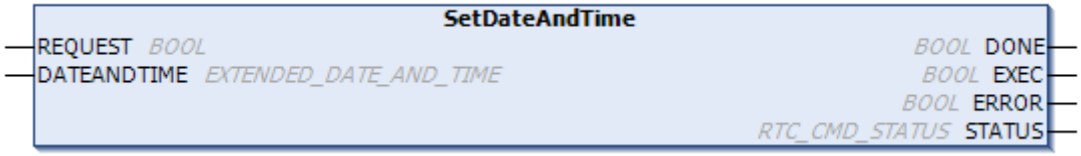


Figure 160: Set Date And Time

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 165.

Table 161: 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 167.

Table 162: 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:

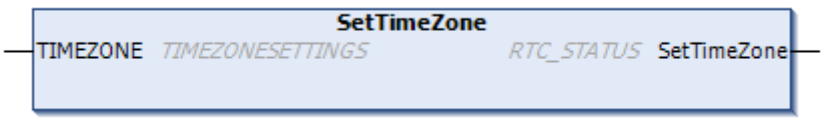


Figure 161: Writing of the Time zone Settings

Input parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	Structure with time zone to be configured. See Table 168.

Table 163: SetTimeZone Input Parameters

Output parameters	Type	Description
SetTimeZone	RTC_STATUS	Returns the error occurred during the reading/setting. See Table 167.

Table 164: 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 165: 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 166: 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 167: 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 168: 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 169 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	2035
Hour	0	23
Minute	0	59
Second	0	59
Millisecond	0	999

Table 169: 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 the MasterTool IEC XE software 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.

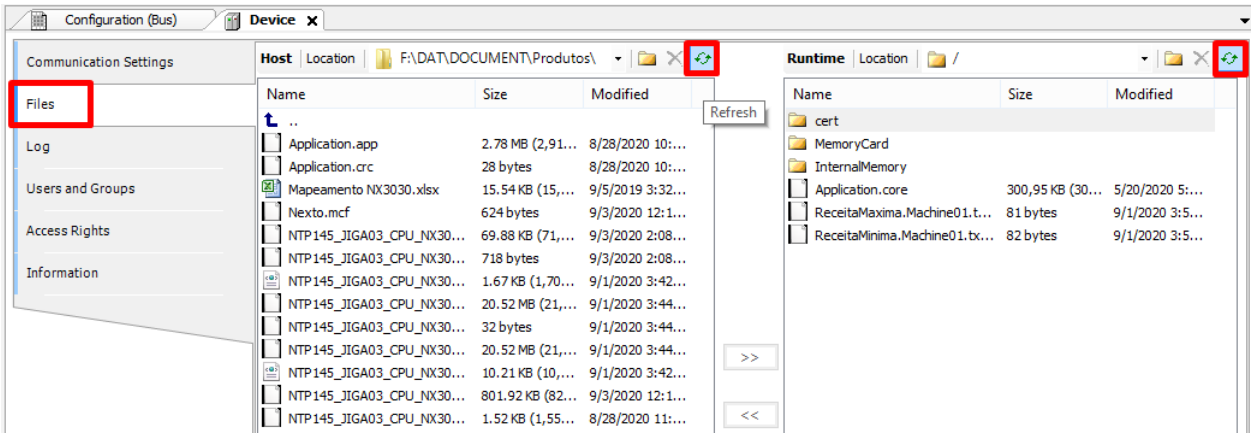


Figure 162: User Files Access

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.

ATTENTION

The files contained in the folder of a project created by MasterTool IEC XE 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 the MasterTool IEC XE software 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 the MasterTool IEC XE screen, of the files in the user memory area and in the computer.

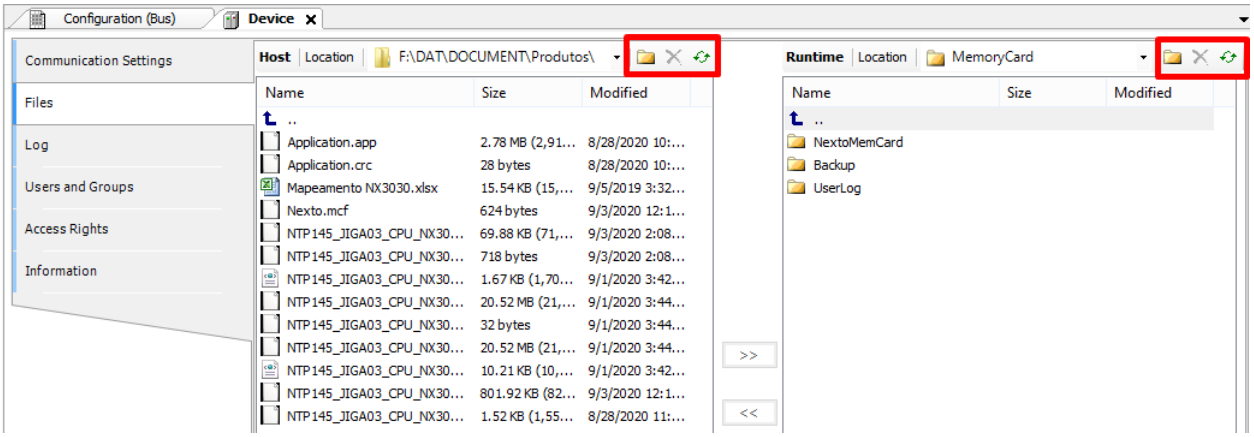


Figure 163: Utilization Options

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 the MasterTool IEC XE software. 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 MasterTool IEC XE Programming Manual – MP399609, 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	
	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.

Data type	Options	Description
	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.
	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	

Data type	Options	Description
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).
SERIAL_RX_QUEUE_ STATUS	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).
	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:	

Data type	Options	Description
SERIAL_STATUS	NO_ERROR	No errors.
	ILLEGAL_*	Return the parameters with invalid values or out of range: - SERIAL_PORT - SERIAL_MODE - BAUDRATE - DATA_BITS - PARITY - STOP_BITS - HANDSHAKE - UART_RX_THRESHOLD - TIMEOUT - TX_BUFF_LENGTH - HANDSHAKE_METHOD - 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.
	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.

Data type	Options	Description
	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 170: 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 164: 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 171: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - ILLEGAL_SERIAL_MODE - ILLEGAL_BAUDRATE - ILLEGAL_DATA_BITS - ILLEGAL_PARITY - ILLEGAL_STOP_BITS - ILLEGAL_HANDSHAKE - ILLEGAL_UART_RX_THRESHOLD - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - DCD_INTERRUPT_NOT_ALLOWED - CTS_INTERRUPT_NOT_ALLOWED - DSR_INTERRUPT_NOT_ALLOWED

Table 172: 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 := RS232_RTS,
UART_RX_THRESHOLD := 8,
MODE :=NORMAL_MODE,
ENABLE_RX_ON_TX := FALSE,
ENABLE_DCD_EVENT := FALSE,
ENABLE_CTS_EVENT := 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 165: 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 173: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - NOT_CONFIGURED
PARAMETERS	SERIAL_PARAMETERS	This structure receives the serial port configuration parameters, as described in the SERIAL_PARAMETERS data type.

Table 174: 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 166: 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 175: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - FB_GET_CTRL_NOT_ALLOWED - 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 176: 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 167: 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 177: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - 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 178: 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 168: 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 179: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - NOT_CONFIGURED

Table 180: 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 169: 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 181: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_RX_BUFF_LENGTH - RX_TIMEOUT_ERROR - FB_SERIAL_RX_NOT_ALLOWED - 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 182: 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 Configuration section.

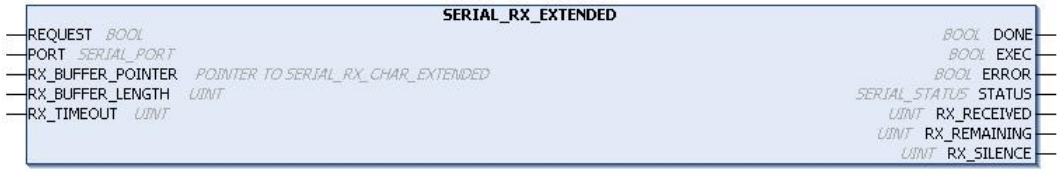


Figure 170: Block Used for Reception Buffer Reading

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 183: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_RX_BUFF_LENGTH - RX_TIMEOUT_ERROR - FB_SERIAL_RX_EXTENDED_NOT _ALLOWED - 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 μs. 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 184: 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 171: 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 185: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - FB_SET_CTRL_NOT_ALLOWED - NOT_CONFIGURED

Table 186: 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 172: 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 187: 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: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_TX_BUFF_LENGTH - ILLEGAL_TIMEOUT - CTS_TIMEOUT_ON - CTS_TIMEOUT_OFF - TX_TIMEOUT_ERROR - 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 188: 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. Inputs and Outputs Update

By default, the local bus and CPU integrated I/O are updated on every execution cycle of MainTask. The Refresh functions allows to update these I/O points asynchronously at any point of user application code.

When the function blocks to update the inputs and outputs are not used, the update is performed every cycle of the MainTask.

ATTENTION

At the startup of a CPU of this series, the inputs and outputs are only updated for reading and prepared for writing when the MainTask is performed.
All other system tasks that run before MainTask will be with the inputs and outputs invalid.

5.16.2.1. RefreshIntegratedIoInputs

This function allows to update all the inputs integrated to the controller's CPU instantly. The function has no input parameters and only finishes the execution after updating all the integrated inputs.

Figure 173: Refresh integrated inputs function

5.16.2.2. RefreshIntegratedIoOutputs

This function allows to update all the outputs integrated to the controller's CPU instantly. The function has no input parameters and only finished the execution after updating all the integrated outputs.

Figure 174: Refresh integrated Outputs function

5.16.3. 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 the MasterTool IEC XE User Manual - MU299609), 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 the MasterTool IEC XE software *NextoStandard* library are described below (for the library insertion proceeding, see MasterTool IEC XE Programming Manual – MP399609, section Library).

5.16.3.1. 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 175 depicts the *TOF_RET* block and Figure 176 shows its graphic behavior.

Figure 175: 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 189: 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 190: TOF_RET Output Parameters

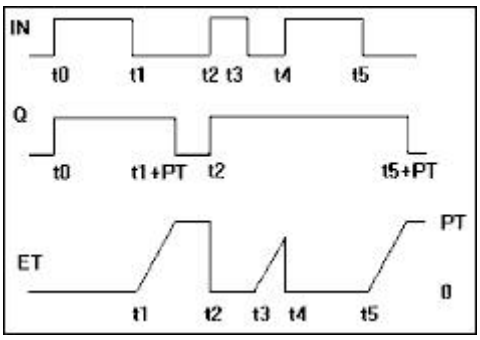


Figure 176: TOF_RET Block Graphic Behavior

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
```

5.16.3.2. 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 177 depicts the *TON_RET* block and Figure 178 shows its graphic behavior.

Figure 177: 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 191: 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 192: TON_RET Output Parameters

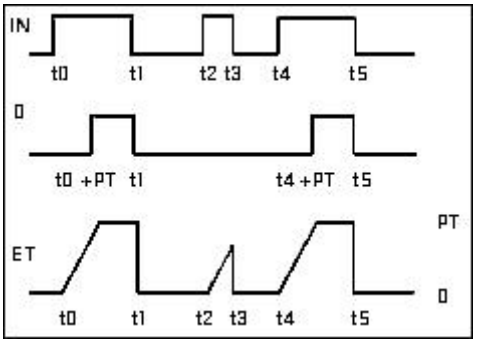


Figure 178: TON_RET Block Graphic Behavior

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
```

5.16.3.3. 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 179 depicts the *TP_RET* and Figure 180 shows its graphic behavior.

