User Manual
Nexto Serie CPU
NX3010

MU214613 Rev. W

September 25, 2024

Contents

altus

altus

1.   Introduction

Nexto Series programmable controllers are the ultimate solution for industrial automation and system control. With high technology embedded, the products of the family are able to control, in a distributed and redundant way, complex industrial systems, machines, high performance production lines and the most advanced processes of Industry 4.0. Modern and high-speed, the Nexto series uses cutting-edge technology to provide reliability and connectivity, helping to increase the productivity of different businesses.

Compact, robust and with high availability, the series products have excellent processing performance and rack expansion possibilities. Its architecture allows easy integration with supervision, control and field networks, in addition to PLC redundancy. The series equipment also offers advanced diagnostics and hot swapping, minimizing or eliminating maintenance downtime and ensuring a continuous production process.

Figure 1: NX3010

1.1.   Nexto Series

Nexto Series is a powerful and complete series of Programmable Controllers (PLC) with exclusive and innovative characteristics. Due to its flexibility, functional design, advanced diagnostic resources and modular architecture, the Nexto PLC can be used to control systems in small, medium and large scale applications.

Nexto Series architecture has a great variety of input and output modules. These modules combined with a powerful processor and a high speed bus based on Ethernet, fit to several application kinds as high speed control for small machines, complex distributed processes, redundant applications and systems with a great number of I/O as building automation. Furthermore, Nexto Series has modules for motion control, communication modules encompassing the most popular field networks among other features.

Nexto Series uses an advanced technology in its bus, which is based on a high speed Ethernet interface, allowing input and output information and data to be shared between several controllers inside the same system. The system can be easily divided and distributed throughout the whole field, allowing the use of bus expansion with the same performance of a local module, turning possible the use of every module in the local frame or in the expansion frames with no restrictions. For interconnection between frames expansions a simple standard Ethernet cable is used.

altus

Figure 2: Nexto Series – Overview

1.2. Innovative Features

Nexto Series brings to the user many innovations regarding utilization, supervision and system maintenance. These features were developed focusing a new concept in industrial automation.

Battery Free Operation: Nexto Series does not require any kind of battery for memory maintenance and real time clock operation. This feature is extremely important because it reduces the system maintenance needs and allows the use in remote locations where maintenance can be difficult to be performed. Besides, this feature is environmentally friendly.

Easy Plug System: Nexto Series has an exclusive method to plug and unplug I/O terminal blocks. The terminal blocks can be easily removed with a single movement and with no special tools. In order to plug the terminal block back to the module, the frontal cover assists the installation procedure, fitting the terminal block to the module.

Multiple Block Storage: Several kinds of memories are available to the user in Nexto Series CPUs, offering the best option for any user needs. These memories are divided in volatile memories and non-volatile memories. For volatile memories, Nexto Series CPUs offer addressable input (%I), addressable output (%Q), addressable memory (%M), data memory and redundant data memory. For applications that require non-volatile functionality, Nexto Series CPUs bring retain addressable memory (%Q), retain data memory, persistent addressable memory (%Q), persistent data memory, program memory, source code memory, CPU file system (doc, PDF, data) and memory card interface.

One Touch Diag: One Touch Diag is an exclusive feature that Nexto Series brings to PLCs. With this new concept, the user can check diagnostic information of any module present in the system directly on CPU's graphic display with one single press in the diagnostic switch of the respective module. OTD is a powerful diagnostic tool that can be used offline (without supervisor or programmer), reducing maintenance and commissioning times.

OFD – On Board Full Documentation: Nexto Series CPUs are capable of storing the complete project documentation in its own memory. This feature can be very convenient for backup purposes and maintenance, since the complete information is stored in a single and reliable place.

ETD – Electronic Tag on Display: Another exclusive feature that Nexto Series brings to PLCs is the Electronic Tag on Display. This new functionality brings the process of checking the tag names of any I/O pin or module used in the system directly to the CPU's graphic display. Along with this information, the user can check the description, as well. This feature is extremely useful during maintenance and troubleshooting procedures.

DHW – Double Hardware Width: Nexto Series modules were designed to save space in user cabinets or machines. For this reason, Nexto Series delivers two different module widths: Double Width (two backplane rack slots are required) and Single Width (only one backplane rack slot is required). This concept allows the use of compact I/O modules with a high-density of I/O points along with complex modules, like CPUs, fieldbus masters and power supply modules.

High-speed CPU: All Nexto Series CPUs were designed to provide an outstanding performance to the user, allowing the coverage of a large range of applications requirements.

iF Product Design Award 2012: Nexto Series was the winner of iF Product Design Award 2012 in industry + skilled trades group. This award is recognized internationally as a seal of quality and excellence, considered the Oscars of the design in Europe..

1.3. Documents Related to this Manual

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

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

For instance, the NX2020 module has the information for utilization features and purchasing on its CE. On another hand, the NX5001 has, besides the CE, a User Manual (MU).

It is advised the following documents as additional information source:

Code	Description	Language
CE114000	Nexto Series – Technical Characteristics	English
CT114000	Série Nexto – Características Técnicas	Portuguese
CE114100	NX3010 Technical Characteristics	English
CT114100	Características Técnicas NX3010	Portuguese
CE114200	NX8000 Power Supply Module Technical Characteristics	English
CT114200	Características Técnicas Fonte de Alimentação NX8000	Portuguese
CE114700	Nexto Series Backplane Racks Technical Characteristic	English
CT114700	Características Técnicas dos Bastidores da Série Nexto	Portuguese
CE114810	Nexto Series Accessories for Backplane Rack Technical Characteristics	English
CT114810	Características Técnicas Acessórios para Bastidor Série Nexto	Portuguese
CE114902	Nexto Series PROFIBUS-DP Master Technical Characteristics	English
CT114902	Características Técnicas do Mestre PROFIBUS-DP da Série Nexto	Portuguese
CE114903	Nexto Series Ethernet Module Technical Characteristics	English
CT114903	Características Técnicas Módulo Ethernet Série Nexto	Portuguese
CE114908	NX5110 and NX5210 PROFIBUS-DP Heads Technical Characteristics	English
CT114908	Características Técnicas Interfaces Cabeça PROFIBUSDP NX5110 e NX5210	Portuguese
MU214600	Nexto Series User Manual	English
MU214000	Manual de Utilização Série Nexto	Portuguese
MU214613	NX3010 CPU User Manual	English
MU214101	Manual de Utilização UCP NX3010	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

altus

Code	Description	Language
MU214601	NX5001 PROFIBUS DP Master User Manual	English
MU214001	Manual de Utilização Mestre PROFIBUS-DP NX5001	Portuguese
MU214608	Nexto PROFIBUS-DP Head Utilization Manual	English
MU214108	Manual de Utilização da Cabeça PROFIBUS-DP Nexto	Portuguese
MU214603	Nexto Series HART Manual	English
MU214606	MQTT User Manual	English
MU214609	OPC UA Server for Altus Controllers User Manual	English
MU214610	Advanced Control Functions User Manual	English
MU214621	Nexto Series PROFINET Manual	English
NAP151	Utilização do Tunneller OPC	Portuguese

Table 1: Related Documents

1.4. 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.

1.5. 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

1.6. 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 from Nexto Series CPUs NX3010.

2.1.   Panels and Connections

The following figure shows the CPU front panel.

Figure 3: NX3010

As it can be seen on the figure, on the front panel upper part is placed the graphic display used to show the whole system status and diagnostics, including the specific diagnostics of each module. The graphic display also offers an easy-to-use menu which brings to the user a quick mode for parameters reading or defining, such as: inner temperature (reading only) and local time (reading only).

Just below the graphic display, there are 2 LEDs used to indicate alarm diagnostics and watchdog circuit. The table below shows the LEDs description. For further information regarding the LEDs status and meaning, see Diagnostics via LED section.

LED	Description
DG	Diagnostics LED
WD	Watchdog LED

Table 2: LEDs Description

Nexto Series CPUs has two switches available to the user. The table below shows the description of these switches. For further information regarding the diagnostics switch, see sections One Touch Diag and CPU's Informative and Configuration Menu. For further information regarding the MS switch, see section Memory Card.

Keys	Description
Diagnostics Switch	Switch placed on the module upper part. Used for diagnostics visualization on the graphic display or for navigation through the informative menu and CPU configuration.
MS	Switch placed on the frontal panel. Used to securely remove the memory card.

Table 3: Keys Description

On the frontal panel the connection interfaces of Nexto Series CPUs are available. The table below presents a brief description of these interfaces.

Interfaces	Description
NET 1	RJ45 communication connector 10/100Base-TX standard. Allows the point to point or network communication. For further utilization information, see Ethernet Interfaces Configuration section.
COM 1	DB9 female connector for RS-232 communication standard. Allows the point to point or network. For further utilization information, see Serial Interfaces Configuration section.
COM 2	For further utilization information, see Serial Interfaces Configuration section.
MEMORY SLOT	Memory card slot. Allows the use of a memory card for different types of data storage such as: user logs, project documentation and files. For further utilization information, see Memory Card section.

Table 4: Connection Interfaces

2.2. General Features

2.2.1. Common General Features

	NX3010
Backplane rack occupation	2 posições sequenciais
Power supply integrated	No
Ethernet TCP/IP local interface	1
Serial Interface	2
CAN Interface	No
USB Port Host	No
Memory Card Interface	Yes
Real time clock (RTC)	Yes 1 ms resolution, max. variation of 2 seconds per day.
Watchdog	Yes
Status and diagnostic Indication	Graphic display LEDs System Web Page CPU internal memory

	NX3010
Programming languages	Structured Text (ST) Ladder Diagram (LD) Function Chart Sequencing (SFC) Function Block Diagram (FBD) Function Continuous Chart (CFC)
Tasks	Cyclic (periodic) Event-triggered (software interrupt) External event-triggered (hardware interrupt) Continuous (free running) State-triggered (software interrupt)
Online changes	Yes
Maximum number of tasks	16
Maximum number of expansion bus	8
Bus expansion redundancy support	Yes
Maximum number of I/O modules on the bus	128
Maximum number of additional Ethernet TCP/IP interface modules	0
Ethernet TCP/IP interface redundancy support	No
Maximum number of PROFIBUS-DP network (using master modules PROFIBUS-DP)	1
PROFIBUS-DP network redundancy support	No
Redundancy support (half-clusters)	No
Hot Swap support	Yes
Event oriented data reporting (SOE)	No
Protocol	-
Maximum Event Queue Size	-
Web pages development (available through the HTTP protocol)	No
One Touch Diag (OTD)	Yes
Electronic Tag on Display (ETD)	Yes

Table 5: Common Features

Notes:

Real Time Clock (RTC): The retention time, time that the real time clock will continue to update the date and time after a CPU power down, is 15 days for operation at 25 °C. At the maximum product temperature, the retention time is reduced to 10 days.

Maximum number of I/O modules on bus: The maximum number of I/O modules refers to the sum of all modules on the local bus and expansions.

altus

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	DNV Type Approval – DNV-CG-0339 (TAA000013D)
CE	2014/30/EU (EMC) 2014/35/EU (LVD) 2011/65/EU and 2015/863/EU (ROHS)
UKCA	S.I. 2016 No. 1091 (EMC) S.I. 2016 No. 1101 (Safety) S.I. 2012 No. 3032 (ROHS)
UL LISTED	UL/cUL Listed – UL 61010-1 UL 61010-2-201 (file E473496)
EAC	TR 004/2011 (LVD) CU TR 020/2011 (EMC)

Table 6: Standards and Certifications

2.2.3. Memory

	NX3010
Addressable input variables memory (%I)	32 Kbytes
Addressable output variables memory (%Q)	32 Kbytes
Direct representation variable memory (%M)	16 Kbytes
Symbolic variable memory	4 Mbytes
Persistent or Retain symbolic variables memory	64 Kbytes
Full Redundant Data Memory	-
Direct representation input variable memory (%I)	-
Direct representation output variable memory (%Q)	-
Direct representation variable memory (%M)	-
Symbolic variable memory	-
Program memory	4 Mbytes
Source code memory (backup)	40 Mbytes
User files memory	16 Mbytes

Table 7: Memory

Notes:

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

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

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

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

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

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

The full list of when the symbolic persistent variables keep their values and when the value is lost can be found in the table below. Besides the persistent area size declared in the table above, are reserved these 44 bytes to store information about the persistent variables (not available for use).

altus

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

Command/Operation	VAR	VAR RETAIN	VAR PERSISTENT
Power cycle	-	X	X
Reset warm	-	X	X
Reset cold	-	-	X
Reset origin	-	-	-
Remove CPU with integrated power supply from the rack while powered on	-	X	X
Remove the power supply or a CPU without integrated power supply from the rack while powered on	-	-	-
Download	-	-	X
Online change	X	X	X
Clean All	-	-	X
Reset Process (IEC 60870-5-104)	-	X	X

Table 8: Variables Behavior after the Event

In lower or equal versions 1.5.1.0 for NX3010, NX3020 and NX3030, the retentive and persistent symbolic memories and addressable output variables memory (%Q) used to have a fixed maximum size. On table below it's possible to consult the maximum sizes allowed in these older versions.

In versions above the ones mentioned, the CPUs allow flexible retentive and persistent memory sizes. For further information, refer to the section Retain and Persistent Memory Areas.

	NX3010
Retentive addressable output variables memory (%Q)	8 Kbytes
Persistent addressable output variables memory (%Q)	8 Kbytes
Retentive symbolic variables memory	32 Kbytes
Persistent symbolic variables memory	16 Kbytes

Table 9: Retentive and Persistent memories in older versions

In the case of Clean All command, if the application has been modified so that persistent variables have been removed, inserted into the top of the list or otherwise have had its modified type, the value of these variables is lost (when prompted by the tool MasterTool to download). Thus it is recommended that changes to the persistent variables GVL only include adding new variables on the list.

Program memory: Program memory is the maximum size that can be used to store the user application. This area is shared with source code memory and compilation data, being the total area the sum of "program memory" and "source code memory" and "Compilation memory".

Source code memory (backup): This memory area is used as project backup. If the user wants to import the project, MasterTool IEC XE will get the information required in this area. Care must be taken to ensure that the project saved as a backup is up to date to avoid the loss of critical information. This area is shared with source code memory, being the total area the sum of "program memory" and "source code memory".

User files memory: This memory area offers another way for the user to store files such as doc, pdf, images, and other files. This function allows data recording as in a memory card. For further information check User Files Memory.

2.2.4. Protocols

	NX3010	Interface
Open Protocol	Yes	COM1/COM2
MODBUS RTU Master	Yes	COM1/COM2
MODBUS RTU Slave	Yes	COM1/COM2
MODBUS TCP Client	Yes	NET1
MODBUS TCP Server	Yes	NET1
MODBUS RTU over TCP Client	Yes	NET1
MODBUS RTU over TCP Server	Yes	NET1
CANopen Master	No	-
CANopen Slave	No	-
CAN low level	No	-
SAE J-1939	No	-
OPC DA Server	Yes	NET1
OPC UA Server	Yes	NET1
EtherCAT Master	No	-
SNMP Agent	Yes	NET1
SOE (Event-oriented data)	No	-
IEC 60870-5-104 Server	No	-
EtherNet/IP Scanner	Yes	NET1
EtherNet/IP Adapter	Yes	NET1
MQTT Client	Yes	NET1
SNTP Client (for clock synchronism)	Yes	NET1
PROFINET Controller	Yes	NET1
PROFINET Device	No	-

Table 10: Protocols

Note:

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

altus

2.2.5. Serial Interfaces

2.2.5.1. COM 1

	COM 1
Connector	Shielded female DB9
Physical interface	RS-232C
Modem signals	RTS, CTS, DCD
Baud rate	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps

Isolation
    Logic to Serial Port
    Serial Port to protection earth ⏚
	Not isolated
	
	1000 Vac / 1 minute

Table 11: COM 1 Serial Interface Features

2.2.5.2. COM 2

	COM 2
Connector	Shielded female DB9
Physical interface	RS-422 or RS-485 (depending on the selected cable)
Communication direction	RS-422: full duplex RS-485: half duplex
RS-422 max. transceivers	11 (1 transmitter and 10 receivers)
RS-485 max. transceivers	32
Termination	Yes (optional via cable selection)
Baud rate	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps

Isolation
    Logic to Serial Port
    Serial Port to protection earth ⏚
	1000 Vac / 1 minute
	
	1000 Vac / 1 minute

Table 12: COM 2 Serial Interface Features

Notes:

Physical interface: Depending on the configuration of the used cable, it's possible to choose the kind of physical interface: RS-422 or RS-485. The list of cables can be found at Related Products section.

RS-422 maximum transceivers: It's the maximum number of RS-422 interfaces that can be used on the same bus.

RS-485 maximum transceivers: It's the maximum number of RS-485 interfaces that can be used on the same bus.

2.2.6.   Ethernet Interfaces

2.2.6.1.   NET 1

	NET 1
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 (Full Duplex)
Data link layer	LLC (Logical Link Control)
Network layer	IP (Internet Protocol))
Transport layer	TCP (Transmission Control Protocol) UDP (User Datagram Protocol)
Diagnostic	LEDs - green (speed), yellow (link/activity)
Isolation **Ethernet interface to Serial Port**	1500 Vac / 1 minute

Table 13: Ethernet NET 1 Interface Features

2.2.7.   Memory Card Interface

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

	Memory Card
Maximum Capacity	32 Gbytes
Minimum Capacity	2 Gbytes
Type	MiniSD
File System	FAT32
Remove card safely	Yes, by pressing MS switch

Table 14: Memory Card Interface Features

Notes:

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

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

File System: It is recommended to format the memory card using the Nexto CPU, otherwise it may result in performance loss in the memory card interface.

altus

2.2.8. Environmental Characteristics

	NX3010
Current consumption on the power supply rail	800 mA
Dissipation	4 W
Operating temperature	0 to 60 °C
Storage temperature	-25 to 75 °C
Relative humidity	5% to 96%, non-condensing
Conformal coating	Yes
IP Level	IP 20
Module dimensions (W x H x D)	36,00 x 114,63 x 115,30 mm
Package dimensions (W x H x D)	44,00 x 122,00 x 147,00 mm
Weight	350 g
Weight with package	400 g

Table 15: Environmental Characteristics

Notes:

Conformal coating of electronic circuits: The covering of electronic circuits protects internal parts of the product against moisture, dust and other harsh elements to electronic circuits.

2.3. Compatibility with Other Products

To develop an application for Nexto Series CPUs, 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:

Nexto Series CPUs	MasterTool IEC XE	Firmware version
NX3010, NX3020, NX3030	1.00 to 2.09	1.2.0.9 to 1.7.0.14
NX3010, NX3020, NX3030	3.00 or above	1.8.3.0 or above

Table 16: Compatibility with other products

Additionally, along the development roadmap of MasterTool IEC XE some features may be included (like special Function Blocks, etc...), which can introduce a requirement of minimum firmware version. During the download of the application, MasterTool 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 Nexto Series CPUs performance relies on:

- User Application Time
- Application Interval
- Operational System Time
- Module quantity (process data, input/output, among others)

2.4.1. MainTask Interval Time

The MainTask interval time setting depends on the selected project profile. For the profiles Simple, Normal, Experienced, and Custom profiles, the interval can be set with values from 1 ms to 750 ms. For the Machine Machine Profile, the interval can be configured with values from 1 ms to 100 ms.

2.4.2. Application Times

The execution time of Nexto CPUs application depends on the following variables:

- Input read time (local and remote)
- Tasks execution time
- Output write time (local and remote)

It is important to stress that the execution time of the "MainTask" will be directly influenced by the "Configuration" system task, a task of high priority, executed periodically by the system. The "Configuration" task may interrupt the "MainTask" and, when using the communication modules, as the Ethernet NX5000 module, for instance, the time addition to the "MainTask" may be up to 25% of the execution average time.

2.4.3. Time for Instructions Execution

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

Instruction	Language	Variables	Instruction Times (μs)
1000 Contacts	LD	BOOL	6
1000 Divisions	ST	INT	43
		REAL	81
	LD	INT	43
		REAL	81
1000 Multiplications	ST	INT	15
		REAL	23
	LD	INT	15
		REAL	23
1000 Sums	ST	INT	15
		REAL	23
	LD	INT	15
		REAL	23
1000 PID	ST	REAL	< 5000

Table 17: Instruction Times

2.4.4. Initialization Times

Nexto Series CPUs have initialization times of 50 s, and the initial screen with the NEXTO logo (Splash) is presented after 20 s from the power switched on.

2.5. Physical Dimensions

Dimensions in mm.
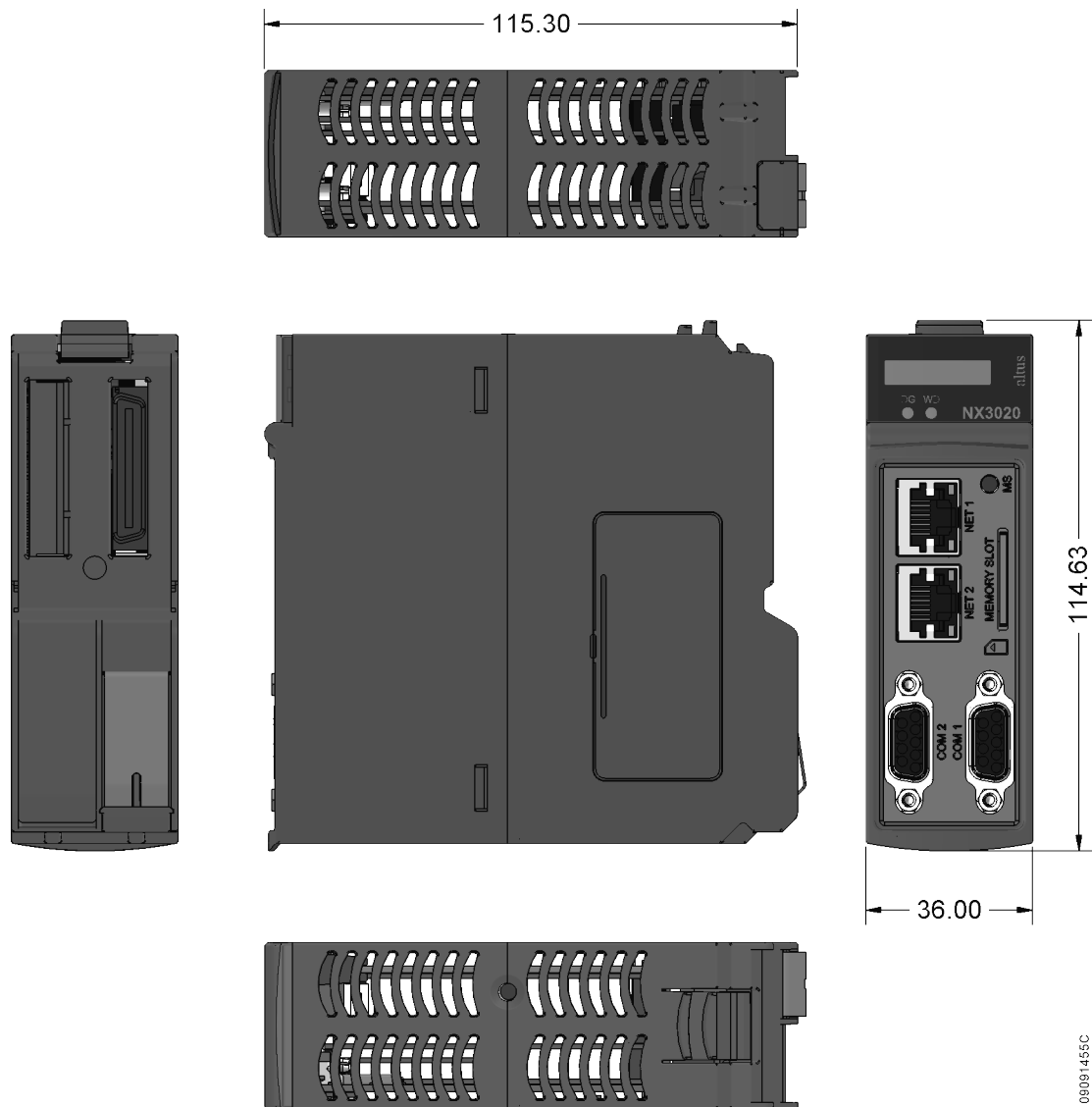
Figure 4: CPU Physical Dimensions

2.6. Purchase Data

2.6.1. Integrant Itens

The product package contains the following items:

- NX3010 module

2.6.2. Product code

The following code should be used to purchase the product:

altus

Code	Description
NX3010	High-speed CPU, 1 Ethernet port, 2 serial channels, memory card interface and remote rack expansion support

Table 18: Product Code

2.7. Related Products

The following products must be purchased separately when necessary:

Code	Description
MT8500	MasterTool IEC XE
AL-2600	RS-485 network branch and terminator
AL-2306	RS-485 cable for MODBUS or CAN network
AL-1729	RJ45-CMDB9 Cable
AL-1748	CMDB9-CFDB9 Cable
AL-1752	CMDB9-CMDB9 Cable
AL-1753	CMDB9-CMDB25 Cable
AL-1754	CMDB9-CFDB9 Cable
AL-1761	CMDB9-CMDB9 Cable
AL-1762	CMDB9-CMDB9 Cable
AL-1763	CMDB9-Terminal Block Cable
AL-1766	CFDB9-Terminal Block Cable
NX9101	32 GB microSD memory card with miniSD and SD adapters
NX9202	RJ45-RJ45 2 m Cable
NX9205	RJ45-RJ45 5 m Cable
NX9210	RJ45-RJ45 10 m Cable
NX9000	8-Slot Backplane Rack
NX9001	12-Slot Backplane Rack
NX9002	16-Slot Backplane Rack
NX9003	24-Slot Backplane Rack
NX8000	30 W 24 Vdc Power Supply Module

Table 19: Related Products

altus

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.

AL-2600: This module is used for branch and termination of RS-422/485 networks. For each network node, an AL-2600 is required. The AL-2600 that is 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-1729: RS-232C standard cable with one RJ45 connector and one DB9 male connector for communication between CPUs of the Nexto Series and other Altus products of the DUO Series, Piccolo Series and Ponto Series.

AL-1748: RS-232C standard cable with one DB9 male connector and one DB9 female connector for communication between CPUs of the Nexto Series and Altus products of the Cimrex Series.

AL-1752: RS-232C standard cable with two DB9 male connectors for communication between CPUs of the Nexto Series and Altus products of the H Series and iX series.

AL-1753: RS-232C standard cable with one DB9 male connector and one DB25 male connector for communication between CPUs of the Nexto Series and Altus products of the H Series.

AL-1754: RS-232C standard cable with one DB9 male connector and one DB9 female connector for communication between CPUs of the Nexto Series and Altus products of the Exter Series or Serial port, RS-232C standard, of a microcomputer.

AL-1761: RS-232C standard cable with two DB9 male connectors for communication between Nexto Series CPUs and Altus products of the AL Series.

AL-1762: RS-232C standard cable with two DB9 male connectors for communication between Nexto Series CPUs.

AL-1763: Cable with one DB9 male connector and terminal block for communication between CPUs of the Nexto Series and products with RS-485/RS-422 standard terminal block.

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

NX9202/NX9205/NX9210: Cables used for Ethernet communication and to interconnect the bus expansion modules.

3.   Installation

This chapter presents the necessary proceedings for the Nexto Series CPUs physical installation, as well as the care that should be taken with other installation within the panel where the CPU 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.

3.1.   Mechanical Installation

Nexto Series CPUs must be inserted in the backplane rack position 2, just beside the Power Supply Module. All information regarding mechanical installation and module insertion can be found at MU214600 - Nexto Series User Manual .

3.2.   Electrical Installation

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

The CPUs energy supply come from the Power Supply Module which supplies the CPUs power through the backplane rack connection. It does not need any external connection. The module grounding is given through the contact between the module grounding spring and the backplane rack.

The figure below shows the Nexto Series CPUs electric diagram installed in a Nexto Series backplane rack.

The connectors placement depicted are merely illustrative.
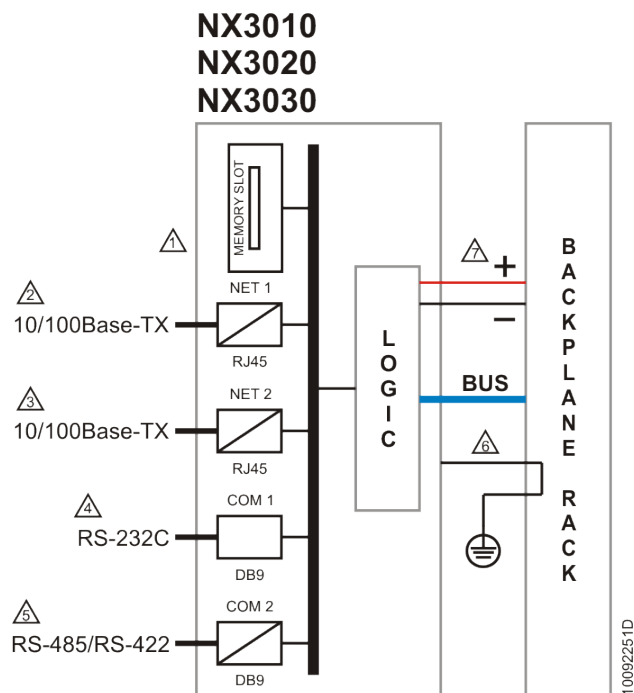
Figure 5: NX3010, NX3020 and NX3030 CPUs Electric Diagram

Diagram Notes:

△1 Memory card interface.