Figure 179: 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 193: 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 194: TP_RET Output Parameters

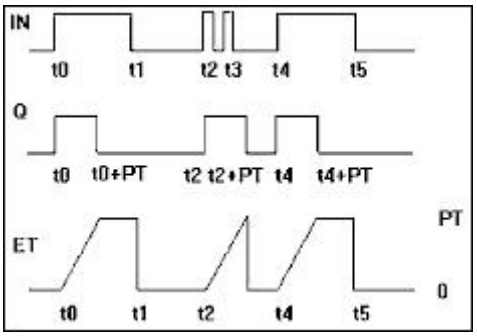


Figure 180: TP_RET Block Graphic Behavior

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:

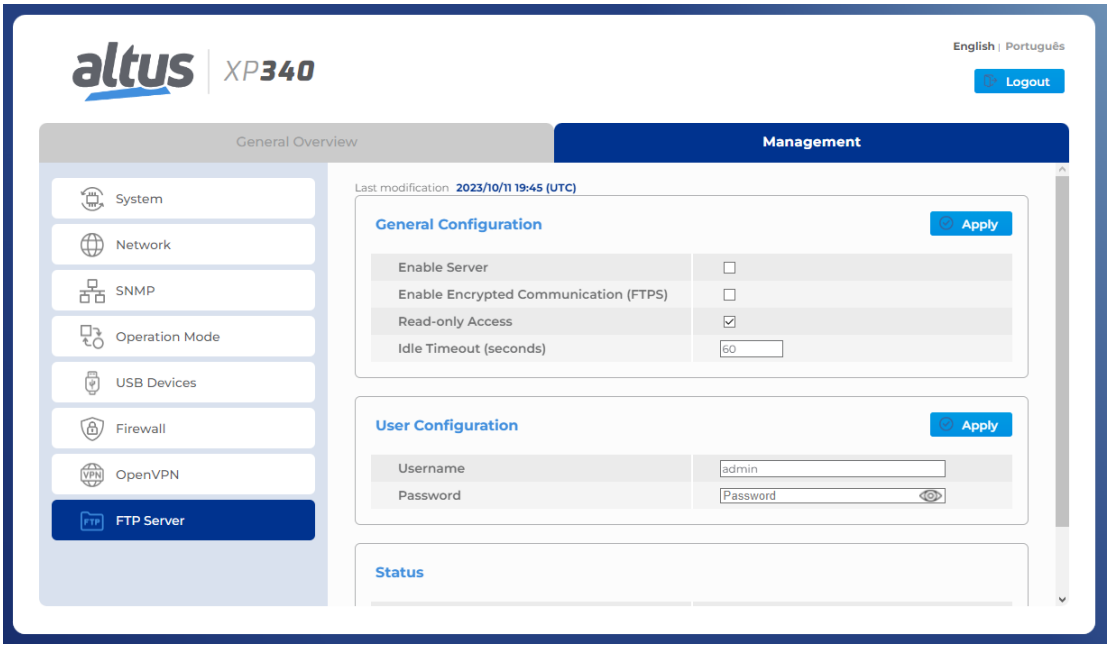


Figure 181: FTP Server Configuration Screen

FTP is a separate feature from MasterTool IEC XE, meaning it does not require any interaction with the programming tool. 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 195: 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. Firewall

5.18.1. 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.

5.18.2. Configuration

Firewall configuration is done through a dedicated section located in the *Management* tab of the controller's System Web Page, as shown below:

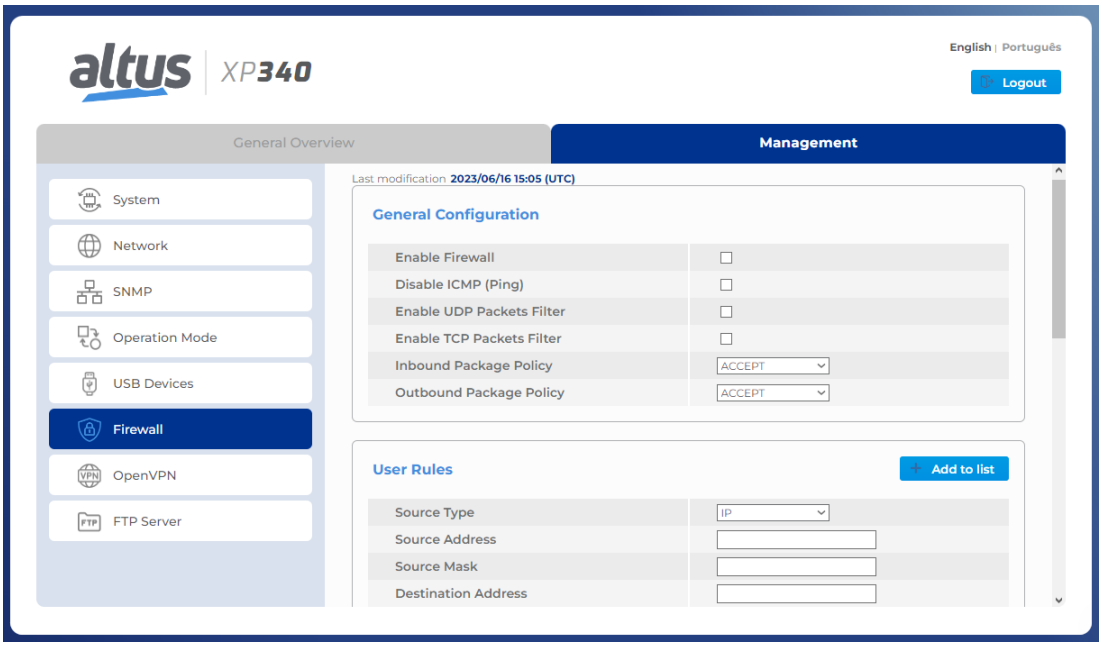


Figure 182: Firewall Configuration Screen

The Firewall is a separate feature from the MasterTool IEC XE, that is it doesn't require any interaction with the programming tool. 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.

5.18.3. General Configuration

The image below shows all the settings in the *General Configuration* table:

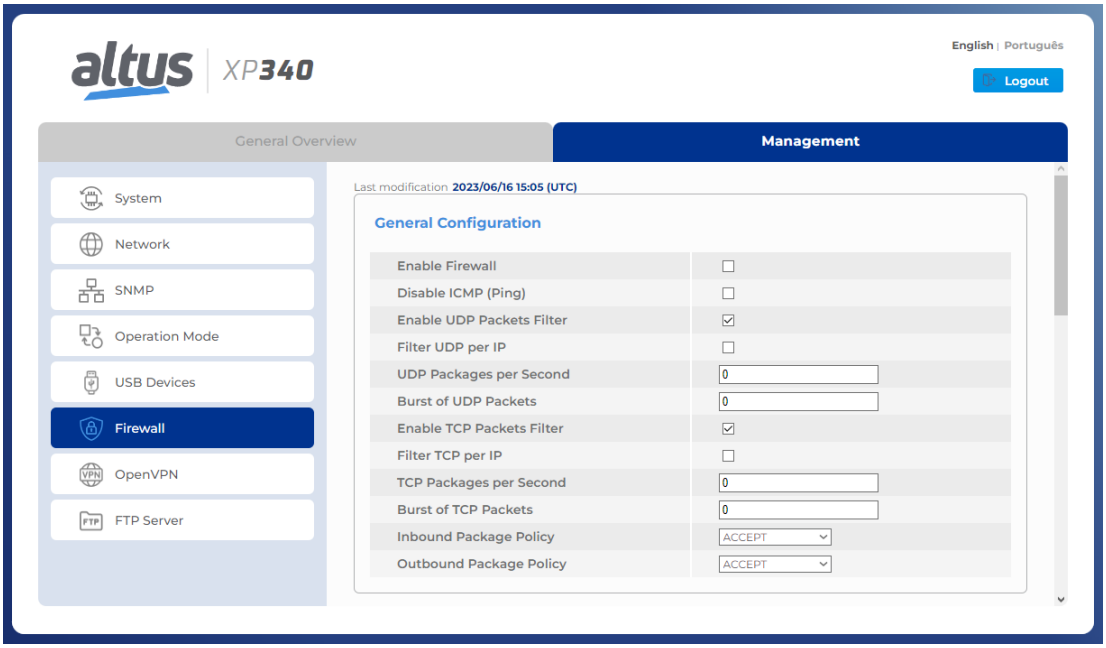


Figure 183: Firewall General Settings Table

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 185.

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.

5.18.4. 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.

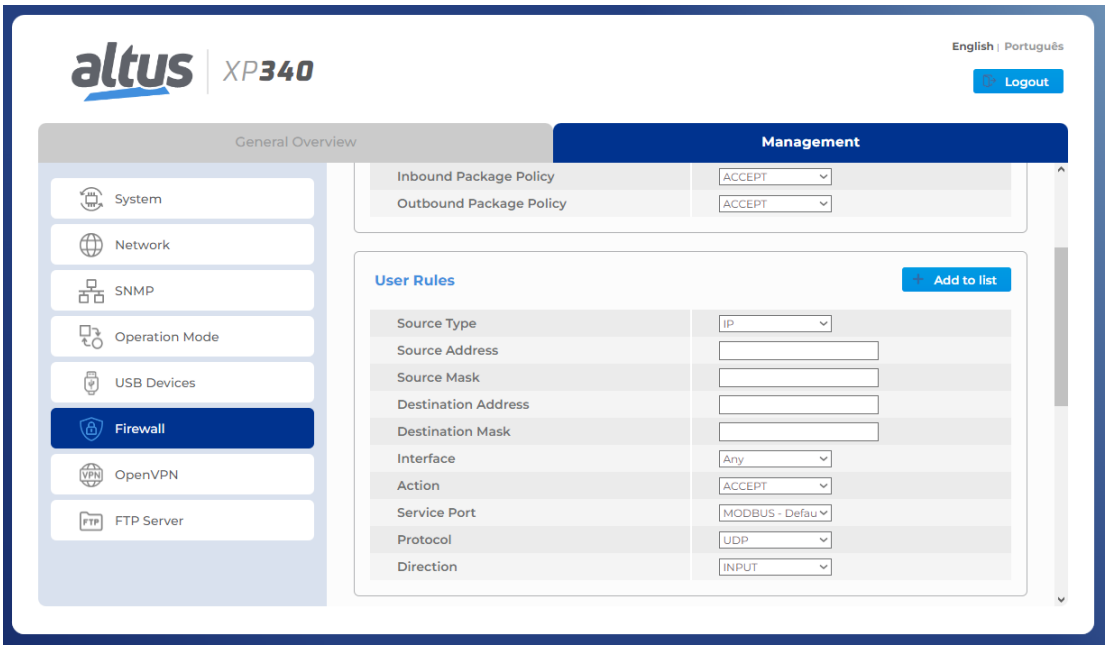


Figure 184: Firewall User Rules Configuration Table

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 services 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:

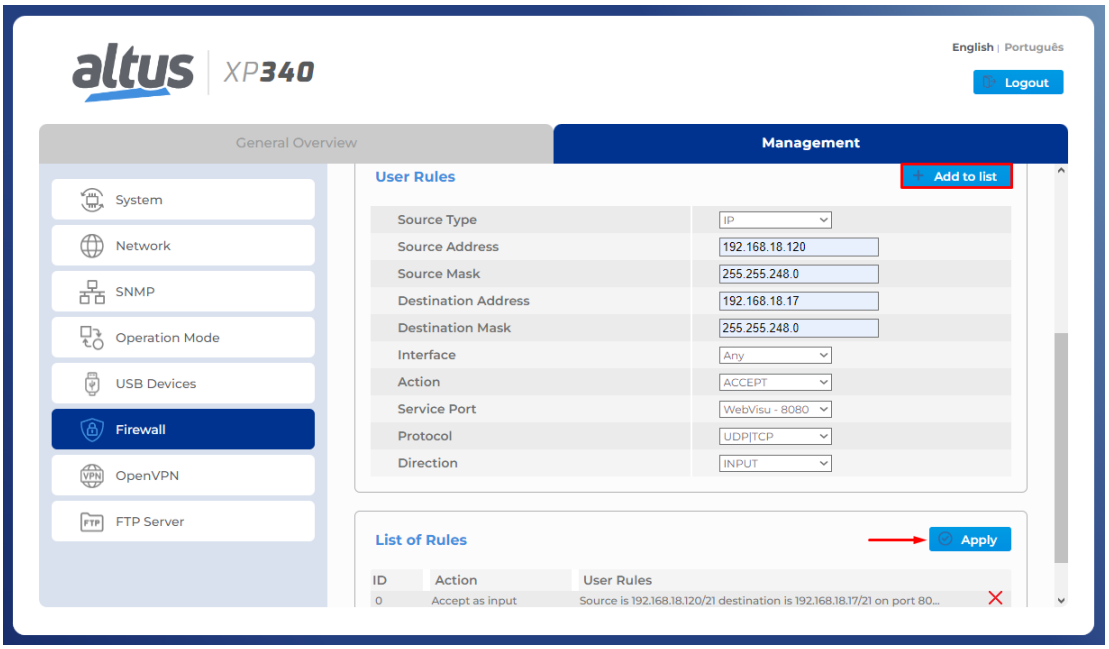


Figure 185: Firewall User Rules Enforcement Table

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.

5.19. OpenVPN

5.19.1. 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.

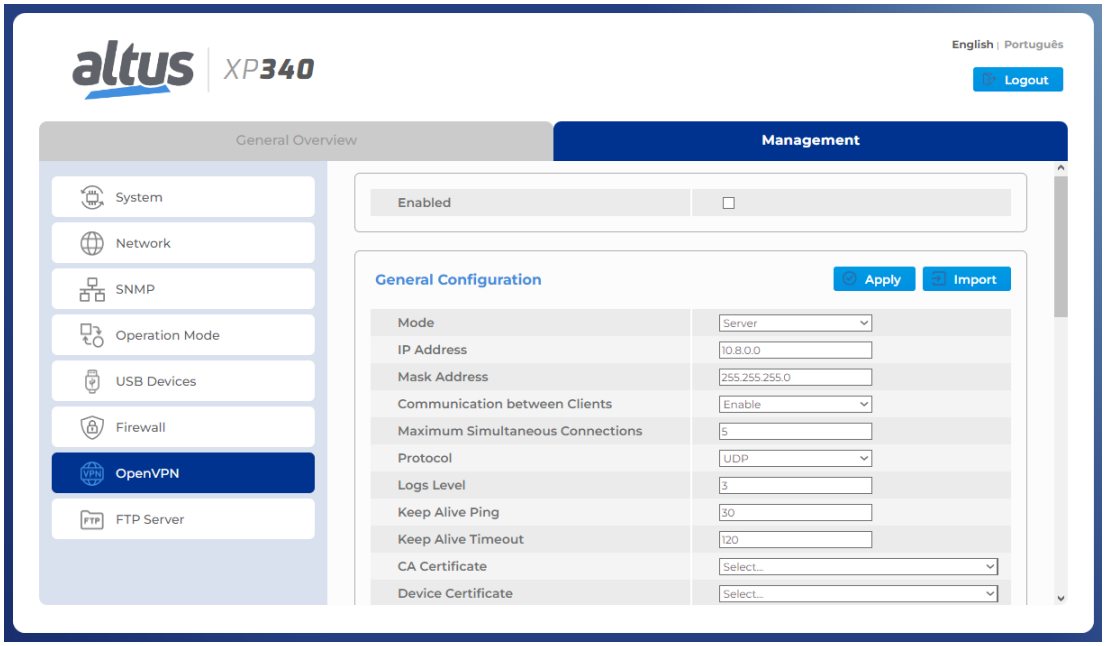


Figure 186: OpenVPN Configuration Screen

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.

5.19.2. Import Configuration

To quickly and easily configure the VPN on your device, you can use the *Import* button that appears in the picture 186 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.

5.19.3. OpenVPN Configuration

Here is an image with all the settings for an OpenVPN server:

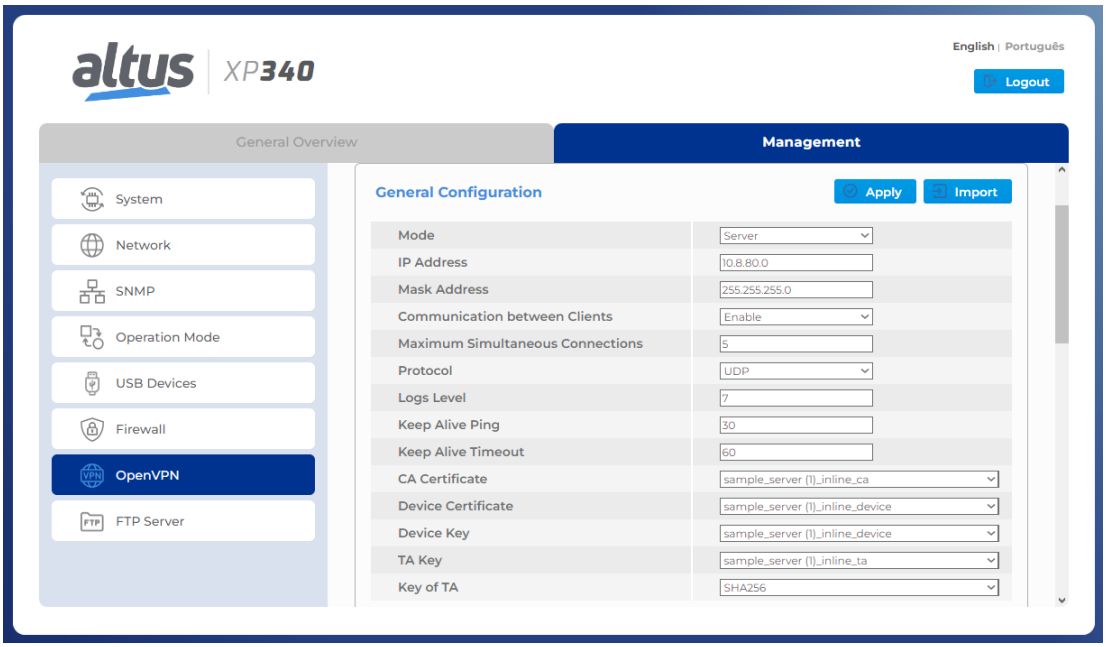


Figure 187: OpenVPN Server Configuration Table

Here is an image with all the settings for an OpenVPN client:

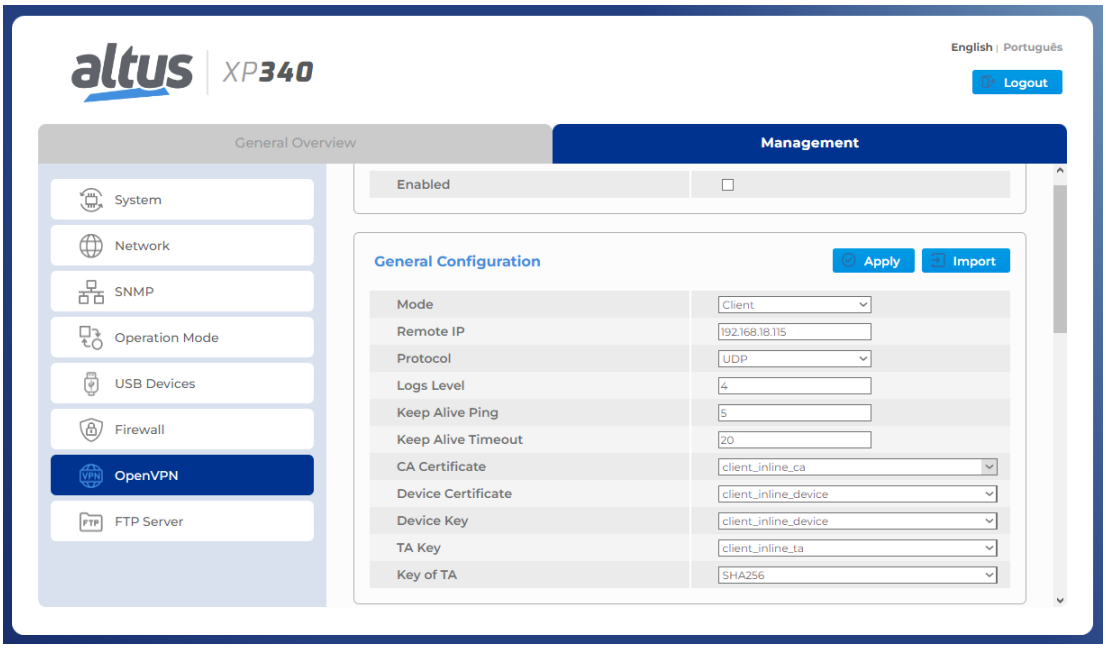


Figure 188: OpenVPN Client Configuration Table

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.

5.19.3.1. Common Configurations

Looking at the figures with the client configurations, figure 188, and the server configuration, figure 187, you can identify that several parameters are the same for both configurations. These are:

5.19.3.1.1. 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.

5.19.3.1.2. 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.

5.19.3.1.3. 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.

5.19.3.1.4. 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.

5.19.3.1.5. 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.

5.19.3.1.6. 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.

5.19.3.1.7. 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.

5.19.3.2. Exclusive Server Configurations

The exclusive server configurations, seen in figure 187, are described below.

5.19.3.2.1. 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.

5.19.3.2.2. 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.

5.19.3.2.3. 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.

5.19.3.2.4. 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.