△2 Ethernet interface 10/100Base-TX standard for programming, debugging and MODBUS TCP network connection or other protocols.

△3 Ethernet interface 10/100Base-TX standard for MODBUS TCP network connection or other protocols (only for NX3020 and NX3030).

△4 Serial interface RS-232C standard for MODBUS RTU network connection or other protocols.

△5 Serial interface RS-485/RS-422 standard for MODBUS RTU network connection or other protocols. The physical interface choice depends on the cable used.

△6 The module is grounded through Nexto Series backplane rack.

△7 The power supply comes from the backplane rack connection. There is no need for external connections.

⏚ Protection earth terminal.

3.3. Ethernet Network Connection

The NET 1 isolated communication interface allows the connection with an Ethernet network, however, the NET 1 interface is the most suitable to be used for communication with MasterTool IEC XE.

The Ethernet network connection uses twisted pair cables (10/100Base-TX) and the speed detection is automatically made by the Nexto CPU. 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 NET 1 Ethernet interface is used for Ethernet communication and for CPU configuration which comes with the following default parameters configuration:

	NET 1
IP Address	192.168.15.1
Subnetwork Mask	255.255.255.0
Gateway Address	192.168.15.253

Table 20: Default Parameters Configuration for Ethernet NET 1 Interface

The IP Address and Subnet Mask parameters can be seen on the CPU graphic display via parameters menu, as described in CPU's Informative and Configuration Menu section.

Initially, 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 Interfaces Configuration section.

3.3.2. Gratuitous ARP

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

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

altus

3.3.3. Network Cable Installation

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

The table below present the RJ-45 Nexto CPU female connector, with the identification and description of the valid pinout for 10BASE-TE and 100BASE-TX physical levels.
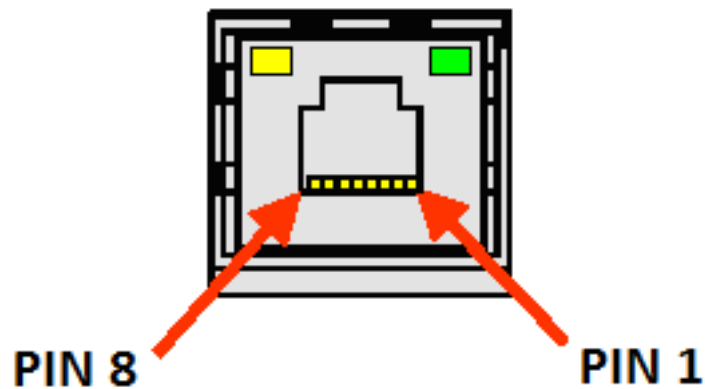
Figure 6: RJ45 Female Connector

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

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

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

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

3.4. Serial Network Connection RS-232C

The COM 1 non isolated communication interface allows the connection to a RS-232C network. As follows it's presented the DB9 female connector to Nexto CPU, with identification and sign description.
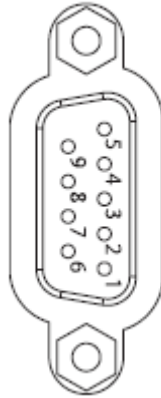
Figure 7: DB9 Female Connector

Pin	Sign	Description
1	DCD	Data Carrier Detect
2	TXD	Data Transmission
3	RXD	Data Reception
4	-	Not used
5	GND	Ground
6	-	Not used
7	CTS	Clear to Send
8	RTS	Request to Send
9	-	Not used

Table 22: COM 1 DB9 Female Connector Pin Layout

3.4.1. RS-232C Communication

For connection to a RS-232C device, use the appropriate cable as the section Related Products.

3.5. Serial Network Connection RS-485/422

The COM 2 isolated communication interface allow the connection to a RS-485/422 network. As follows it's presented the DB9 female connector to Nexto CPU, with identification and sign description.

altus
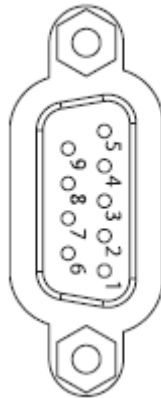
Figure 8: DB9 Female Connector

Pin	Sign	Description
1	-	Not used
2	Term+	Internal Termination, positive
3	TXD+	Data Transmission, positive
4	RXD+	Data Reception, positive
5	GND	Negative Reference for External Termination
6	+5V	Positive Reference for External Termination
7	Term-	Internal Termination, negative
8	TXD-	Data Transmission, negative
9	RXD-	Data Reception, negative

Table 23: COM 1 and COM 2 DB9 Female Connector Pin Layout

3.5.1. RS-485 Communication without termination

In order to connect in a RS-485 network with no termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	Device terminal signals
0	Shield
1	Not connected
2	D+
3	D+
4	Not connected
5	Not connected
6	Not connected
7	D-
8	D-

Table 24: RS-485 Connections without Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.
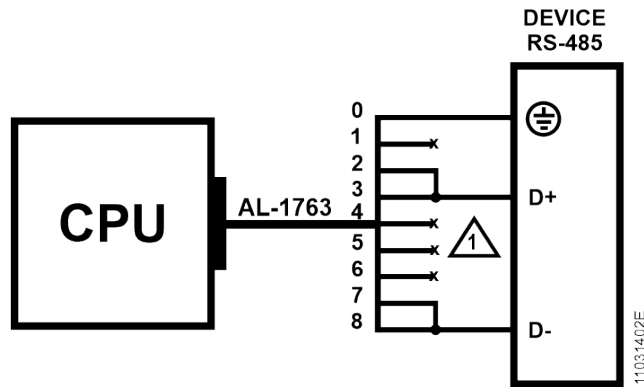
Figure 9: RS-485 Connections without Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.2. RS-485 Communication with Internal Termination

In order to connect in a RS-485 network using the internal termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	D+
2	D+
3	D+
4	Not connected
5	Not connected
6	D-
7	D-
8	D-

Table 25: RS-485 Connections with Internal Termination

PS.: The internal termination available is a safe state type in open mode.

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.
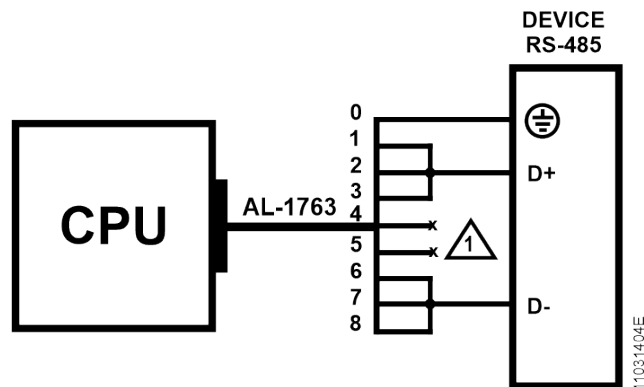
Figure 10: RS-485 Connections with Internal Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.3.   RS-485 Communication with External Termination

In order to connect to a RS-485 network wih external termination, the AL-1763 cable identified terminals must be connected in the respective device terminals according to the table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	Not connected
2	D+
3	D+
4	0 V
5	+5 V
6	Not connected
7	D-
8	D-

Table 26: RS-485 Connections with External Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.
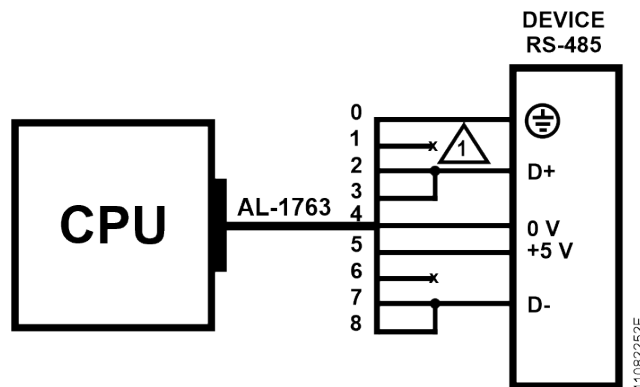
Figure 11: RS-485 Connections with External Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.4.   RS-422 Communication without Termination

In order to connect in a RS-422 network with no termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	Not connected
2	TX+
3	RX+
4	Not connected
5	Not connected
6	Not connected
7	TX-
8	RX-

Table 27: RS-422 Connections without Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.
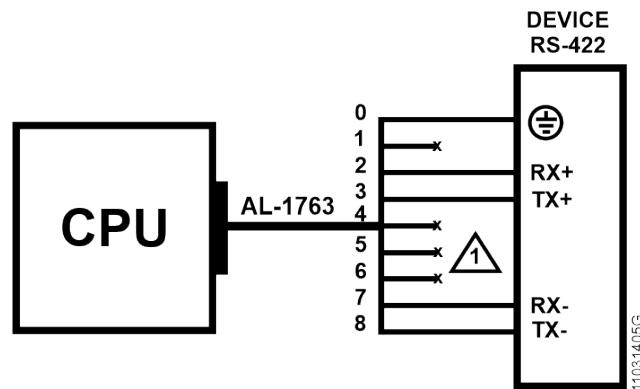
Figure 12: Connections without Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.5. RS-422 Communication with Internal Termination

In order to connect in a RS-422 network using the internal termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	TERM+
2	TX+
3	RX+
4	Not connected
5	Not connected
6	TERM-
7	TX-
8	RX-

Table 28: RS-422 Connections with Internal Termination

PS.: The internal terminations available are secure state in open mode.

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.
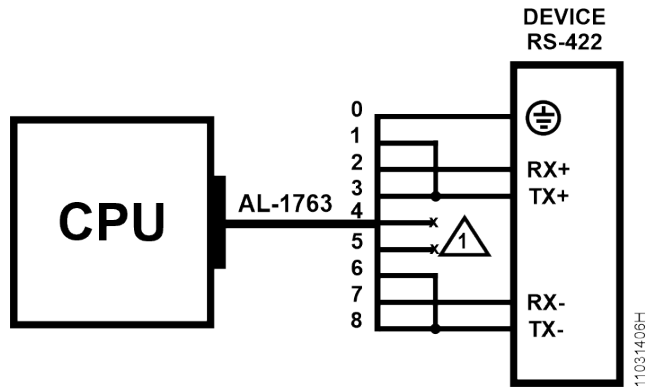
Figure 13: RS-422 Connections with Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.6. RS-422 Communication with External Termination

In order to connect in a RS-422 network using interface external termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 Terminals	CPU terminal signals
0	Shield
1	Not connected
2	TX+
3	RX+
4	0 V
5	+5 V
6	Not connected
7	TX-
8	RX-

Table 29: RS-422 Connections with External Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.
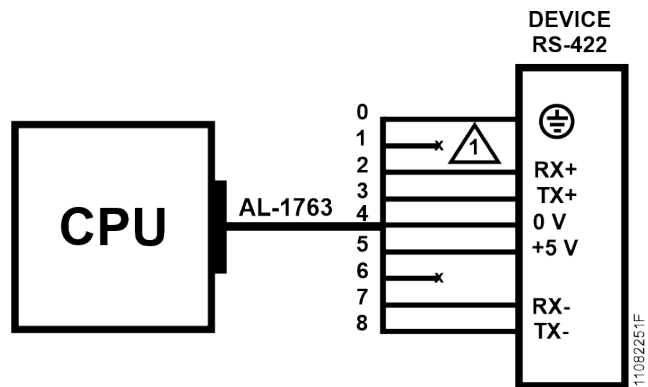
Figure 14: RS-422 Connections with External Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.7. RS-422 Network Example

The figure below shows an example of RS-422 network utilization, using the Nexto CPU as master, slave devices with RS-422 Interface, and Altus solutions for terminators and connections.
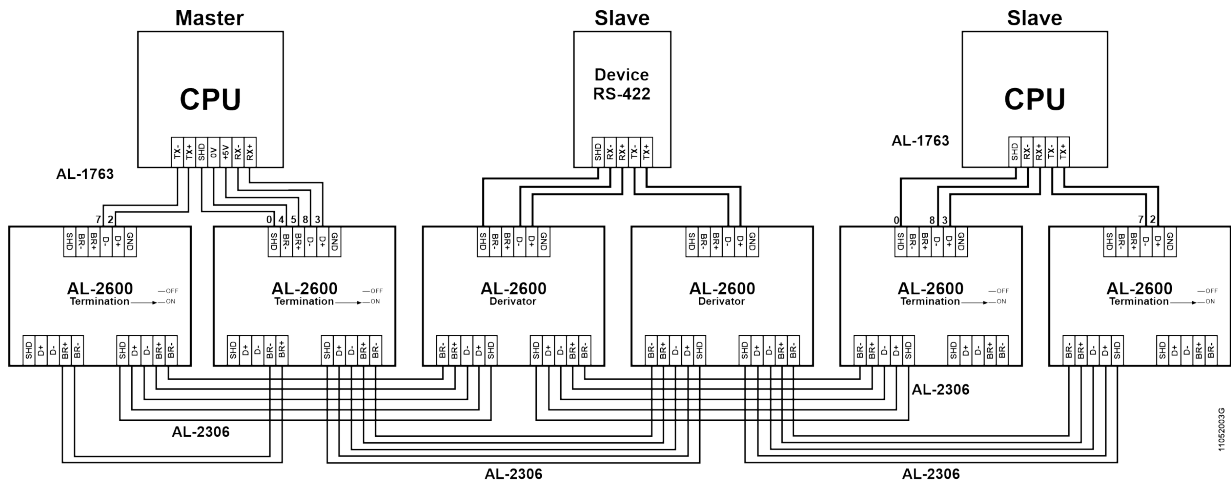
Figure 15: RS-422 Network Example

Diagram Note:

The AL-2600 modules which are in the network endings perform the terminators function. In this case the AL-2600 keys must be configured in PROFIBUS Termination.

3.6. Memory Card Installation

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

Initially, care must be taken with the correct position the memory card must be inserted. One corner of it is different from the other three and this one must be used as reference for the card correct insertion. Therefore, the memory card must be inserted following the depiction on the CPU frontal part or the way showed on figure below.
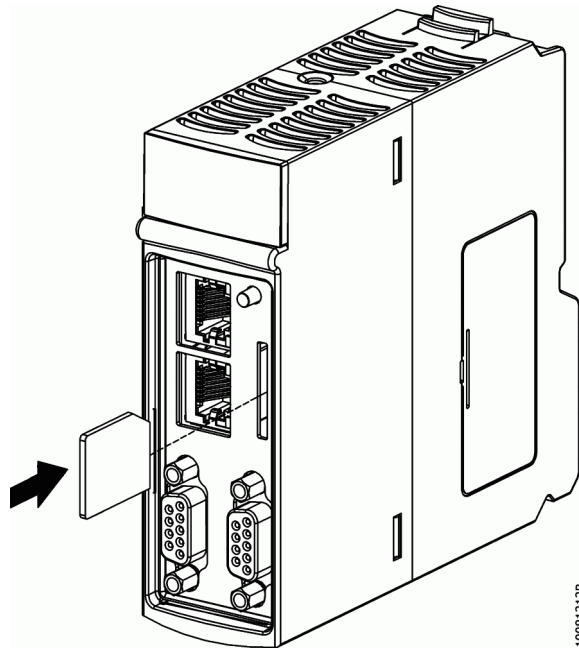
Figure 16: Memory Card Insertion in the CPU

When the card is correctly installed, a symbol will appear on the CPU graphic display. For card secure removing the MS key must be pressed then there is a little delay and the card symbol will disappear from the graphic display. The card is now ready to be taken off. For that, the card must be pressed against the CPU until a click is heard, then release it and withdraw it from the compartment as showed on figure below. At this moment the card will be loose.
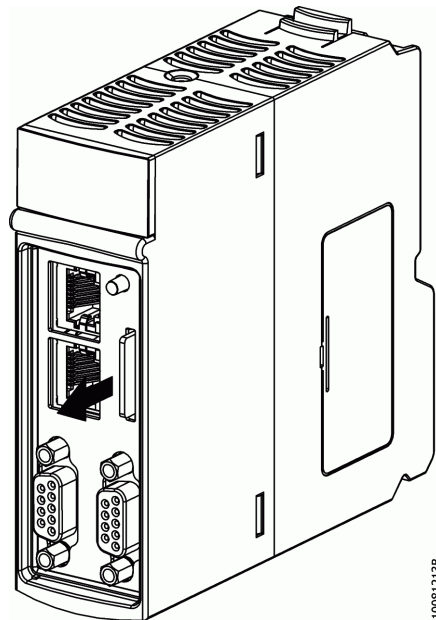
Figure 17: Memory Card Withdrawal

3.7.   Architecture Installation

3.7.1.   Module Installation on the Main Backplane Rack

Nexto Series has an exclusive method for connecting and disconnecting modules on the bus which does not require much effort from the operator and guarantee the connection integrity. For further information regarding Nexto Series products fixation, please see Nexto Series User Manual – MU214600.

3.8.   Programmer Installation

To execute the MasterTool IEC XE development software installation, it is necessary to have the distribution CD-ROM or download the installation file from the site https://www.altus.com.br/en/. For further information about the step by step to installation, consult MasterTool IEC XE User Manual MT8500 – MU299609.

4. Programmer Installation

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

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

The programming is made through the MasterTool IEC XE (IDE) development interface. The MasterTool IEC XE allows the use of the six languages in the same project, so the user can apply the best features offered by each language, resulting in more efficient applications development, for easy documentation and future maintenance.

For further information regarding programming, see MasterTool IEC XE User Manual - MU299609, MasterTool IEC XE Programming Manual - MP399609 or IEC 61131-3 standard.

4.1. Memory Organization and Access

Nexto Series uses an innovative memory organization and access feature called big-endian, where the most significant byte is stored first and will always be the smallest address (e.g. %QB0 will always be more significant than %QB1, as in table below, where, for CPUNEXTO string, the letter C is byte 0 and the letter O is the byte 7).

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

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

Table 30: Memory Organization and Access Example

	SIGNIFICANCE					OVERLAPPING			
Bit	Byte	Word	DWord	LWord	Byte	Word		DWord	
%QX0.7									
%QX0.6									
%QX0.5									
%QX0.4	%QB 00				%QB00				
%QX0.3									
%QX0.2									
%QX0.1									
%QX0.0		%QW 00				%QW 00			
%QX1.7									
%QX1.6									
%QX1.5					%QB01				
%QX1.4	%QB 01								
%QX1.3									
%QX1.2									
%QX1.1									
%QX1.0			%QD 00				%QW 00	%QD 00	
%QX2.7							01		
%QX2.6									
%QX2.5					%QB02				
%QX2.4	%QB 02								
%QX2.3									
%QX2.2									
%QX2.1									
%QX2.0		%QW 02				%QW 02		%QD 00	
%QX3.7							02	01	
%QX3.6									
%QX3.5					%QB03				
%QX3.4	%QB 03								
%QX3.3									
%QX3.2									
%QX3.1									
%QX3.0				%QL 00			%QW	%QD 01	
%QX4.7							03	02	
%QX4.6									
%QX4.5					%QB04				
%QX4.4	%QB 04								
%QX4.3									
%QX4.2									
%QX4.1									
%QX4.0		%QW 04				%QW 04		%QD 03	
%QX5.7							04	03	
%QX5.6									
%QX5.5					%QB05				
%QX5.4	%QB 05								
%QX5.3									
%QX5.2									
%QX5.1									
%QX5.0			%QD 04				%QW	%QD 04	
%QX6.7							05	04	
%QX6.6									
%QX6.5					%QB06				
%QX6.4	%QB 06								
%QX6.3									
%QX6.2									
%QX6.1									
%QX6.0		%QW 06				%QW 06			
%QX7.7							06		
%QX7.6									
%QX7.5					%QB07				
%QX7.4	%QB 07								
%QX7.3									
%QX7.2									
%QX7.1									
%QX7.0									

(Left labels: MSB ⇑ LSB)

Table 31: Memory Organization and Access

altus

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

4.2. Project Profiles

A project profile in the MasterTool IEC XE consists in an application template together with a group of verification rules which guides the development of the application, reducing the programming complexity. The applications can be created according the following profiles:

- Single
- Basic
- Normal
- Expert
- Custom
- 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, which follow a gradual complexity slope. Based in these definitions, it's recommended that the user always use the simplest profile that meets his application needs, migrating to a more sophisticated profile only when the corresponding rules are being more barriers to development than didactic simplifications. 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.

> **ATTENTION**
>
> When more than one task is used, the I/O access can only be done in the context of the MainTask. In case that the option Enable I/O Update per Task can't be used, present as of MasterTool IEC XE version 2.01.

4.2.1. Single

In the Single Project Profile, the application has only one user task, MainTask. This task is responsible for the execution of a single Program type programming unit called MainPrg. This single program can call other programming unit, of the Program, Function or Function Block types, but the whole code will be executed exclusively by the MainTask.

In this profile, the MainTask will be of the cyclical type (Cyclic) with priority fixed as 13 (thirteen) and runs exclusively the MainPrg program in a continuous loop. The MainTask is already fully defined and the developer needs to create the MainPrg program, using any of the languages of the IEC 61131-3 standard. It is not always possible to convert a program to another language, but it's always possible to create a new program, built in a different language, with the same name and replace it. The MasterTool IEC XE standard option is to use the MasterTool Standard Project associated with the Single profile, which also include the MainPrg created in the language selected during the project creation.

This type of application never needs to consider issues as data consistence, resource sharing or mutual exclusion mechanisms.

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

Table 32: Single Profile Task

4.2.2. Basic

In the Basic Project Profile, the application has one user task of the Continuous type called MainTask, which executes the program in a continuous loop (with no definition of cycle time) with priority fixed in 13 (thirteen). This task is responsible for the execution of a single programming unit POU called MainPrg. It's important to notice that the cycle time may vary according to the quantity of communication tasks used, as in this mode, the main task is interrupted by communication tasks.

This profile also allows the inclusion of two event tasks with higher priority, that can interrupt (preempt) the MainTask at any given moment: the task named ExternInterruptTask00 is an event task of the External type with priority fixed in 02 (two); the task named TimeInterruptTask00 is an event task of the Cyclic type with priority fixed as 01 (one).

The Basic project template model includes three tasks already completely defined as presented in table below. The developer need only to create the associated programs.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Continuous	-	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-

Table 33: Basic Profile Tasks

4.2.3. Normal

In the Normal Project Profile, the application has one user task of the Cyclic type, called MainTask. This task is responsible for the execution of a single programming unit POU called MainPrg. This program and this task are similar to the only task and only program of the Single profile, but here the application can integrate additional user tasks. These other tasks, named CyclicTask00 and CyclicTask01, each one responsible for the exclusive execution of its respective CyclicPrg<nn> program. The CyclicTask<nn> tasks are always of the cyclic type and with priority fixed in 13 (thirteen), same priority as MainTask. These two types form a group called basic tasks, which associated programs can call other POUs of the Program, Function and Function Block types.

Furthermore, this profile can include event tasks with higher priority than the basic tasks, which can interrupt (preempt) these tasks execution at any time.

The task called ExternInterruptTask00 is an event task of the External type which execution is triggered by some external event, such as the variation of a control signal on a serial port or the variation of a digital input on the NEXTO bus. This task priority is fixed in 02 (two), being responsible exclusively for the execution of the ExternInterruptPrg00 program. The task called TimeInterruptTask00 is an event task of the Cyclic type with a priority fixed as 01 (one), being responsible for the execution exclusively of TimeInterruptPrg00 program.

In the Normal project model, there are five tasks, and its POUs, already fully defines as shown in table below. The developer needs only to implement the programs content, opting, on the wizard, for any of the languages in IEC 61131-3 standard. The tasks interval and trigger events can be configured by the developer and the unnecessary tasks can be eliminated.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
CyclicTask00	CyclicPrg00	13	Cyclic	200 ms	-
CyclicTask01	CyclicPrg01	13	Cyclic	500 ms	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-

Table 34: Normal Profile Tasks

4.2.4. Expert

The Expert Project Profile includes the same basic tasks, CyclicTask<nn>, ExternInterruptTask00 and TimeInterruptTask00 with the same priorities (13, 02 and 01 respectively), but it's an expansion from the previous ones, due to accept multiple events tasks. That is, the application can include various ExternInterruptTask<nn> or TimeInterruptTask<nn> tasks that execute the ExternInterruptPrg<nn> and TimeInterruptPrg<nn> programs. The additional event tasks priorities can be freely selected from 08 to 12. In this profile, besides the standard programs, each task can execute additional programs.

In this project profile, the application may also include the user task FreeTask of the Freewheeling type with priority 31, responsible for the FreePrg program execution. As this task is low priority it can be interrupted by all others so it can execute codes that might be blocked.

There are eight tasks already fully defined, as shown in table below, as well as their associated programs in the chosen language. Intervals and trigger events of any task, as well as the priorities of the event tasks can be configured by the user.

When developing the application using Expert project's profile, a special care is needed with the event tasks scaling. If there is information and resource sharing between these tasks or between them and the basic tasks, it is strongly recommended to adopt strategies to ensure data consistency.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
CyclicTask00	CyclicPrg00	13	Cyclic	200 ms	-
CyclicTask01	CyclicPrg01	13	Cyclic	500 ms	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-
ExternInterruptTask01	ExternInterruptPrg01	11	External	-	IO_EVT_1
TimeInterruptTask01	TimeInterruptPrg01	09	Cyclic	30 ms	-
FreeTask	FreePrg	31	Continuous	-	-

Table 35: Expert Profile Tasks

4.2.5. Custom

The Custom project profile allows the developer to explore all the potential of the Runtime System implemented in the CPUs. No functionality is disabled; no priority, task and programs association or nomenclatures are imposed. The only exception is for MainTask, which must always exist with this name in this Profile.

Beyond the real time tasks, with priority between 00 and 15, which are scheduled by priority, in this profile it is also possible to define tasks with lower priorities in the range 16 to 31. In this range, it's used the Completely Fair Scheduler (time sharing), which is necessary to run codes that can be locked (for example, use of sockets).

The developer is free to partially follow or not the organization defined in other project profiles, according to the characteristics of the application. On the other hand, the Custom model associated with this profile needs no pre-defining elements such as task, program or parameter, leaving the developer to create all the elements that make up the application.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
CyclicTask00	CyclicPrg00	13	Cyclic	200 ms	-
CyclicTask01	CyclicPrg01	13	Cyclic	500 ms	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-
ExternInterruptTask01	ExternInterruptPrg01	11	External	-	IO_EVT_1
TimeInterruptTask01	TimeInterruptPrg01	09	Cyclic	30 ms	-
FreeTask	FreePrg	31	Continuous	-	-

Table 36: Custom Profile Tasks

4.2.6. 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.

This profile is characterized by allowing shorter intervals in the MainTask, allowing faster execution of user code. This optimization is possible because MainTask also performs the processing of the bus. This way, different from other profiles, the machine profile requires no context switch for the bus treatment, which reduces the overall processing time.

altus

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.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	4 ms	-

Table 37: Machine Profile Tasks

Also, this profile supports the inclusion of additional tasks associated to the external interruptions.

4.2.7. General Table

		Project Profiles					
		Single	**Machine**	**Basic**	**Normal**	**Expert**	**Custom**
Total tasks		01	04	[01..03]	[01..32]	[01..32]	[01..32]
Tasks per program		01		01	01	<n>	<n>
Main Task	**Type**	Cyclic	Cyclic	Continuous	Cyclic	Cyclic	Cyclic
	Priority	13	13	13	13	13	13
	Quantity	01	01	01	01	01	01
Time Interrupt Task	**Type**		Cyclic	Cyclic	Cyclic	Cyclic	Cyclic
	Priority		01	01	01	01 or [08..12]	01 or [08..12]
	Quantity		[00..01]	[00..01]	[00..01]	[00..31]	[00..31]
Extern Interrupt Task	**Type**		External	External	External	External	External
	Priority		02	02	02	02 or [08..12]	02 or [08..12]
	Quantity		[00..01]	[00..01]	[00..01]	[00..31]	[00..31]
Ciclic Task	**Type**				Cyclic	Cyclic	Cyclic
	Priority				13	13	13
	Quantity				[00..31]	[00..31]	[00..31]
Free Task	**Type**					Continuous	Continuous
	Priority					31	31
	Quantity					[00..01]	[00..01]
Event Task	**Type**						Event
	Priority						<n>
	Quantity						[00..31]

Table 38: General Profile x Tasks Table

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.

4.2.8. Maximum Number of Tasks

The maximum number of tasks that the user can create is only defined for the Custom profile, the only one which has this permission. The others already have their tasks created and configured. However, the tasks that will be created must use the following prefixes, according to the type of each of the tasks: CyclicTaskxx, TimeInterruptTaskxx, ExternInterruptTaskxx, where xx represents the number of the task that being created.

The table below describes the maximum IEC task quantity per CPU and project profile, where the protocol instances are also considered communication tasks by the CPU.