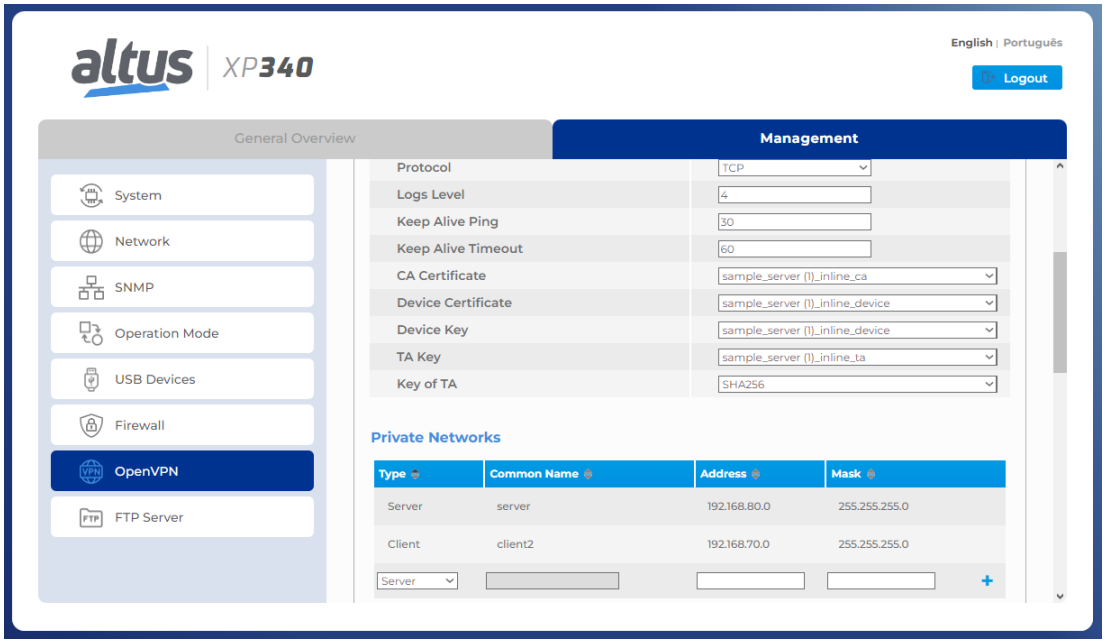


Figure 189: OpenVPN Private Network Configuration Table

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:

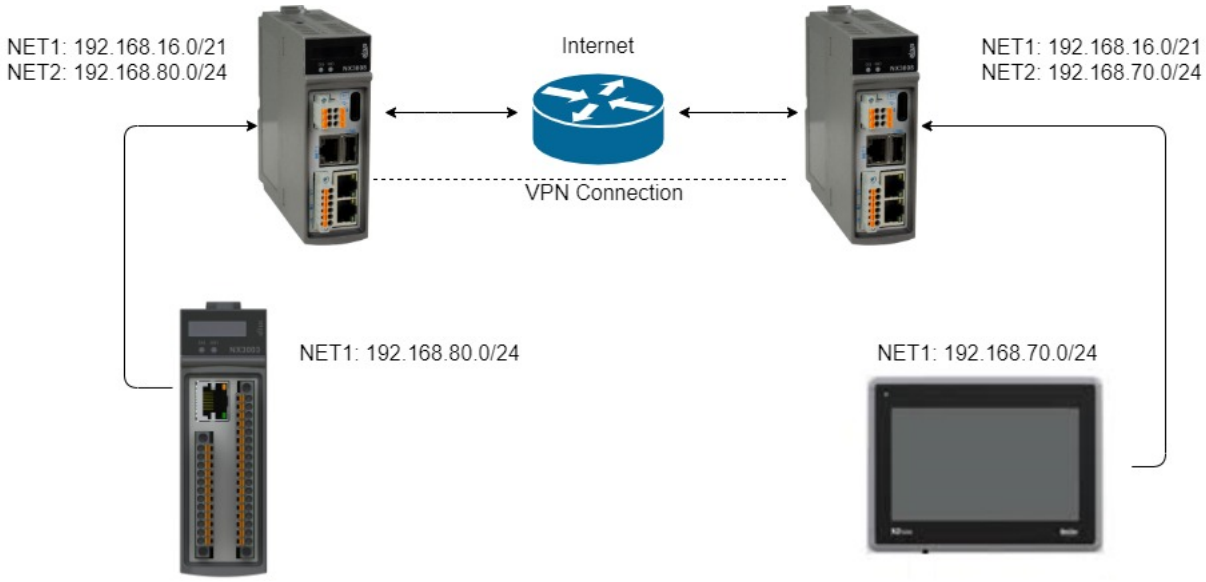


Figure 190: Architecture Example with Private Networks

In the example picture, the NX3008 on the left has a private network 80 configured on its NET 2, and connected to it is an NX3003 on the same network. The NX3008 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 NX3003 and HMI devices over VPN by configuring their respective private networks.

After filling the fields, shown in the figure 189, 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 the image 189. 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.

5.19.3.3. Exclusive Client Configurations

There is only one configuration unique to OpenVPN clients on the page, which you can see in the picture 188. This configuration is the *IP Remote*.

5.19.3.3.1. 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.

5.19.3.4. 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 188 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.

5.19.4. 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:

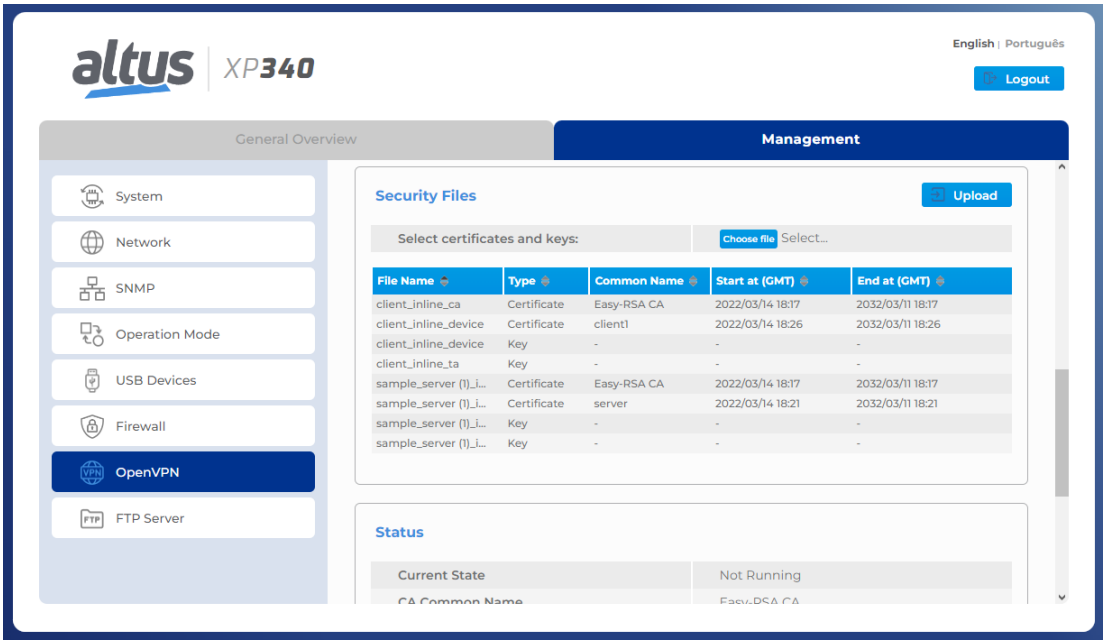


Figure 191: OpenVPN Security Files Table

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 191. 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*.

5.19.5. 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.

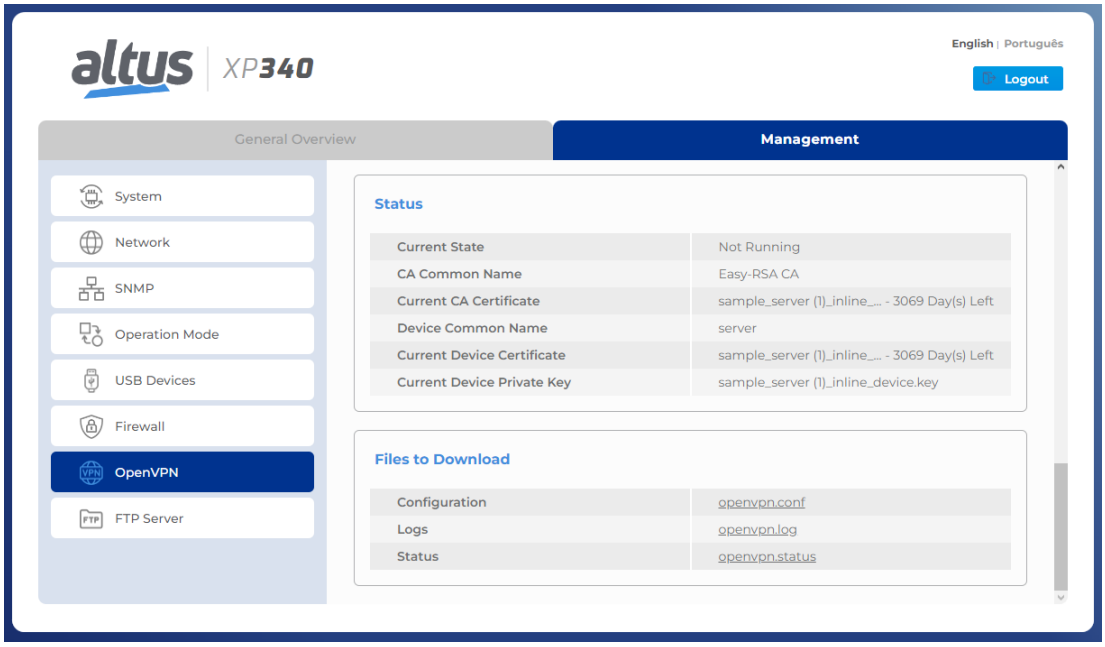


Figure 192: OpenVPN Status Table with Feature Disabled

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.

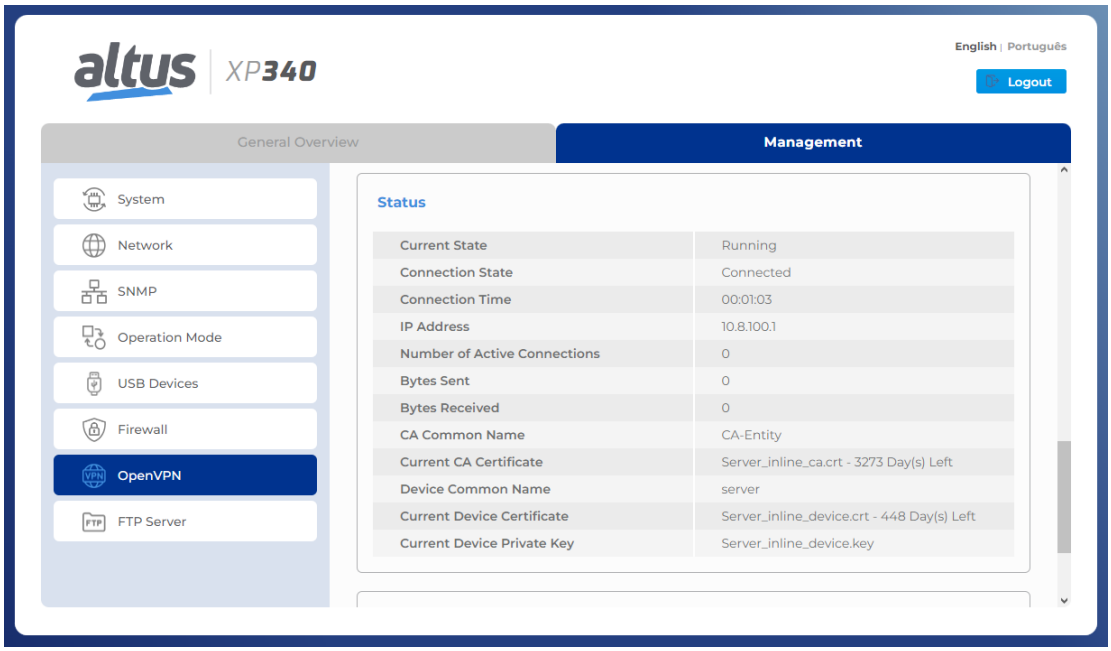


Figure 193: OpenVPN Status Table with Feature Enabled

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.