Task Type		NX3010					
		S	B	N	E	P	M
Configuration Task (WHSB Task)	Cyclic	1	1	1	1	1	0
User Tasks	Cyclic	1	1	15	15	15	2
	Triggered by Event	0	0	0	0	15	0
	Disp. External Event	0	1	1	14	15	0
	Freewheeling	0	1	0	1	15	0
	State-triggered	0	0	0	0	15	0
NETs - Client or Server Instances	Cyclic	4					
COM (n) - Master or Slave Instances	Cyclic	1					
TOTAL		16					

Table 39: NX3010 IEC Tasks Maximum Number

Notes:

Profile Legend: The S, B, N, E, C and M letters correspond to the Single, Basic, Normal, Expert, Custom and Machine profiles respectively.

Values: The number defined for each task type represents the maximum values allowed.

Task WHSB: The WHSB is a system task that must be considered so the total value is not surpassed.

NETs - Client or Server Instances: The maximum value defined considers all system Ethernet interfaces, including the expansion modules when these are applied. E.g. MODBUS protocol instances.

COM (n) - Master or Slave Instances: The "n" represents the number of the serial interface. Even with expansion modules, the table value will be the maximum per interface. E.g. MODBUS protocol instances.

Total: The total value does not represent the sum of all profile tasks, but the maximum value allowed per CPU. Therefore, the user can create several task types, while the established numbers for each one and the total value are not surpassed.

4.3. CPU Configuration

The Nexto CPU configuration is located in the device tree, as shown on figure below, and can be accessed by a double-click on the corresponding object. In this tab it's possible to configure the diagnostics area, the retentive and persistent memory area and hot swap mode, among other parameters, as described in the CPU Configuration.
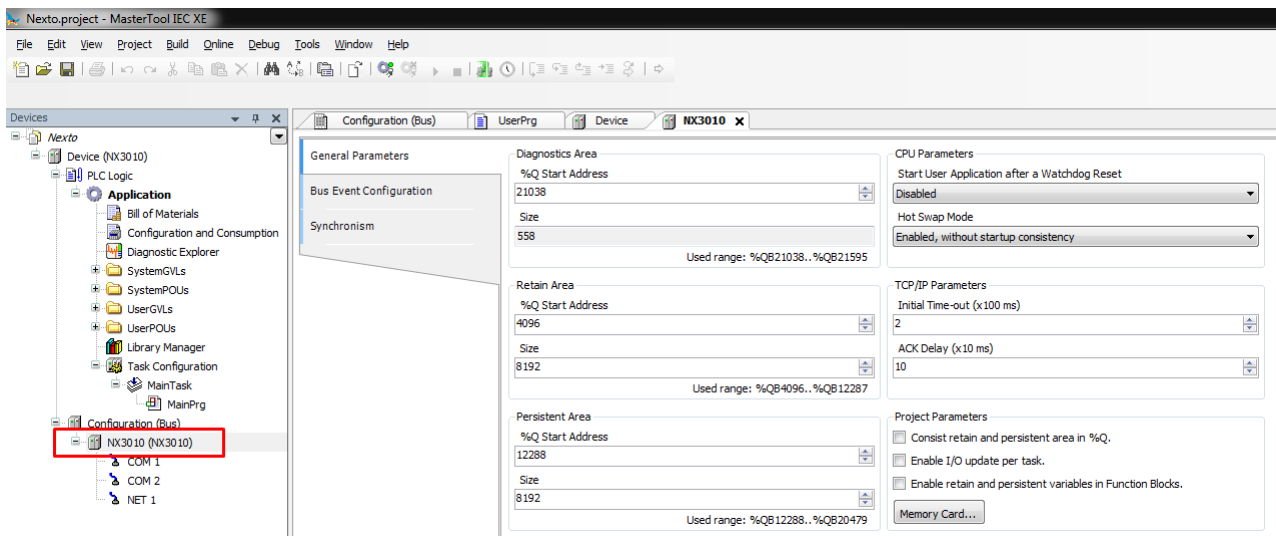
Figure 18: CPU Configuration

Besides that, by double-clicking on CPU's NET 1 icon, it's possible to configure the Ethernet interface that will be used for communication between the controller and the software MasterTool IEC XE.
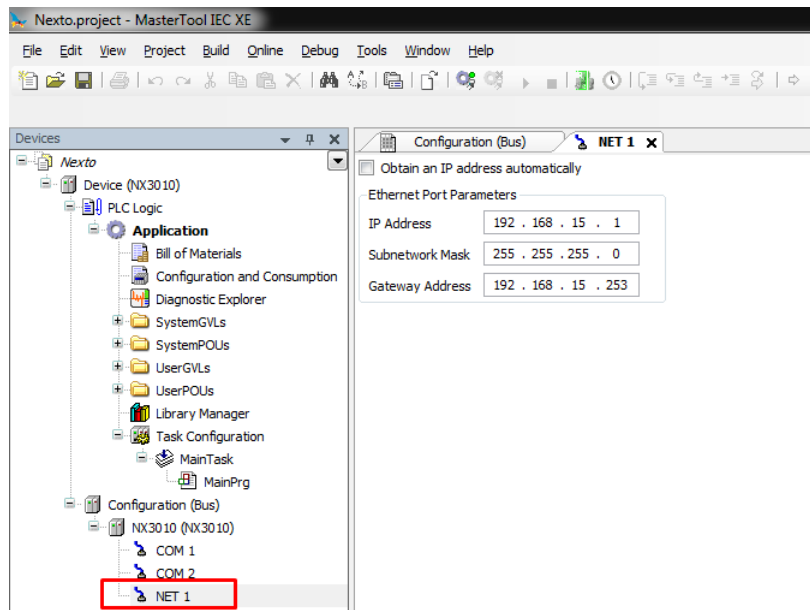
Figure 19: Configuring the CPU Communication Port

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.

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 must be found in the MasterTool Programming Manual – MP399609.

4.5. Inserting a Protocol Instance

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

4.5.1. MODBUS Ethernet

The first step to configure the MODBUS Ethernet (Client in this example), is to include the instance in the desired NET (in this case, NET 1, as the CPU NX3010 has only one Ethernet interface). Click on the *NET* with the mouse right button and select *Add Device...*, as shown on figure below.
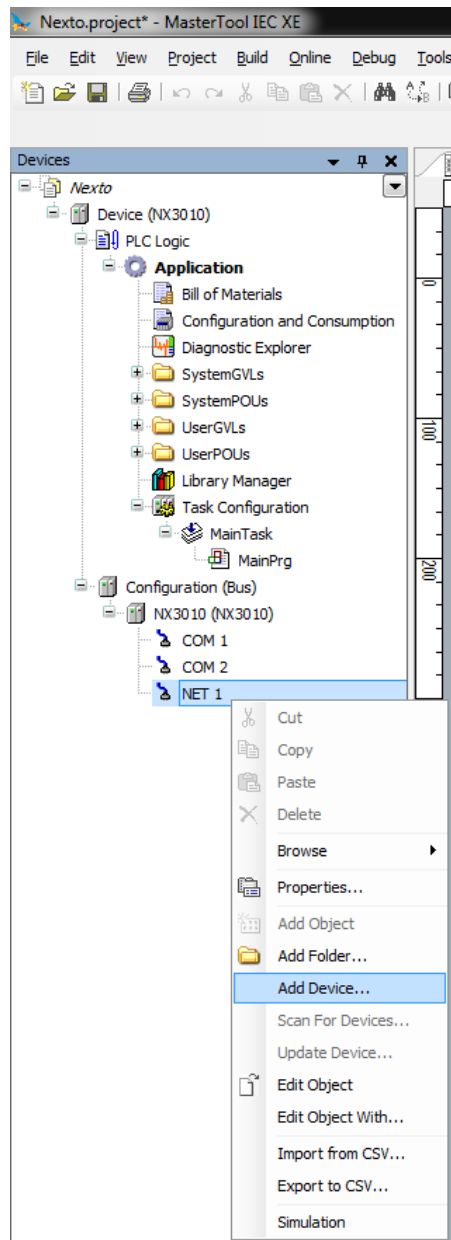
Figure 20: Adding the Instance

After that, the available protocols for the user will appear on the screen. In this menu is defined the configuration mode of the protocol. Selecting the option *MODBUS Symbol Client*, for Symbolic Mapping setting or *MODBUS Client*, for Direct Addressing (%Q). Then, click *Add Device*, as shown in the figure below.
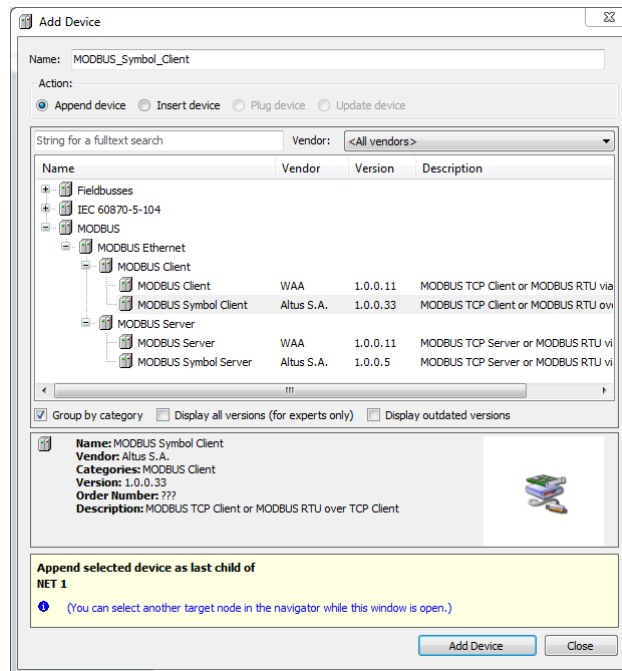
Figure 21: Selecting the Protocol

4.6. Finding the Device

To establish the communication between the CPU 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 CPUs found on the network of the Ethernet interface of the station where the tool is running.
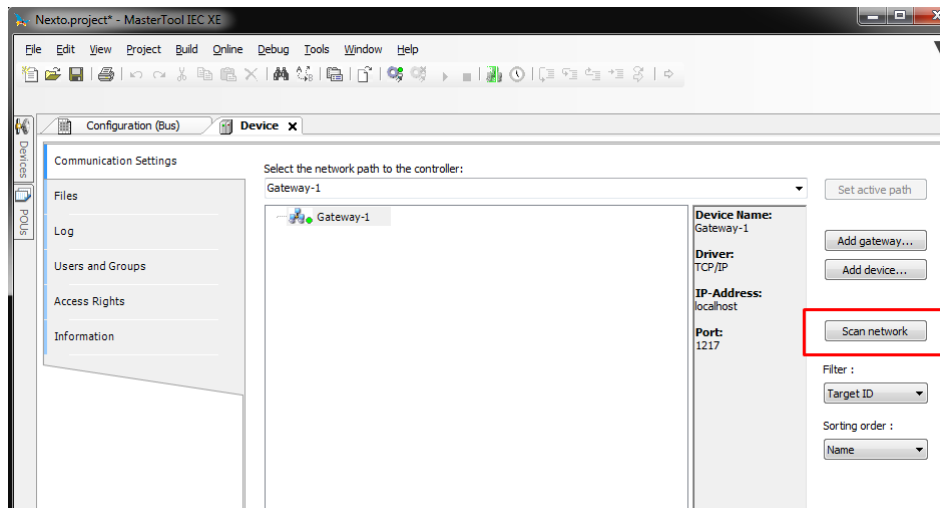
Figure 22: Finding the CPU

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

Next, the desired controller must be selected by clicking on *Set active path*. This action selects the controller and informs the configuration software which controller shall be used to communicate and send the project.
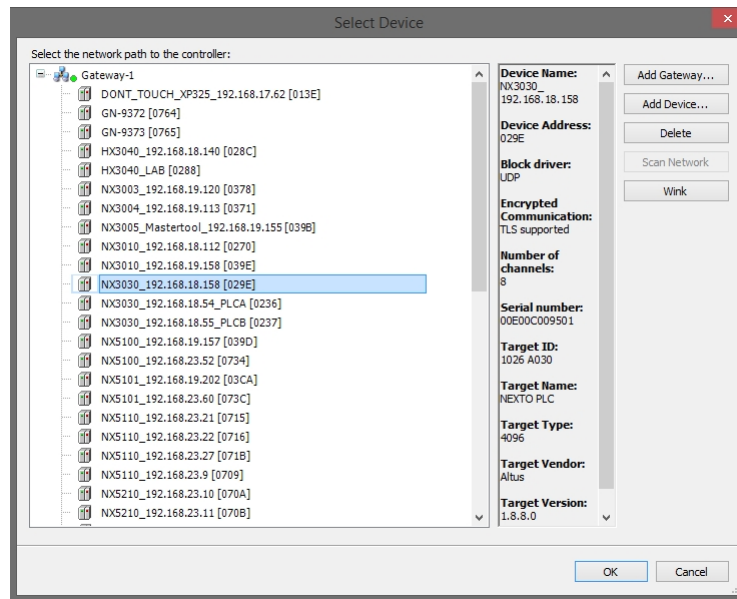
Figure 23: Selecting the CPU

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

In case the Ethernet configuration of the CPU to be connected is in a different network from the Ethernet interface of the station, 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.
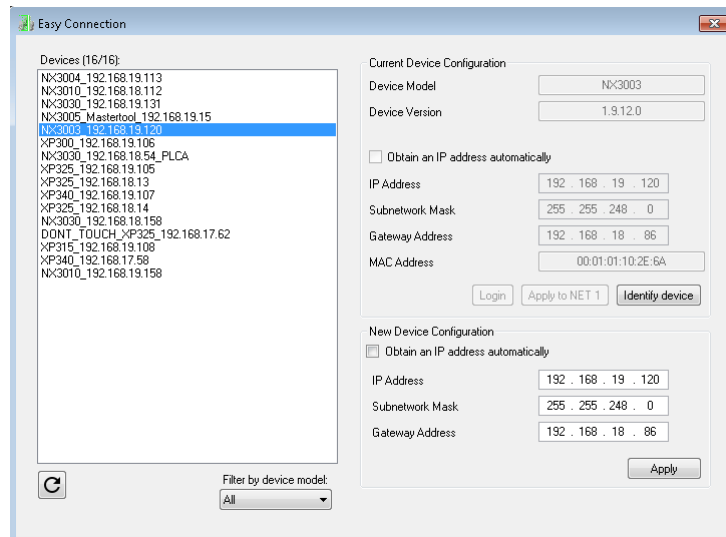
Figure 24: Easy Connection

This command performs a MAC level communication with the NET 1 interface of the device, allowing to permanently change the configuration of the CPU's Ethernet interface, independently of the IP configuration of the station and from the one previously configured on the device. So, with this command, it's possible to change the device configuration to put it on the same network of the Ethernet interface of the station where MasterTool 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 CPU. 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.
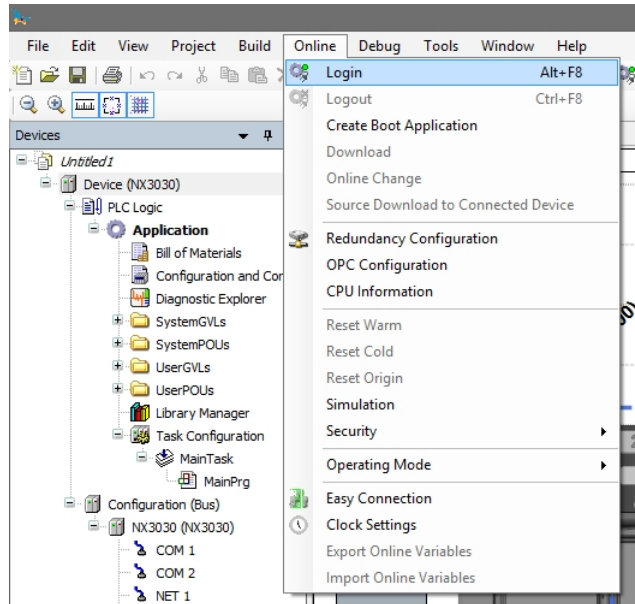
Figure 25: Sending the Project to the CPU

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

If the Ethernet configuration of the project is different from the device, the communication may be interrupted at the end of download process when the new configuration is applied on the device. So, the following warning message will be presented, asking the user to proceed or not with this operation.
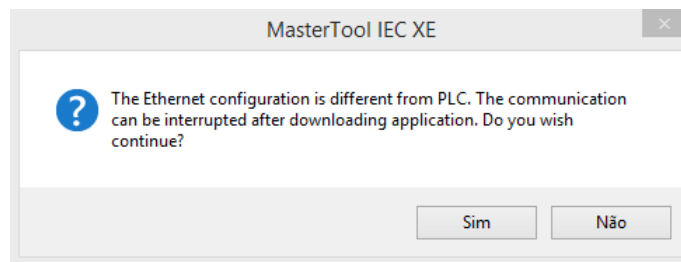
Figure 26: IP Configuration Warning

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

Figure 27: No application on the device

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

- **Login with online change:** execute the login and send the new project without stopping the current CPU application (see Run Mode item), updating the changes when a new cycle is executed.
- **Login with download:** execute the login and send the new project with the CPU stopped (see Stop Mode). When the application is initiated, the update will have been done already.
- **Login without any change:** executes the login without sending the new project.
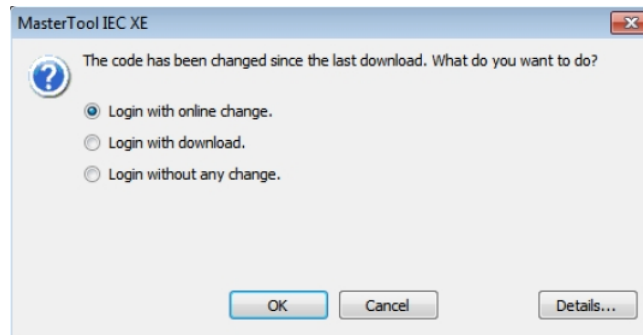
Figure 28: Project Update at the CPU

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

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

PS.: The button *Details...* shows the changes made in the application.
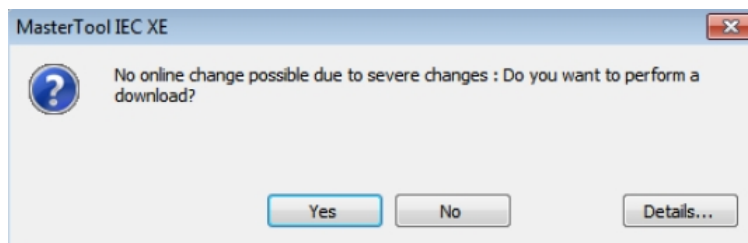
Figure 29: Configuration Changes

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:
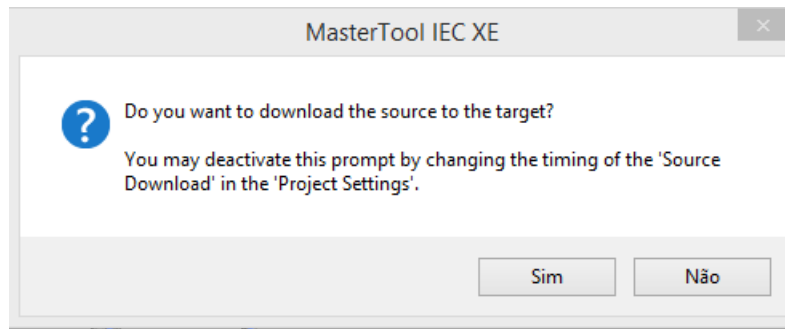
Figure 30: Source code download

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

4.8.   Run Mode

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

To execute this command, simply go to the *Debug* menu and select the option *Start*, as shown on figure below.
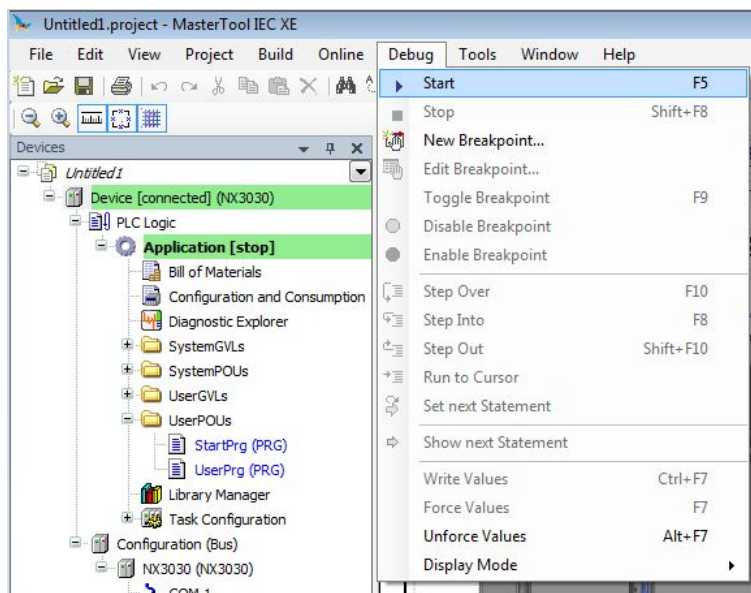
Figure 31: Starting the Application

The figure below shows the application in execution. In case the POU tab is selected, the created variables are listed on a monitoring window, in which the values can be visualized and forced by the user.
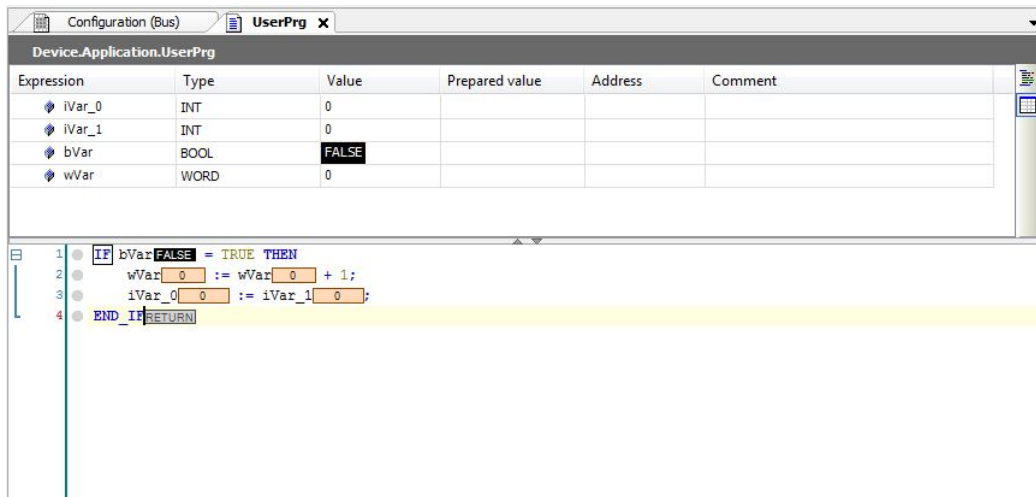
Figure 32: Program running

If the CPU already have a boot application internally stored, it goes automatically to Run Mode when the device is powered on, with no need for an online command through MasterTool 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 below.
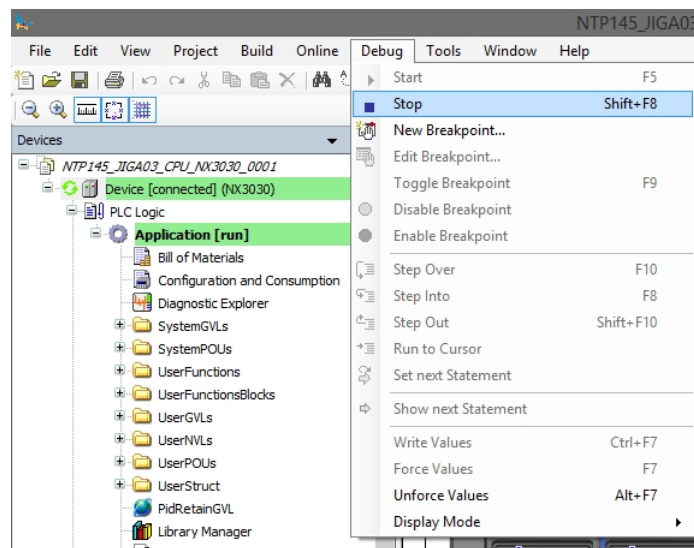
Figure 33: Stopping the Application

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

4.10. Writing and Forcing Variables

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

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

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

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

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

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

4.11. Logout

To finalize the online communication with the CPU, the command *Logout* must be executed, located in the *Online* menu, as shown on figure below.
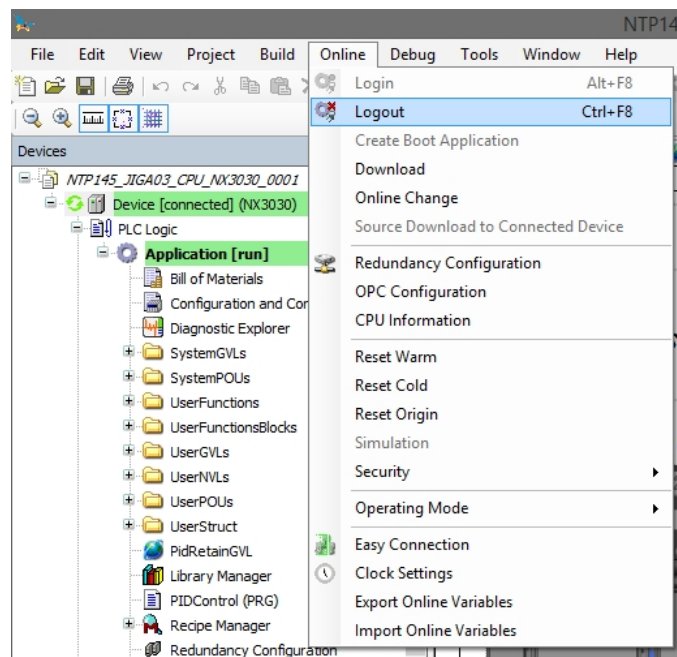
Figure 34: Finalizing the online communication with the CPU

4.12. Project Upload

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

To recover a project previously stored on the internal memory of the CPU, the command located on *File* menu must be executed as shown on the following figure.
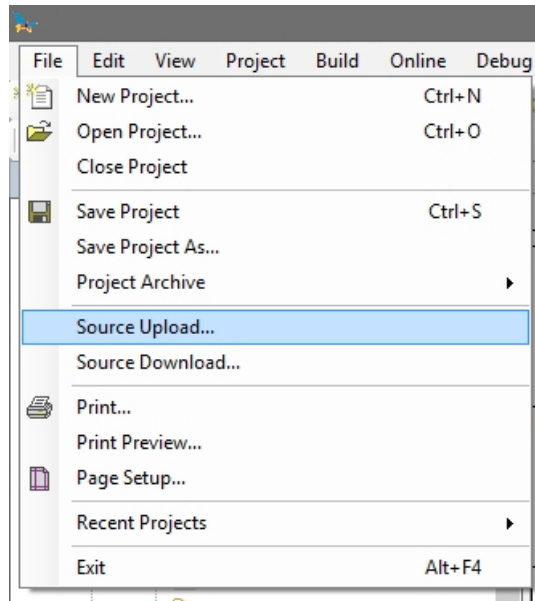
Figure 35: Project Upload Option

Next, just select the desired CPU and click *OK*, as shown on figure below.
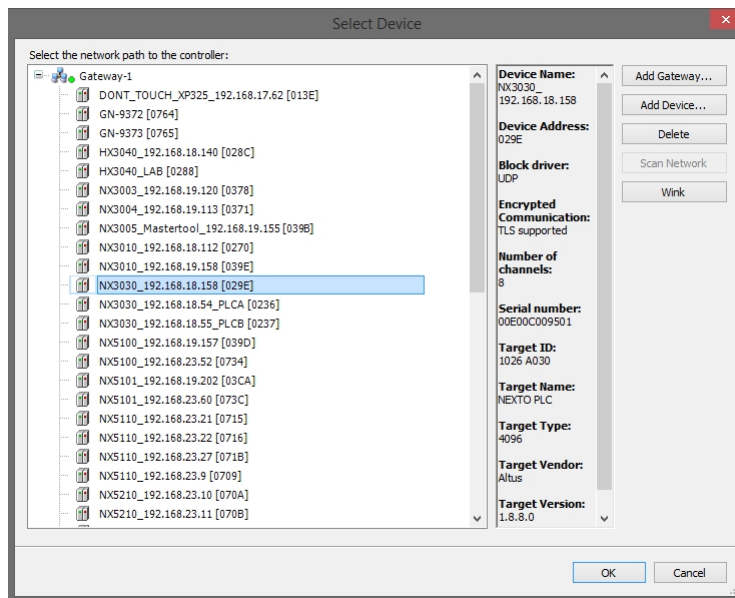
Figure 36: Selecting the CPU

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

ATTENTION

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

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

4.13.   CPU Operating States

4.13.1.   Run

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

4.13.2.   Stop

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

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

4.13.3.   Breakpoint

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

For further information about the use of breakpoints, please consult 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 Origen

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

Notes:

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

Command: To execute the commands *Reset Warm*, *Cold* or *Origin*, it's necessary to have MasterTool IEC XE in *Online* mode with 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 describe the main elements that are part of this standard project structure.