5.19.6. 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.

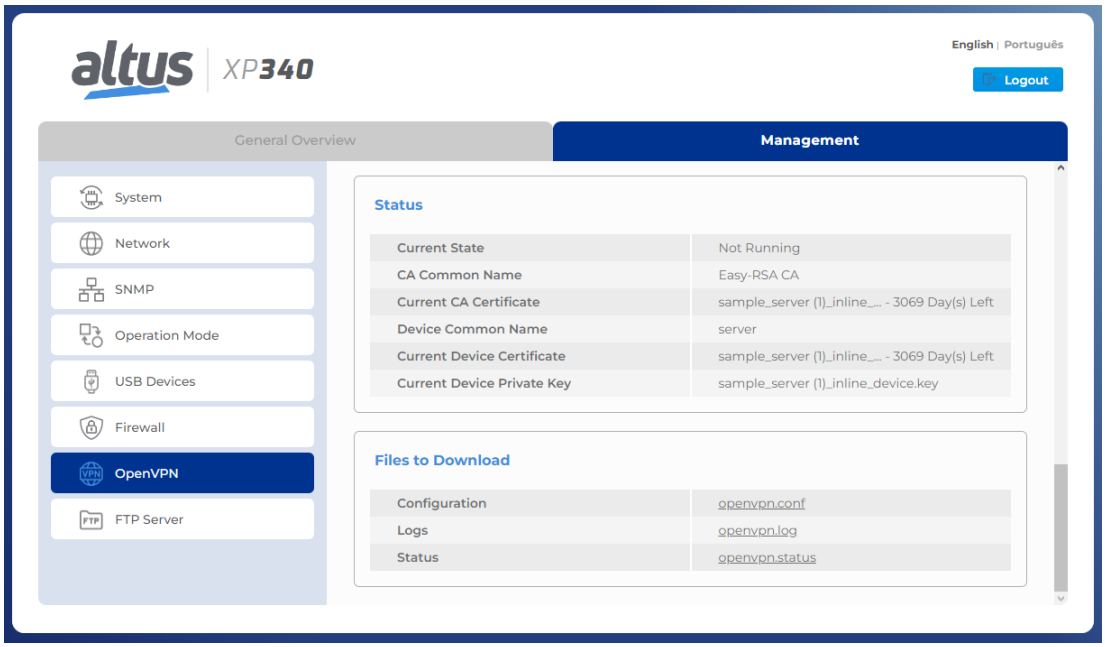


Figure 194: OpenVPN Download Section

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.

5.19.7. Architectures Configuration

This section will cover some possible configurations for OpenVPN, such as Host-to-Host, Host-to-Site, and Site-to-Site architectures.

5.19.7.1. Host-to-Host Configuration

Below is a picture that represents a Host-to-Host connection:

Figure 195: 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.

5.19.7.2. Host-to-Site Configuration

Below is a picture that represents a Host-to-Site connection:

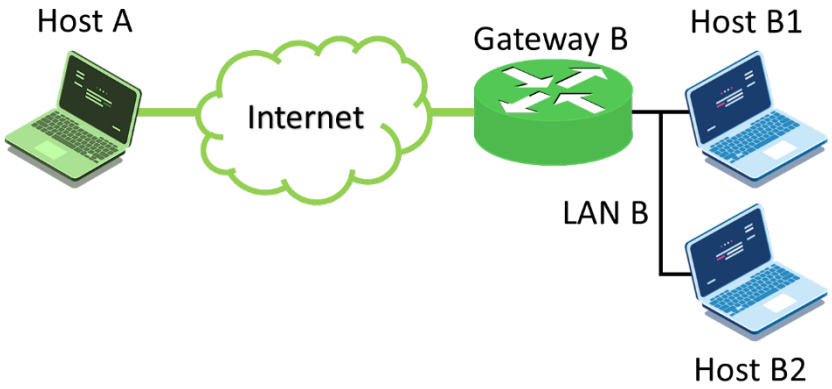


Figure 196: Example of Host-to-Site architecture

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.

5.19.7.3. Site-to-Site Configuration

Below is a picture that represents a Site-to-Site connection:

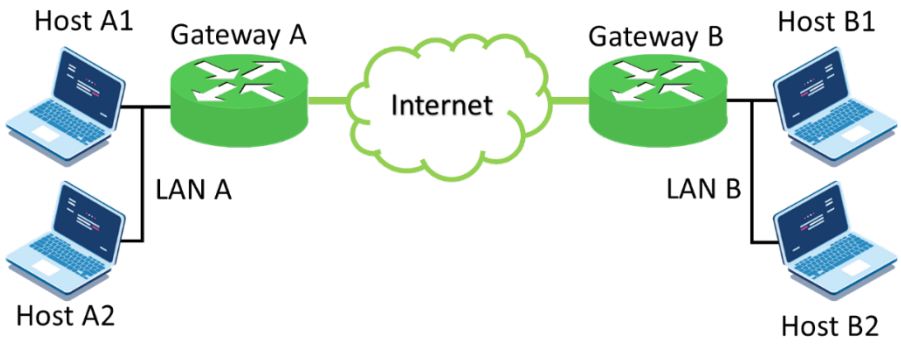


Figure 197: Example of Site-to-Site architecture

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"*.

5.20. Motion Control (Softmotion)

The XP350 and XP351 Nexto Series PLCs, support Motion Control functionality (Softmotion), enabling applications with up to 3 axis at 8 ms. For more detailed information or examples of use, consult the CODESYS Help at: https://www.helpme-codesys.com/codesys-softmotion.html.

Due to memory limitations, the XP350 and XP351 do not support the AxisGroup Object.

6. Maintenance

6.1. Diagnostics

Nexto Xpress controllers permit many ways to visualize the diagnostics generated by the system, which are:

- Diagnostics via LED
- Diagnostics via System Web Page
- Diagnostics via Variables
- Diagnostics via Function Blocks

The first one is purely visual, generated through two LEDs placed on the front panel (PWR and DG). The next feature is the graphic visualization in a System Web Page. The diagnostics are also provided as global symbolic variables to be used on the user application, for instance, being presented in a supervisory system. The last ones present specific conditions of the system functioning.

These diagnostics function is to point possible system installation or configuration problems, and communication network problems or deficiency.

6.1.1. Diagnostics via LED

Nexto Xpress controllers have a power (PWR) and a diagnostic indication (DG) LEDs. The following table shows the meaning of each state and its respective descriptions:

PWR	DG	Description	Causes	Priority
Off	**Off**	Not used	No power supply or Hardware problem	-
On	**Off**	Controller is booting	-	-
On	**On**	CPU is in RUN state, and there are no active diagnostics	-	5 (low)
On	**Blinking 1x**	CPU is in STOP state or no application loaded	-	2
On	**Blinking 2x**	There are active diagnostics	-	3
On	**Blinking 3x**	Data forcing	Some memory area is being forced by the user through MasterTool IEC XE	4
On	**Blinking 4x**	Hardware error	Internal hardware error	1
On	**Blinking 5x**	Power Failure	External power supply voltage is lower than acceptable threshold	0 (high)

Table 196: Description of the Diagnostic LEDs States

6.1.2. Diagnostics via System Web Page

As already known on Nexto Series, Nexto Xpress provides access to the operation states through a WEB page.

The utilization, and dynamics, is very intuitive and facilitates the user operations. The use of a supervisory system can be replaced when it is restricted to system status verification.

To access the controller WEB page, it is just to use a standard navigator (Internet Explorer 7 or superior, Mozilla Firefox 3.0 or superior and Google Chrome 8 or superior) and type, on the address bar, the controller IP address (e.g. Ex.: http://192.168.15.1). First, the controller information is presented, according to figure below:

Figure 198: Initial Screen

The user can choose from three language options: Portuguese, English and Spanish. The desired language is selected through the upper right menu. Additionally, the management tab has other features like *Firmware Update* and *SNMP*.

Firmware Update tab is restricted to the user, that is, only for internal use of Altus. In cases where the update is performed remotely (via a radio or satellite connection for example), the minimum speed of the link must be 128Kbps.

6.1.3. Diagnostics via Variables

Nexto Xpress 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 the MasterTool IEC XE.

6.1.3.1. Summarized Diagnostics

The following table shows the meaning of summarized diagnostics:

DG_XP3xx.tSummarized.*	Type	Description
bHardwareFailure	BOOL	TRUE – Controller has internal hardware failure.
		FALSE – The hardware is working properly.
bSoftwareException	BOOL	TRUE – One or more exceptions generated by the software.
		FALSE – No exceptions generated in the software.
bCOM1ConfigError	BOOL	TRUE – Error during/after the COM 1 serial interface configuration.
		FALSE – Correct COM 1 serial interface configuration.

DG_XP3xx.tSummarized.*	Type	Description
bNET1ConfigError	BOOL	TRUE – Error during/after the NET 1 Ethernet interface configuration.
		FALSE – Correct NET 1 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.
bRetentivityError	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

Table 197: 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.

6.1.3.2. Detailed Diagnostics

The tables below contains Nexto Xpress controllers' 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 IEC XE tree view. There, it is possible to see all datatypes defined in the structure
- Counters: All controller diagnostics counters return to zero when their limit value is exceeded

DG_XP3xx.tDetailed.*		Type	Description
Target.*	dwCPUModel	ENUM (BYTE)	Controller model, ex: MODEL_XP325
	abyCPUVersion	BYTE ARRAY(4)	Firmware version
	abyBootloaderVersion	BYTE ARRAY(4)	Bootloader version
Hardware.*	bRTCFailure	BOOL	The main processor is unable to communicate with the RTC hardware.
	bIntegratedIoFailure	BOOL	The main processor is unable to communicate with the integrated I/O hardware.

DG_XP3xx.tDetailed.*		Type	Description
Exception.*	wExceptionCode	WORD	Exception code generated by the RTS.
	byProcessorLoad	BYTE	Level, in percentage (%), of charge in the processor.
WebVisualization.*	byConnectedClients	BYTE	Clients number connected to the WebVisualization.
RetainInfo.*	byCPUInitStatus	BYTE	Controller startup status: 01: Hot start 02: Warm Start 03: Cold Start PS.: These variables are restarted in all startup.
	wCPUColdStartCounter	WORD	Counter of cold startups: Increments when the PLC starts with loss of retentivity. (0 to 65535)
	wCPUWarmStartCounter	WORD	Counter of hot startups: Increments when the PLC starts normally with valid retain data. (0 to 65535)
	wRTSResetCounter	WORD	Counter of reset performed by the RTS - Runtime System (0 to 65535).
	wWritesCounter	WORD	Counter of writes on retentive memory.
Reset.*	bBrownOutReset	BOOL	Last reset caused by failure in power supply.
	bWatchdogReset	BOOL	Last reset caused by internal watchdog error.
Serial. COM1.*	byProtocol	ENUM (BYTE)	Selected protocol in COM 1: NO_PROTOCOL (0): No protocol MODBUS_RTU_MASTER (1): MODBUS RTU Master MODBUS_RTU_SLAVE (2): MODBUS RTU Slave OTHER_PROTOCOL (3): 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 4095).
	wTXPendingBytes	WORD	Number of characters left in the transmission buffer in COM 1 (0 to 1023).
	wBreakErrorCounter	WORD	These counters are restarted in the following conditions: - Energizing - COM 1 serial port configuration - Removal of RX and TX queues PS.: When the controller is set Without Parity, the parity errors counter is not incremented in case it receives a different parity. In this case, an error of frame is indicated. The maximum value of each counter is 65535.
	wParityErrorCounter	WORD	
	wFrameErrorCounter	WORD	
	wRXOverrunCounter	WORD	
	bBusAlarm	BOOL	The bus has a critical error and is shutdown.