4.14.1. Programa MainPrg

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

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

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

PROGRAM MainPrg
VAR
    isFirstCycle : BOOL := TRUE;
END_VAR

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

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

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

4.14.2. Programa StartPrg

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. Programa UserPrg

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 System_Diagnostics

The *System_Diagnostics* GVL contains the diagnostic variables of the CPU, of the communication interface (Ethernet and PROFIBUS) and of all communication drivers. 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 Requests.

Some devices, like the MODBUS Symbol communication driver, doesn't have its diagnostics allocated at %Q variables with the AT directive. The same occurs with newest communication drivers, as Server IEC 60870-5-104.

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

Device.Application.System_Diagnostics				
Expression	Type	Value	Address	Con
⊞ 🌐 DG_IEC_60870_5_104_Server	T_DIAG_IEC104_SERVER_1			DG_I
⊟ 🌐 DG_MODBUS_Symbol_Client	T_DIAG_MODBUS_ETH_CLIENT_1			DG_N
⊞ ◆ tDiag	T_DIAG_MODBUS_DIAGNOSTICS_CLIENT			
◆ byDiag_1_reserved	BYTE	0		Rese
⊞ ◆ tCommand	T_DIAG_MODBUS_COMMANDS			
◆ byDiag_3_reserved	BYTE	0		Rese
⊟ ◆ tStat	T_DIAG_MODBUS_ETH_CLIENT_STATS			
◆ wTXRequests	WORD	1589		Coun
◆ wRXNormalResponses	WORD	1589		Coun
◆ wRXExceptionResponses	WORD	0		Coun
◆ wRXIllegalResponses	WORD	0		Coun
◆ wDiag_12_reserved	WORD	0		Rese
◆ wDiag_14_reserved	WORD	0		Rese
◆ wDiag_16_reserved	WORD	0		Rese
◆ wDiag_18_reserved	WORD	0		Rese
⊞ 🌐 DG_MODBUS_Symbol_Client_NX5000	T_DIAG_MODBUS_ETH_CLIENT_1			DG_N
⊞ 🌐 DG_MODBUS_Symbol_RTU_Master	T_DIAG_MODBUS_RTU_MASTER_1			DG_N
⊞ 🌐 DG_MODBUS_Symbol_Server_NX5000	T_DIAG_MODBUS_ETH_SERVER_1			DG_N
⊟ 🌐 DG_NX3030	T_DIAG_NX3030_1		%QB66229	DG_N
⊞ ◆ tSummarized	T_DIAG_SUMMARIZED_1			
⊞ ◆ tDetailed	T_DIAG_DETAILED_1			
⊞ 🌐 DG_NX5001	T_DIAG_NX5001_1		%QB66922	DG_N
⊞ 🌐 DG_MODBUS_Client	T_DIAG_MODBUS_ETH_CLIENT_1		%QB67191	DG_N
⊟ 🌐 DG_MBUS_Direct_1_Mapping_000	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67211	DG_N
⊟ ◆ byStatus	T_DIAG_MODBUS_ETH_MAPPING_STAT...			
◆ bCommIdle	BIT	FALSE		Comm
◆ bCommExecuting	BIT	FALSE		Comm
◆ bCommPostponed	BIT	TRUE		Comm
◆ bCommDisabled	BIT	FALSE		Comm
◆ bCommOk	BIT	TRUE		Previ
◆ bCommError	BIT	FALSE		Previ
◆ bCommAborted	BIT	FALSE		Previ
◆ bDiag_7_reserved	BIT	FALSE		Rese
◆ eLastErrorCode	MASTER_ERROR_CODE	NO_ERROR		Last
◆ eLastExceptionCode	MODBUS_EXCEPTION	NO_EXCEPTION		Last
◆ byDiag_3_reserved	BYTE	0		reser
◆ wCommCounter	WORD	397		Coun
◆ wCommErrorCounter	WORD	0		Coun
⊞ 🌐 DG_MBUS_Direct_1_Mapping_001	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67219	DG_N
⊞ 🌐 DG_MBUS_Direct_1_Mapping_003	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67235	DG_N
⊞ 🌐 DG_MBUS_Direct_1_Mapping_002	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67243	DG_N
⊞ 🌐 DG_NX5000	T_DIAG_NX5000_1		%QB67251	DG_N

Figure 37: System_Diagnostics GVL in Online Mode

4.14.5. GVL Disables

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

Requisition disabling variables declaration:

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

Where:

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

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

Example:

Device.Application.Disables

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

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

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

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

Figure 38: Disable GVL in Online Mode

4.14.6. GVL IOQualities

The *IOQualities* GVL contains the quality variables of I/O modules declared on CPU's bus. This GVL is not editable and the variables are automatically declared as *LibDataTypes.QUALITY* type arrays, and dimensions according to I/Os quantities of the module to which it belongs when that is added to the project.

Example: Device.Application.IOQualities

```
VAR_GLOBAL
    QUALITY_NX1001: ARRAY[0..15]  OF LibDataTypes.QUALITY;
    QUALITY_NX2020: ARRAY[0..15]  OF LibDataTypes.QUALITY;
    QUALITY_NX6000: ARRAY[0..7]   OF LibDataTypes.QUALITY;
    QUALITY_NX6100: ARRAY[0..3]   OF LibDataTypes.QUALITY;
END_VAR
```

Once the application is in *RUN* it is possible to watch the I/O modules quality variables values that were added to the project through *IOQualities* GVL.

4.14.7. GVL Module_Diagnostics

The *Module_Diagnostics* GVL contains the diagnostics variables of the I/O modules used in the project, except by the CPU and communication drivers. This GVL isn't editable and the variables are automatically declared with type specified by the module, to which it belongs, when that is added to the project.

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

Expression	Type	Value	Address	Comment
Device.Application.Module_Diagnostics				
⊟ 🌐 DG_NX1001	T_DIAG_NX1001_1		%QB67008	DG_NX1001 diagnostics variable
⊟ ◆ tGeneral	T_DIAG_GENERAL_NX1001_1			
◆ bReserved_8	BIT	FALSE		Reserved
◆ bReserved_9	BIT	FALSE		Reserved
◆ bReserved_10	BIT	FALSE		Reserved
◆ bReserved_11	BIT	FALSE		Reserved
◆ bReserved_12	BIT	FALSE		Reserved
◆ bReserved_13	BIT	FALSE		Reserved
◆ bReserved_14	BIT	FALSE		Reserved
◆ bReserved_15	BIT	FALSE		Reserved
◆ bActiveDiagnostics	BIT	FALSE		Module has active diagnostics
◆ bFatalError	BIT	FALSE		Module has fatal error
◆ bConfigMismatch	BIT	FALSE		Module has parameterization error
◆ bWatchdogError	BIT	FALSE		Module has watchdog expired
◆ bOTDSwitchError	BIT	FALSE		Module one touch diag switch error
◆ bReserved_5	BIT	FALSE		Reserved
◆ bReserved_6	BIT	FALSE		Reserved
◆ bReserved_7	BIT	FALSE		Reserved
⊞ 🌐 DG_NX1005	T_DIAG_NX1005_1		%QB67010	DG_NX1005 diagnostics variable
⊞ 🌐 DG_NX2001	T_DIAG_NX2001_1		%QB67014	DG_NX2001 diagnostics variable
⊞ 🌐 DG_NX2020	T_DIAG_NX2020_1		%QB67018	DG_NX2020 diagnostics variable
⊞ 🌐 DG_NX6000	T_DIAG_NX6000_1		%QB67022	DG_NX6000 diagnostics variable
⊟ 🌐 DG_NX6100	T_DIAG_NX6100_1		%QB67040	DG_NX6100 diagnostics variable
⊟ ◆ tGeneral	T_DIAG_GENERAL_NX6100_1			
◆ bActiveDiagnosticsOutput00	BIT	FALSE		Output 00 with diagnostics
◆ bActiveDiagnosticsOutput01	BIT	FALSE		Output 01 with diagnostics
◆ bActiveDiagnosticsOutput02	BIT	FALSE		Output 02 with diagnostics
◆ bActiveDiagnosticsOutput03	BIT	FALSE		Output 03 with diagnostics
◆ bReserved_12	BIT	FALSE		Reserved
◆ bReserved_13	BIT	FALSE		Reserved
◆ bReserved_14	BIT	FALSE		Reserved
◆ bReserved_15	BIT	FALSE		Reserved
◆ bActiveDiagnostics	BIT	FALSE		Module has active diagnostics
◆ bFatalError	BIT	FALSE		Module has fatal error
◆ bConfigMismatch	BIT	FALSE		Module has parameterization error
◆ bWatchdogError	BIT	FALSE		Module has watchdog expired
◆ bOTDSwitchError	BIT	FALSE		Module one touch diag switch error
◆ bCalibrationError	BIT	FALSE		Module has calibration error
◆ bNoExternalSupply	BIT	FALSE		External power s...y is below the ...
◆ bReserved_07	BIT	FALSE		Reserved
⊟ ◆ tDetailed	T_DIAG_DETAILED_NX6100_1			
⊞ ◆ tAnalogOutput_00	T_DIAG_ANALOG_OUTPUT			
⊞ ◆ tAnalogOutput_01	T_DIAG_ANALOG_OUTPUT			
⊞ ◆ tAnalogOutput_02	T_DIAG_ANALOG_OUTPUT			

Figure 39: Module_Diagnostics GVL in Online Mode

4.14.8. GVL ReqDiagnostics

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

Requisition diagnostic variable declaration:

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

Where:

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

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

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

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

Example:

Device.Application.ReqDiagnostics

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

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

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

Device.Application.ReqDiagnostics		
Expression	Type	Value
⊟ 🌐 MODBUS_Slave_1_REQDG_0001	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊞ ◈ byStatus	T_DIAG_MODBUS_RTU_MAPPING_STATUS	
◈ eLastErrorCode	MASTER_ERROR_CODE	NO_ERROR
◈ eLastExceptionCode	MODBUS_EXCEPTION	NO_EXCEPTION
◈ byDiag_3_reserved	BYTE	0
◈ wCommCounter	WORD	969
◈ wCommErrorCounter	WORD	0
⊞ 🌐 MODBUS_Slave_1_REQDG_0002	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊞ 🌐 MODBUS_Slave_1_REQDG_0003	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊞ 🌐 MODBUS_Slave_1_REQDG_0004	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊞ 🌐 MODBUS_Server_1_REQDG_0001	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊞ 🌐 MODBUS_Server_1_REQDG_0002	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊟ 🌐 MODBUS_Server_1_REQDG_0003	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
⊞ ◈ byStatus	T_DIAG_MODBUS_ETH_MAPPING_STATUS	
◈ eLastErrorCode	MASTER_ERROR_CODE	ERR_CONNECTION_TIMEOUT
◈ eLastExceptionCode	MODBUS_EXCEPTION	NO_EXCEPTION
◈ byDiag_3_reserved	BYTE	0
◈ wCommCounter	WORD	116
◈ wCommErrorCounter	WORD	49
⊞ 🌐 MODBUS_Server_1_REQDG_0004	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	

Figure 40: ReqDiagnostics GVL in Online Mode

4.14.9.   Prepare_Start Function

In this POU, the *PrepareStart* system event function is defined. It belongs to the communication task and is called before starting the application. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user starts the application, the count is incremented.

4.14.10.   Prepare_Stop Function

In this POU, the *PrepareStop* system event function is defined. It belongs to the communication task and is called before stopping the application. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user stops the application, the count is incremented.

4.14.11.   Start_Done Function

In this POU, the *StartDone* system event function is defined. It belongs to the communication task and is called when the application is successfully started. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user successfully launches the application, the count is incremented.

4.14.12.   Stop_Done Function

In this POU, the *StopDone* system event function is defined. It belongs to the communication task and is called when the application is successfully stopped. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user successfully stops the application, the count is incremented.

altus

5. Configuration

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

5.1. Device

5.1.1. User Management and Access Rights

It provides functions to define users accounts and to configure the access rights to the project and to the CPU. Using 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.
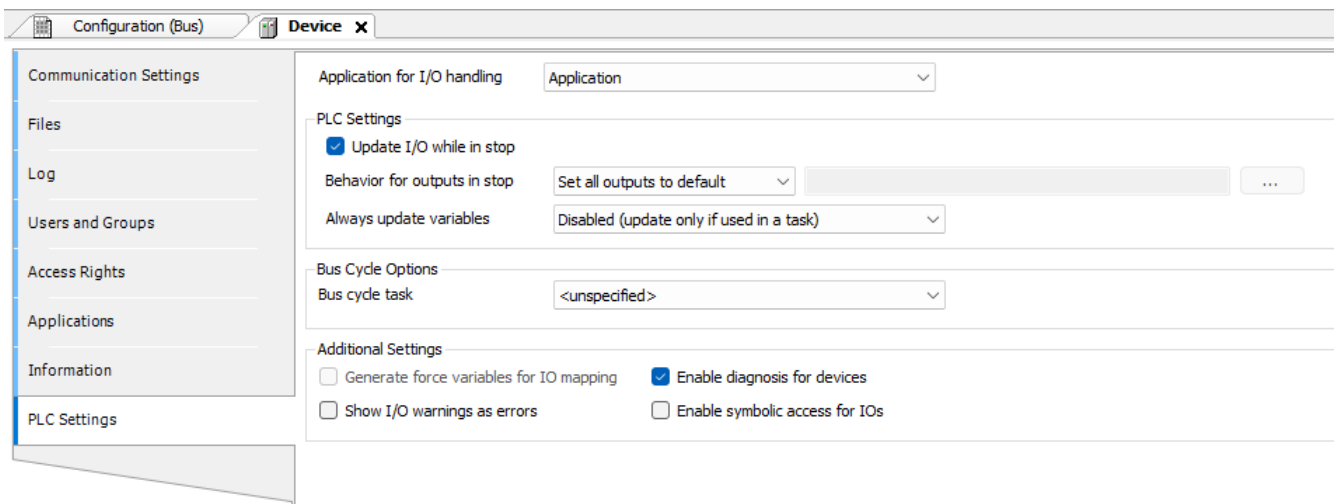
Figure 41: PLC Settings

Parameter	Description
Application for I/O handling	Application that is responsible for the I/O handling.
Refresh I/Os 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 of the outputs at 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.
Force variables for the I/O 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.
Activate 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.
Display I/O warnings as errors	Warnings concerning the I/O configuration are displayed as errors.
Enable symbolic access for I/Os	**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 40: PLC Settings

ATTENTION

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

---

altus

5.2. CPU Configuration

5.2.1. General Parameters

The parameters related below are part of the CPU configuration included in the application. Each item must be properly verified for the correct project execution.

Besides these parameters, it is possible to change the name of each module inserted in the application by clicking the right button on the module. In the *Properties* item from the *Common* sheet, change the name, what is limited to 24 characters.