DG_XP3xx.tDetailed.*		Type	Description
CAN.*	byBusState	ENUM (BYTE)	Informs the status of the device: UNKNOWN: impossible to get the network state. ERR_FREE: no occurrence of CAN bus errors. ACTIVE: only few CAN bus errors (below warning level). WARNING: occurence of some CAN bus errors (above warning level). PASSIVE: too many CAN bus errors (above error level). BUSOFF: the node is shutdown (errors exceeded the admissible maximum).
	udiTxCounter	UDINT	Number of packets Tx changed in the PLC CAN bus.
	udiRxCounter	UDINT	Number of packets Rx changed in the PLC CAN bus.
	udiTxErrorCounter	UDINT	Number of packets Tx with errors in the PLC CAN bus.
	udiRxErrorCounter	UDINT	Number of packets Rx with errors in the PLC CAN bus.
	udiLostCounter	UDINT	Number of lost packets in the PLC CAN bus.
USB.*	byUSBDevice	ENUM (BYTE)	Type of the device connected to the USB port: NO_DEVICE UNKNOWN_DEVICE MASS_STORAGE_DEVICE SERIAL_CONVERTER_DEVICE MODEM_DEVICE WIFI_ADAPTER_DEVICE ETHERNET_ADAPTER_DEVICE
	bOvercurrent	BOOL	The device connected on the USB port is draining more current than supported
	bEnable	BOOL	The USB port is enabled
	tMassStorage. byMountState	ENUM (BYTE)	Informs the status of the device: MOUNTED UNMOUNTED
	tMassStorage. dwFreeSpaceKb	DWORD	Informs the free space on the mass storage device.
	tMassStorage. dwTotalSizeKb	DWORD	Informs the total size of the mass storage device.
	tSerialConverter. byProtocol	ENUM (BYTE)	Selected protocol in COM 10: NO_PROTOCOL (0): No protocol
	tSerialConverter. dwRXBytes	DWORD	Counter of characters received from COM 10 (0 to 4294967295).
	tSerialConverter. dwTXBytes	DWORD	Counter of characters transmitted from COM 10 (0 to 4294967295).
	tSerialConverter. wRXPendingBytes	WORD	Number of characters left in the reading buffer in COM 10 (0 to 4095).
	tSerialConverter. wTXPendingBytes	WORD	Number of characters left in the transmission buffer in COM 10 (0 to 1023).

DG_XP3xx.tDetailed.*	Type	Description
tSerialConverter. wBreakErrorCounter	WORD	These counters are restarted in the following conditions: - Energizing - COM 10 serial port configuration - Removal of RX and TX queues PS.: When the controller is set Without Parity, the parity errors counter is not incremented in case it receives a different parity. In this case, an error of frame is indicated. The maximum value of each counter is 65535.
tSerialConverter. wParityErrorCounter	WORD	
tSerialConverter. wFrameErrorCounter	WORD	
tSerialConverter. wRXOverrunCounter	WORD	
tModem. bConfigured	BOOL	Indicates that the modem was configured in the Web page.
tModem. byConnectionState	ENUM (BYTE)	The modem connection state: DISCONNECTED (1): modem is not connected or configured. CONNECTING (2): device configured, trying to connect to the Internet. FAILED_RETRYING (3): connection failed, the modem will try to connect again. CONNECTED (4): modem connected and up, the IP is available in the szIP diagnostic.
tModem. szIP	STRING	String with IP address used by the modem.
tWifiAdapter. bConfigured	BOOL	Indicates that the WiFi adapter was configured in the Web page.
tWifiAdapter. byConnectionState	ENUM (BYTE)	The WiFi adapter connection state: DISCONNECTED (1): WiFi is not connected and/or configured. CONNECTING (2): device configured, trying to connect to WiFi network. FAILED_RETRYING (3): connection failed, the WiFi will try to connect again. This can occur due to wrong password or the network is not available. CONNECTED (4): WiFi adapter connected to the network.
tWifiAdapter. szIP	STRING	String with the IP used in the WiFi network.
tWifiAdapter. szMask	STRING	String with the WiFi network mask.
tWifiAdapter. szGateway	STRING	String with the WiFi network Gateway.
tWifiAdapter. szMAC	STRING	String with WiFi adapter's MAC address (exclusive for the device).
tEthernetAdapter. bLinkDown	BOOL	Indicates the link state of the Ethernet adapter's interface.

DG_XP3xx.tDetailed.*		Type	Description
	tEthernetAdapter. szIP	STRING	String with the IP used by Ethernet adapter.
	tEthernetAdapter. szMask	STRING	String with the network mask used by Ethernet adapter.
	tEthernetAdapter. szGateway	STRING	String with the network Gateway used by Ethernet adapter.
	tEthernetAdapter. szMAC	STRING	String with Ethernet adapter's MAC address (exclusive for the device).
Ethernet. NET1.*	bLinkDown	BOOL	Indicates the link state in the interface.
	wProtocol	WORD	Selected protocol in NET 1: 00: Without protocol
	wProtocol. bMODBUS_RTU_ ETH_Client	BOOL	MODBUS RTU Client via TCP
	wProtocol. bMODBUS_ETH_ Client	BOOL	MODBUS TCP Client
	wProtocol. bMODBUS_RTU_ ETH_Server	BOOL	MODBUS RTU Server via TCP
	wProtocol. bMODBUS_ETH_ Server	BOOL	MODBUS TCP Server
	szIP	STRING(15)	Port IP Address
	szMask	STRING(15)	Port Subnet Mask
	szGateway	STRING(15)	Port Gateway Address
	szMAC	STRING(15)	Port MAC Address
	abyIP	BYTE ARRAY(4)	Port IP Address
	abyMask	BYTE ARRAY(4)	Port Subnet Mask
	abyGateway	BYTE ARRAY(4)	Port Gateway Address
	abyMAC	BYTE ARRAY(6)	Port MAC Address
	dwPacketsSent	DWORD	Counter of packets sent through the interface (0 to 4294967295).
	dwPacketsReceived	DWORD	Counter of packets received through the interface (0 to 4294967295).
	dwBytesSent	DWORD	Counter of bytes sent through the interface (0 to 4294967295).
	dwBytesReceived	DWORD	Counter of bytes received through the port (0 to 4294967295).
	dwTXDropErrors	DWORD	Counter of connection losses in the transmission through the interface (0 to 4294967295).
	dwTXCollisionErrors	DWORD	Counter of collision errors in the transmission through the interface (0 to 4294967295).
	dwRXDropErrors	DWORD	Counter of connection losses in the reception through the interface (0 to 4294967295).
	dwRXFrameErrors	DWORD	Counter of frame errors in the reception through the interface (0 a 4294967295).
UserFiles.*	byMounted	BYTE	Indicates if the memory used for recording the user files is able to receive data.

DG_XP3xx.tDetailed.*		Type	Description
	dwFreeSpacekB	DWORD	Free memory space for user files (Kbytes).
	dwTotalSizekB	DWORD	Storage capacity of the memory of user files (Kbytes).
UserLogs.*	byMounted	BYTE	Status of memory in which the user logs are inserted.
	wFreeSpacekB	WORD	Free memory space of user logs (Kbytes)
	wTotalSizekB	WORD	Storage capacity of the memory of user logs (Kbytes).
Application.*	byCPUState	ENUM (BYTE)	Informs the operation state of the CPU: 01: All user applications are in Run Mode 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.
Application Info.*	dwApplicationCRC	DWORD	32 bits CRC of Application. When the application is modified and sent to the controller, a new CRC is generated.
SNTP.*	bServiceEnabled	BOOL	SNTP Service enabled.
	byActiveTimeServer	ENUM (BYTE)	Indicates which server is active: NO_TIME_SERVER (0): None active server. PRIMARY_TIME_SERVER (1): Active Primary Server. SECONDARY_TIME_SERVER (2): Active Secondary Server.
	wPrimaryServerDownCount	WORD	Counter of times in which the primary server is unavailable (0 to 65535).
	wSecondaryServerDownCount	WORD	Counter of times in which the secondary server is unavailable (0 to 65535).
	dwRTCTimeUpdatedCount	DWORD	Counter of times the RTC was updated by the SNTP service (0 to 4294967295).
	byLastUpdateSuccessful	ENUM (BYTE)	Last update status: NOT_UPDATED (0): Not updated. UPDATE_FAILED (1): Failure. UPDATE_SUCCESSFUL (2): Successful.
	byLastUpdateTimeServer	ENUM (BYTE)	Server used in the last update: NO_TIME_SERVER (0): None update. PRIMARY_TIME_SERVER (1): Primary Server. SECONDARY_TIME_SERVER (2): Secondary Server.
	sLastUpdateTime. byDayOfMonth	BYTE	Date and time of the last sync time via SNTP.
	sLastUpdateTime. byMonth	BYTE	
	sLastUpdateTime. wYear	WORD	
	sLastUpdateTime. byHours	BYTE	
	sLastUpdateTime. byMinutes	BYTE	

DG_XP3xx.tDetailed.*		Type	Description
	sLastUpdateTime. bySeconds	BYTE	
	sLastUpdateTime. wMilliseconds	WORD	
IntegratedIO.*	AnalogInputs. tAnalogInput_xx. bInputNotEnable	BOOL	The input channel is not enabled on the configuration.
	AnalogInputs. tAnalogInput_xx. bOverRange	BOOL	The input signal level is above the maximum value defined for the selected input type.
	AnalogInputs. tAnalogInput_xx. bOpenLoop	BOOL	The input signal level is below minimum (only for 4-20mA mode).
	AnalogOutputs. tAnalogOutput_xx. bOutputNotEnable	BOOL	The output channel is not enabled on the configuration.
	AnalogOutputs. tAnalogOutput_xx. bOpenLoop	BOOL	The impedance of the load connected to the output channel is above the maximum accepted (only for current output mode).
	AnalogOutputs. tAnalogOutput_xx. bShortCircuit	BOOL	The impedance of the load connected to the output channel is below the minimum accepted (only for voltage output mode).
	RTDInputs. tRtdInput_xx. bInputNotEnable	BOOL	The input channel is not enabled on the configuration.
	RTDInputs. tRtdInput_xx. bOverRange	BOOL	The resistance is above the maximum value defined for the selected type.
	RTDInputs. tRtdInput_xx. bUnderRange	BOOL	The resistance is below the minimum value defined for the selected type (only for temperature sensors).
	byOperationMode	ENUM (BYTE)	VPN operating mode: SERVER (0): The device is operating as a server. CLIENT (1): The device is operating as a client.
	bServiceEnabled	BIT	VPN service enabled.
	bServiceRunning	BIT	VPN service running.
	byConnectionState	ENUM (BYTE)	VPN connection state: DISCONNECTED (0): Device disconnected from VPN. CONNECTING (1): Device configured, trying to connect to VPN. CONNECTED (2): Device connected to VPN.
	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 via VPN communication.

DG_XP3xx.tDetailed.*		Type	Description
	dwReceivedBytes	DWORD	Number of bytes received via VPN communication.
	CACertificate. CertificateName	STRING(64)	CA certificate name.
	CACertificate. CommonName	STRING(64)	CA certificate common name.
	CACertificate. sStartDate. byDayOfMonth	BYTE	Start date and time of the CA certificate.
	CACertificate. sStartDate. byMonth	BYTE	
	CACertificate. sStartDate. wYear	WORD	
	CACertificate. sStartDate. byHours	BYTE	
	CACertificate. sStartDate. byMinutes	BYTE	
	CACertificate. sStartDate. bySeconds	BYTE	
	CACertificate. sStartDate. wMilliseconds	WORD	
	CACertificate. sExpirationDate. byDayOfMonth	BYTE	Expiration date and time of the CA certificate.
	CACertificate. sExpirationDate. byMonth	BYTE	
	CACertificate. sExpirationDate. wYear	WORD	
	CACertificate. sExpirationDate. byHours	BYTE	
	CACertificate. sExpirationDate. byMinutes	BYTE	
	CACertificate. sExpirationDate. bySeconds	BYTE	
	CACertificate. sExpirationDate. wMilliseconds	WORD	
	CACertificate. dwValidDaysLeft	DWORD	Days left until the CA certificate expires.

DG_XP3xx.tDetailed.*		Type	Description
	DeviceCertificate. CertificateName	STRING(64)	Device certificate name.
	DeviceCertificate. CommonName	STRING(64)	Device certificate common name.
	DeviceCertificate. sStartDate. byDayOfMonth	BYTE	Start date and time of the Device certificate.
	DeviceCertificate. sStartDate. byMonth	BYTE	
	DeviceCertificate. sStartDate. wYear	WORD	
	DeviceCertificate. sStartDate. byHours	BYTE	
	DeviceCertificate. sStartDate. byMinutes	BYTE	
	DeviceCertificate. sStartDate. bySeconds	BYTE	
	DeviceCertificate. sStartDate. wMilliseconds	WORD	
	DeviceCertificate. sExpirationDate. byDayOfMonth	BYTE	Expiration date and time of the Device certificate.
	DeviceCertificate. sExpirationDate. byMonth	BYTE	
	DeviceCertificate. sExpirationDate. wYear	WORD	
	DeviceCertificate. sExpirationDate. byHours	BYTE	
	DeviceCertificate. sExpirationDate. byMinutes	BYTE	
	DeviceCertificate. sExpirationDate. bySeconds	BYTE	
	DeviceCertificate. sExpirationDate. wMilliseconds	WORD	
	DeviceCertificate. dwValidDaysLeft	DWORD	Days left until the Device certificate expires.
	sDeviceKeyName	STRING(64)	Device private key name.
	bServiceEnabled	BIT	Firewall service enabled.

DG_XP3xx.tDetailed.*		Type	Description
	sLatsModificationDateUTC.byDayOfMonth	BYTE	Date and time when the firewall rules were last modified, in UTC.
	sLatsModificationDateUTC.byMonth	BYTE	
	sLatsModificationDateUTC.wYear	WORD	
	sLatsModificationDateUTC.byHours	BYTE	
	sLatsModificationDateUTC.byMinutes	BYTE	
	sLatsModificationDateUTC.bySeconds	BYTE	
	sLatsModificationDateUTC.wMilliseconds	WORD	
FTP.*	bServiceEnabled	BIT	FTP server enabled.
	dwConnectedClients	DWORD	Number of FTP clients connected simultaneously.
	sLatsModificationDateUTC.byDayOfMonth	BYTE	Date and time when the FTP Server settings were last modified, in UTC.
	sLatsModificationDateUTC.byMonth	BYTE	
	sLatsModificationDateUTC.wYear	WORD	
	sLatsModificationDateUTC.byHours	BYTE	
	sLatsModificationDateUTC.byMinutes	BYTE	
	sLatsModificationDateUTC.bySeconds	BYTE	
	sLatsModificationDateUTC.wMilliseconds	WORD	

Table 198: Detailed Diagnostics Description

Notes:

Exception Code: The exception codes generated by the RTS (Runtime System) is presented below:

Code	Description	Code	Description
0x0000	There is no exception code.	**0x0051**	Access violation.
0x0010	Watchdog time of the expired IEC task (Software Watchdog).	**0x0052**	Privileged instruction.
0x0012	I/O configuration error.	**0x0053**	Page failure.
0x0013	Check-up errors after 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	Unsolved external references.	**0x0059**	Invalid OpCode.

Code	Description	Code	Description
0x0019	Download rejected.	**0x0100**	Data type misalignment.
0x001A	Project unloaded, as the retentive variables cannot be reallocated.	**0x0101**	Arrays limit exceeded.
0x001B	Project unloaded and deleted.	**0x0102**	Division by zero.
0x001C	Out of memory stack.	**0x0103**	Overflow.
0x001D	Corrupted retentive memory; cannot be mapped.	**0x0104**	Cannot be continued.
0x001E	Project can be loaded but it causes a break 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... IEC task configuration failure. Application working with wrong target. Illegal instruction.	**0x0151**	FPU: Abnormal operand.
		0x0152	FPU: Division by zero.
0x0023	Downloaded file Check-up error.	**0x0153**	FPU: Inexact result.
0x0024	Mismatch between the retentive identity and the current boot project program identity	**0x0154**	FPU: Invalid operation.
0x0025	IEC task configuration failure.	**0x0155**	FPU: Overflow.
0x0026	Application is running with the wrong target.	**0x0156**	FPU: Stack verification.
0x0050	Illegal instruction.	**0x0157**	FPU: Underflow.

Table 199: Exception Codes

Brownout Reset: The brownout reset diagnostic is only true when the power supply exceeds the minimum limit required in its technical features, remaining in low voltage, without suffering any interruption. The controller identifies the voltage break and indicates the power supply failure diagnostic. When the voltage is reestablished, the controller is restarted automatically and indicates the brownout reset diagnostic.

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.

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.

RTD Inputs: the table below describes the behavior of over and under range diagnostics according to the input type selected:

Diagnostics	**0 to 400 Ω Scale**		**0 to 4000 Ω Scale**		**Sensors of Platinum type (Pt) α = 0.00385**		**Sensors of Platinum type (Pt) α = 0.003916**	
	Resist.	**Count**	**Resist.**	**Count**	**Temp.**	**Count**	**Temp.**	**Count**
Over range	>420 Ω (420 to 404.1 Ω)	4200 (4200 to 4041)	>4200 Ω (4200 to 4041 Ω)	>4200 (4200 to 4041)	>850 °C	8500	>630 °C	6300
No diagnostics	0 to 404 Ω	0 to 4040	0 to 4040 Ω	0 to 4040	-200 to 850 °C	-2000 to 8500	-200 to 630 °C	-2000 to 6300
Under range	-	-	-	-	<-200 °C	-2000	<-200 °C	-2000

Table 200: RTD Input Diagnostics

6.1.4. 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.

6.1.4.1. GetTaskInfo

This function returns the task information of a specific application.

Figure 199: 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 201: 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 μs resolution.
dwMinScanTime	DWORD	Task cycle minimum time with 1 μs resolution.
dwMaxScanTime	DWORD	Task cycle maximum time 1 μs resolution.
dwAvgScanTime	DWORD	Task cycle average time with 1 μs resolution.
dwLimitMaxScan	DWORD	Task cycle maximum time before watchdog occurrence.
dwIECCycleCount	DWORD	IEC cycle counter.

Table 202: 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.
```

6.2. 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.
- The TVS diodes used for transient protection caused by atmospheric discharges must be periodically inspected, as they might be damaged or destroyed in case the absorbed energy is above limit. In many cases, the failure may not be visual. In critical applications, is recommendable the periodic replacement of the TVS diodes, even if they do not show visual signals of failure
- Connector block 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.

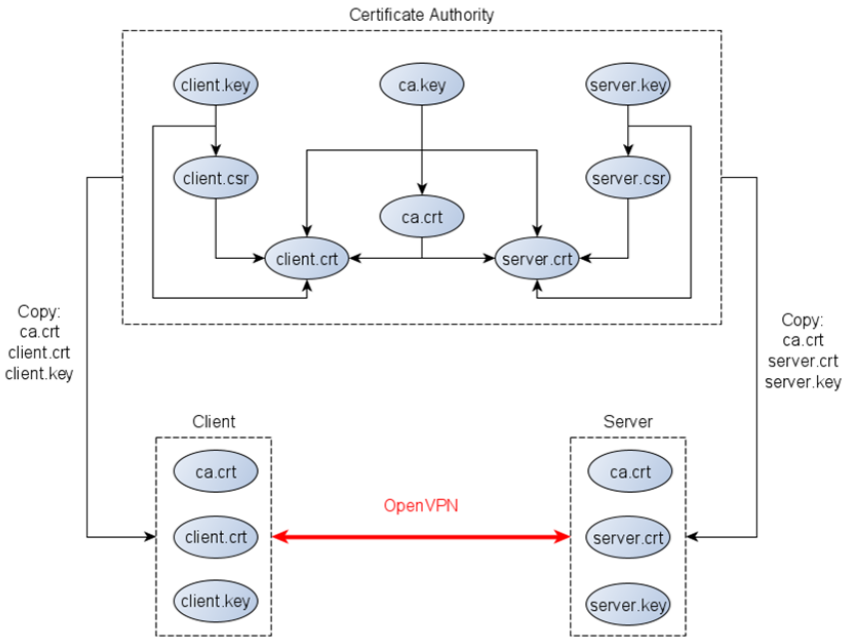


Figure 200: TLS Certificate Generation Flow

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 *.\EasyRSA-Start.bat* to enter the tool shell.

```
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 201: Certificate Generation using Easy-RSA (step 1)

2 - Copy the file *vars.example* and rename it to *vars* in the tools folder.

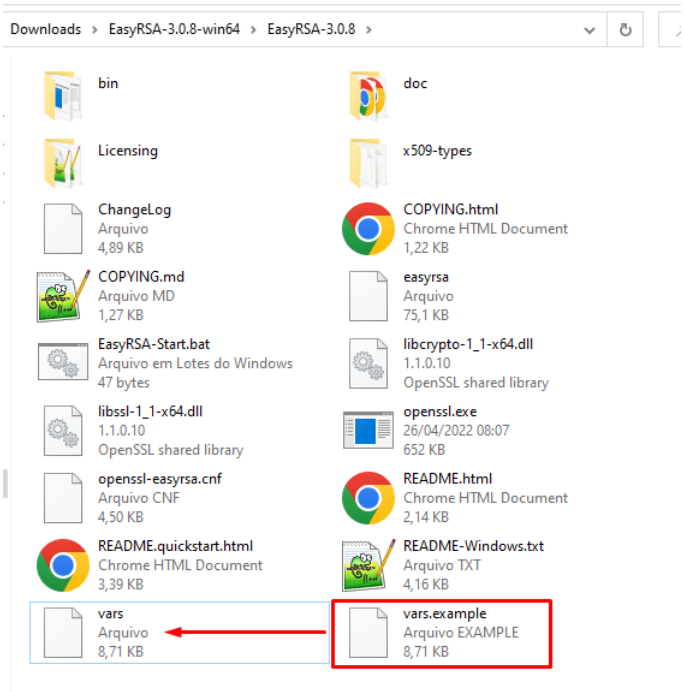


Figure 202: Certificate Generation using Easy-RSA (step 2)

3- Open the file *vars* with a text editor and change the Certification Authority information.

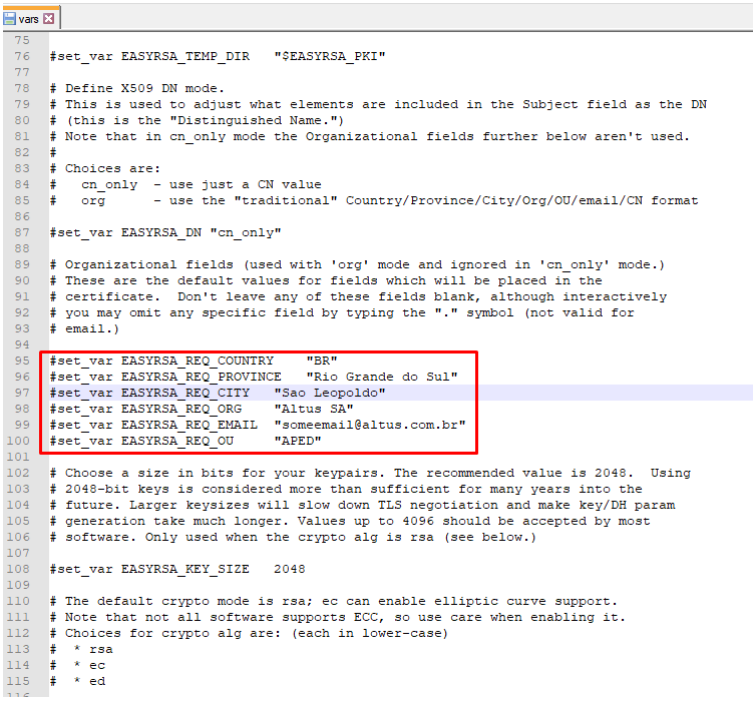


Figure 203: Certificate Generation using Easy-RSA (step 3)

4- Use the command *./easyrsa init-pki* to prepare the configuration.

Figure 204: Certificate Generation using Easy-RSA (step 4)

5- Then type *./easyrsa build-ca nopass* to generate the CA certificate. 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).

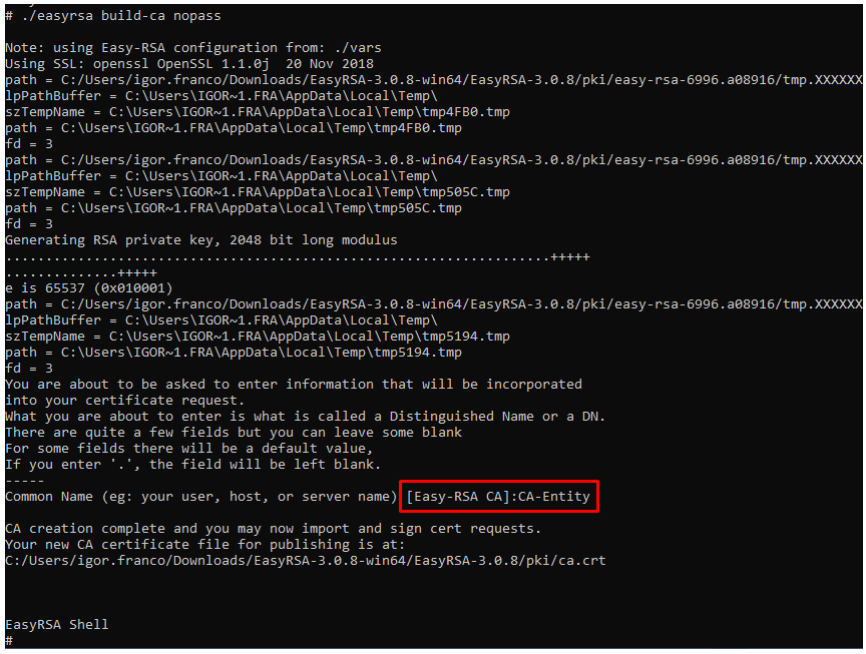


Figure 205: Certificate Generation using Easy-RSA (step 5)

6 - Generate the device key and request files using the command *./easyrsa gen-req DeviceName nopass*. Change the *DeviceName* with the desired common name. 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).

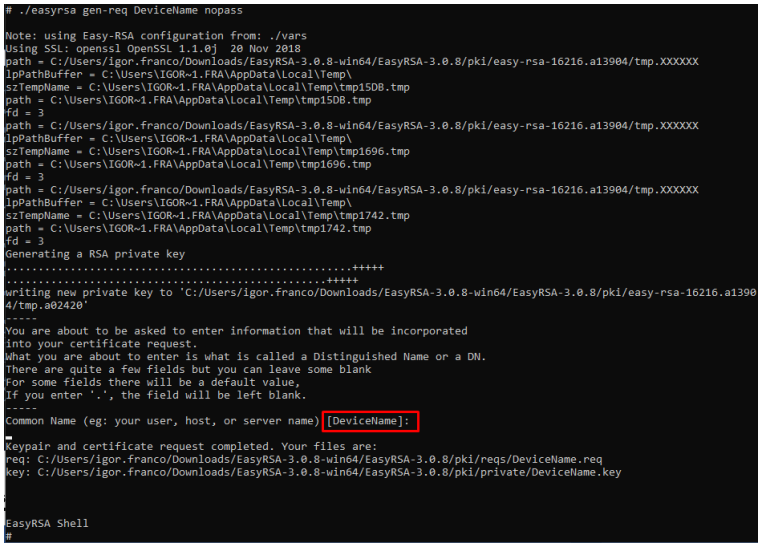


Figure 206: Certificate Generation using Easy-RSA (step 6)

7- Finally, type *./easyrsa sign-req server DeviceName* 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).

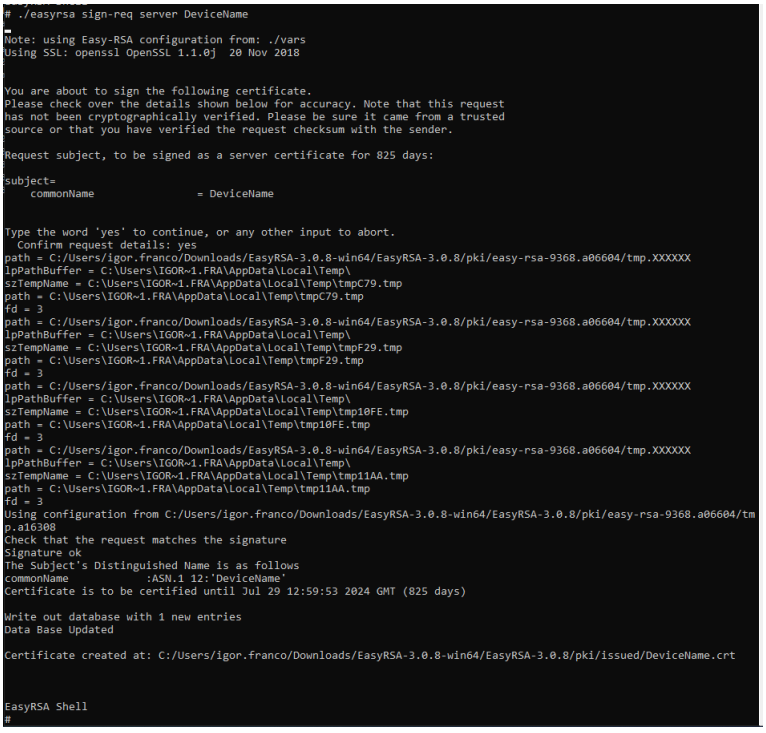


Figure 207: Certificate Generation using Easy-RSA (step 7)

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 208: 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*.

```
C:\Users\igor.franco\Downloads\Certificate>openssl genrsa -out ca.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....++++
.....................................................++++
e is 65537 (0x010001)

C:\Users\igor.franco\Downloads\Certificate>
```

Figure 209: Certificate Generation using OpenSSL (step 2)

3- Then generate the CA certificate based on the private key, using the *openssl req -new -x509 -days 365 -key ca.key -out ca.crt* command.

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.

```
C:\Users\igor.franco\Downloads\Certificate>openssl req -new -x509 -days 365 -key ca.key -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:CA-Entity
Email Address []:

C:\Users\igor.franco\Downloads\Certificate>
```

Figure 210: Certificate Generation using OpenSSL (step 3)

4- Now generate the device's private key, similar to step 2, using the command *openssl genrsa -out DeviceName.key 2048*.

```
C:\Users\igor.franco\Downloads\Certificate>openssl genrsa -out DeviceName.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................+++++
.....................+++++
e is 65537 (0x010001)

C:\Users\igor.franco\Downloads\Certificate>
```

Figure 211: Certificate Generation using OpenSSL (step 4)

5- After that, generate the certificate request file based on the private key using the command *openssl req -new -key DeviceName.key -out DeviceName.csr*.

Enter the desired information, and remember to use a common name other than CA.

```
C:\Users\igor.franco\Downloads\Certificate>openssl req -new -key DeviceName.key -out DeviceName.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:DeviceName
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

C:\Users\igor.franco\Downloads\Certificate>
```

Figure 212: 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 *openssl x509 -req -days 365 -in DeviceName.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out* command.

Set the expiration date as desired with the parameter *-days* and the serial number of the certificate with the argument *-set_serial*.

```
C:\Users\igor.franco\Downloads\Certificate>openssl x509 -req -days 365 -in DeviceName.csr -CA ca.crt -CAkey ca.key -s
et_serial 01 -out DeviceName.crt
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd, CN = DeviceName
Getting CA Private Key

C:\Users\igor.franco\Downloads\Certificate>
```

Figure 213: 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 command *openssl verify -purpose sslserver -CAfile ca.crt DeviceName.crt*.

```
C:\Users\igor.franco\Downloads\Certificate>openssl verify -purpose sslserver -CAfile ca.crt DeviceName.crt
DeviceName.crt: OK
```

Figure 214: 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 can be done with the following command: *openvpn –genkey secret ta.key*. Example of using the command in Windows:

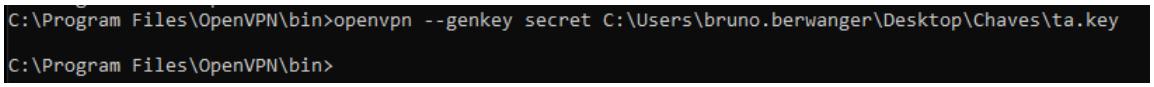


Figure 215: TA Key Generation in Windows

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: *openvpn –genkey –secret ta.key*. To run it, type the command in the terminal, as follows the example:

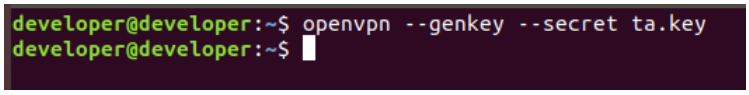


Figure 216: TA Key Generation in Linux

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.