Settings	Description	Standard	Options
Diagnostics Area (%Q)			
%Q Start Address	Starting address of the UCP diagnostics (%Q)	Automatically allocated on project creation.	0 a 32210
Size	Size of diagnostics area in bytes	558	It is not possible to change the size of the CPU diagnostics area
Retaining Area (%Q)			
%Q Start Address	Starting address of the retentive data memory (%Q)	4096	0 a 32767
Size	Retain data memory size in bytes	32768	0 a 32768
Persistent Area (%Q)			
%Q Start Address	Persistent data memory start address (%Q)	12288	0 a 32767
Size	Persistent data memory size in bytes	32768	0 a 32768
CPU Parameters			
Start User Application after Reset by Watchdog	When enabled, starts the user application after resetting the hardware watchdog or restarting Runtime, but maintaining the diagnostic indication via WD LED and via variables.	Disabled	Enabled Disabled
Hot Swap Mode	Module hot swap mode	Enabled, no match consistency. (may vary according to CPU model)	- Denabled, only for declared modules - Disabled (with match consistency) - Disabled, no match consistency - Enabled, with match consistency only for declared modules - Enabled, with match consistency - Enabled, no match consistency

Settings	Description	Standard	Options
	Project Parameters		
Enable I/O update per task	Setting to update inputs and outputs in the tasks in which they are used.	unmarked	- Checked: Inputs and outputs are updated by the tasks in which they are used. - Unchecked: Inputs and outputs are only updated by MainTask
Enable retain and persistent variables in Function Blocks	Setting that allows the use of retentive and persistent variables in Function Blocks	unmarked	- Marked: allows the use of retentive and persistent variables in Function Blocks. - Unchecked: Exception error may occur at startup.

Table 41: CPU settings

Notes:

Generate error on tasks watchdog consistency: This parameter was discontinued as of MasterTool IEC XE version 1.32.

Enable I/O update per task: This parameter was added as of MasterTool IEC XE version 2.01.

> **ATTENTION**
>
> When the initial address or the retentive or persistent data memory size are changed in the user application, the memory is totally reallocated, what makes the retentive and persistent variable area be clean. So the user has to be careful so as not to lose the saved data in the memory.

> **ATTENTION**
>
> In situations where the symbolic persistent memory area is modified, a message will be displayed by MasterTool IEC XE programmer, to choose the behavior for this area after charging the modified program. The choice of this behavior does not affect the persistent area of direct representation, which is always clean.

> **ATTENTION**
>
> The option *Enable I/O update per task* is not supported for fieldbus masters such as NX5001 module. This feature is applicable only for input and output modules present on the controller local bus (main rack and expansion racks).

> **ATTENTION**
>
> Even when an I/O point is used in other tasks, with the *Enable I/O update per task* marked, it will continue to be updated in the MainTask as well; except when all the points of the module are used in some other task, in this case they will not be updated on MainTask anymore.

5.2.1.1. Hot Swap

Nexto Series CPUs have the possibility of I/O modules change in the bus with no need for system turn off and without information loss. This feature is known as hot swap.

> **CAUTION**
>
> Nexto Series CPUs do not guarantee the persistent and retentive variables retentivity in case the power supply or even the CPU is removed from the energized backplane rack.

altus

On the hot swap, the related system behavior modifies itself following the configuration table defined by the user which represents the options below:

- Disable, for declared modules only
- Disabled (with startup consistency)
- Disabled, without startup consistency
- Enabled, with startup consistency for declared modules only
- Enabled, with startup consistency
- Enabled, without startup consistency

Therefore, the user can choose the behavior that the system must assume in abnormal bus situations and when the CPU is in *Run* Mode. The table below presents the possible abnormal bus situations.

Situation	Possible causes
Incompatible configuration	- Some module connected to the bus is different from the model that is declared in configuration.
Absent module	- The module was removed from the bus. - Some malfunctioning module is not responding to CPU - Some bus position is malfunctioning.

Table 42: Bus Abnormal Situations

For further information regarding the diagnostics correspondent to the above described situations, see *Diagnostics via Variables*.

If a module is present in a specific position in which should not exist according to the configuration modules, this module is considered as non-declared. The options of hot swap *Disabled, for Declared Modules Only* and *Enabled, with Startup Consistency for Declared Modules Only* do not take into consideration the modules that are in this condition.

5.2.1.1.1. Hot Swap Disabled, for Declared Modules Only

In this configuration, the CPU is immediately in *Stop* Mode when an abnormal bus situation (as described on Table 42) happens. The LED DG starts to blink 4x (according to Table 43). In this case, in order to make the CPU to return to the normal state *Run*, in addition to undo what caused the abnormal situation, it is necessary to execute a *Reset Warm* or a *Reset Cold*. If a *Reset Origin* is carried out, it will be necessary to perform the download so that the CPU can return to the normal state (*Run*). The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

The CPU will remain in normal *Run* even if find a module not declared on the bus.

5.2.1.1.2. Hot Swap Disabled

This setting does not allow any abnormal situation in the bus (as shown in Table 42) modules including undeclared and present on the bus. The CPU enters in *Stop* mode, and the DG LED begins to blink 4x (as in Table 43). For these cases, to turn the CPU back to normal *Run*, in addition to undo what caused the abnormal situation it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is done, you need to download the project so that the CPU can return to normal *Run*. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

5.2.1.1.3. Hot Swap Disabled, without Startup Consistency

Allows the system to start up even when some module is in an abnormal bus situation (as shown in Table 42). Abnormal situations are reported via diagnosis.

Any modification to the bus will cause the CPU to enter *Stop* Mode, and the DG LED will start blinking 2x (as in Table 43). In order for the CPU to return to the normal *Run* state in these cases, it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is performed, it will be necessary to download the CPU so that the CPU can return to the normal *Run* state. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

5.2.1.1.4. Hot Swap Enabled, with Startup Consistency for Declared Modules Only

"Startup" is the interval between the CPU energization (or reset command or application download) until the first time the CPU gets in *Run* Mode after been switched on. This configuration verifies if any abnormal bus situation has occurred (as described on Table 42) during the start. In affirmative case, the CPU gets in *Stop* Mode and the LED DG starts to blink 4x (according to Table 43). Afterwards, in order to set the CPU in *Run* mode, further to fix what caused the abnormal situation, it is necessary to execute a *Reset Warm* or *Reset Cold* command, which can be done by the MasterTool IEC XE (Online menu). If a *Reset Origin* is carried out, it will be necessary to perform the download so that the CPU can return to the normal state (*Run*).

After the start, if any module present any situation described in the previous table, the system will continue to work normally and will signalize the problem via diagnostics.

If there is no other abnormality for the declared modules, the CPU will go to the normal state (*Run*) even if a non-declared module is present on the bus.

> **ATTENTION**
>
> In this configuration when a power fault occurs (even temporally), *Reset Warm* Command,*Reset Cold* Command or a new application *Download* has been executed, and if any module is in an abnormal bus situation, the CPU will get into *Stop* Mode and the LED DG will start to blink 4x (according to Table 43). This is considered a startup situation.
> This is the most advised option because guarantee the system integrity on its initialization and allows the modules change with a working system.

5.2.1.1.5. Hot Swap Enabled with Startup Consistency

This setting checks whether there has been any abnormal situation in the bus (as shown in Table 42) during the startup, even if there is no declared modules and present on the bus; if so, the CPU goes into *Stop* mode and the LED DG starts to blink 4x (as shown in Table 43). For these cases, to turn the CPU back to normal *Run*, in addition to undo what caused the abnormal situation it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is done, you need to download the project so that the CPU can return to normal *Run*. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

5.2.1.1.6. Hot Swap Enabled without Startup Consistency

Allows the system to start working even if a module is in an abnormal bus situation (as described on Table 42). The abnormal situations are reported via diagnostics during and after the startup.

> **ATTENTION**
>
> This option is advised for the system implementation phase as it allows the loading of new applications and the power off without the presence of all configured modules.

5.2.1.1.7. How to do the Hot Swap

> **CAUTION**
>
> Before performing the Hot Swap it is important to discharge any possible static energy accumulated in the body. To do 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.

> **ATTENTION**
>
> It is recommended the hot swapping diagnostics monitoring in the application control developed by the user in order to guarantee the value returned by the module is validated before being used.

The hot swap proceeding is described below:

■ Unlock the module from the backplane rack, using the safety lock.

- Take off the module, pulling firmly.
- Insert the new module in the backplane rack.
- Certify the safety lock is completely connected. If necessary, push the module harder towards to the backplane rack.

In case of output modules is convenient the points to be disconnected when in the changing process, in order to reduce the generation of arcs in module connector. This must be done by switching off the power supply or by forcing the output points using the software tools. If the load is small, there is no need for disconnecting.

It is important to note that in the cases the CPU gets in *Stop* Mode and the DG LED starts to blink 4x (according to Table 43, due to any abnormal bus situation (as described on Table 42, the output modules have its points operation according to the module configuration when CPU toggles from *Run* Mode to *Stop* Mode. In case of application startup, when the CPU enters *Stop* Mode without having passed to the *Run* Mode, the output modules put their points in failure secure mode, in other words, turn it off (0 Vdc).

Regarding the input modules, if one module is removed from energized backplane rack, the logic point's state will remain in the last value. In the case a connector is removed, the logic point's state will be put in a safe state, it means zero or high impedance.

> **ATTENTION**
>
> Always proceed to the substitution of one module at a time for the CPU to update the modules state.

Below, Table 43 presents the bus conditions and the Nexto CPU DG LED operation state. For further information regarding the diagnostics LEDs states, see Diagnostics via LED section.

Condition	Enabled, with Startup Consistency	Enabled, with Startup Consistency for Declared Modules Only	Enabled, without Startup Consistency	Disabled	Disabled, for declared modules only	Disabled, without Startup Consistency
Non declared module	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Stop
Non declared module (startup condition)	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run
Absent module	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop
Absent module (startup condition)	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run
Incompatible configuration	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop

altus

Condition	Enabled, with Startup Consistency	Enabled, with Startup Consistency for Declared Modules Only	Enabled, without Startup Consistency	Disabled	Disabled, for declared modules only	Disabled, without Startup Consistency
Incompatible configuration(startup condition)	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run or LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run
Duplicated slot address	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop
Non-operational module	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop

Table 43: Hot Swap and Conditions Relations

Note:

Enabled, without startup consistency: When this hot-swap mode is configured, in normal situations when there's an incompatible module on the system's startup, the application will go from Stop to Run. However, if that module is configured as a NX5000 or a NX5001 and there's a different module in that position, the application will stay in Stop.

5.2.1.2. Retain and Persistent Memory Areas

The Nexto CPU allows the use of symbolic variables and output variables of direct representation as retentive or persistent variables.

The output variables of direct representation which will be retentive or persistent must be declared in the CPU *General Parameters*, as described at CPU Configuration. Symbolic names also can be attributed to these output variables of direct representation using the AT directive, plus using the key word RETAIN or PERSISTENT on its declaration. For example, being %QB4096 and %QB20480 within the retentive and persistent memory, respectively:

```
PROGRAM UserPrg
VAR RETAIN
byRetentiveVariable_01 AT %QB4096 : BYTE;
END_VAR
VAR PERSISTENT
byPersistentVariable_01 AT %QB20480 : BYTE;
END_VAR
```

In case the symbolic variables declared with the AT directive are not inside the respective retentive and/or persistent memory, errors during the code generation in MasterTool can be presented, informing that there are non-retentive or non-persistent variables defined in the retentive or persistent memory spaces.

Regarding the symbolic variables which will be retentive or persistent, only the retentive variables may be local or global, as the persistent symbolic variables shall always be global. For the declaration of retentive symbolic variables, it must be used the key word *RETAIN*. For example, for local variables:

altus

```
PROGRAM UserPrg
VAR RETAIN
  wLocalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

Or, for global variables, declared within a list of global variables:

```
VAR_GLOBAL RETAIN
  wGlobalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

On the other hand, the persistent symbolic variables shall be declared in a Persistent Variables object, being added to the application. These variables will be global and will be declared in the following way within the object:

```
VAR_GLOBAL PERSISTENT RETAIN
  wGlobalSymbolicPersistentVariable_01 : WORD;
END_VAR
```

As of versions 1.5.1.1 the Nexto Series CPUs allow flexibility on the usage of retentive and persistent memories. This means that the user will be able to choose the size that will be used for each type of memory, as long as the retentive and persistent memory sum don't exceed the total limit available in each CPU model. The total of retentive and persistent memory available is described in the Table 7 in Memory.

If the retentive symbolic, persistent symbolic, retentive %Q and persistent %Q memory sum exceed the total available, MasterTool will show an error during the code generation.

```
VAR_GLOBAL PERSISTENT RETAIN
  wGlobalSymbolicPersistentVariable_01 : WORD;
END_VAR

VAR_GLOBAL RETAIN
  wGlobalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

ATTENTION

To use the retentive and persistent memory flexibly, it's necessary to use MasterTool IEC XE 2.03 or higher.

5.2.1.3. Project Parameters

The CPU project parameters are related to the configuration for input/output refreshing at the task that they are used of the project tasks and the options for reading and writing on the memory card.

altus

Configuration	Description	Default	Options
Enable I/O update per task	Updates the input and output in the tasks where they are used	Unmarked	- Marked - Unmarked
Enable retain and persistent variables in Function Blocks	Setting to allow the use of retentive and persistent variables in function blocks	Unmarked	- Marked - Unmarked
	Memory Card		
Copy Project from CPU to Memory Card	Copy the project from the CPU internal memory to the memory card	Disabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
Password to Copy Project from CPU to Memory Card	Password for coping the project from the CPU internal memory to memory card	-	6 digits password (0 to 999999)
Copy Project from Memory Card to CPU	Copy the project from the memory card to the CPU internal memory	Disabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
Password to Copy Project from Memory Card to CPU	Password for coping the project from the memory card to the CPU internal memory	-	6 digits password (0 to 999999)

Table 44: CPU Project Parameters

> **ATTENTION**
>
> After setting the project copy possibilities and having created the boot application, it must be found the "*Application.crc*" file in order the configurations concerning the memory card have effect. The search can be done at *Select the Application.crc* through the *Find File...* key, as can be seen on Figure 105.

5.2.2. External Event Configuration

The external event is a feature available in the CPU which enables a digital input, configured by the user, when activated, triggers the execution of a specific task with user-defined code. Thus, it is possible that through this input, when triggered, interrupt the execution of the main application and run the set application in the task *ExternInterruptTask00*, which has higher priority than other application tasks. Because the inputs and outputs are updated in the context of the MainTask task, the External Event task does not have the input and output data updated at the time of its call. If necessary, use the I/O update functions.

It is also important to note that, to avoid the generation of several events in a very short space of time, that was limited the treatment of this type of event in every 10 ms, i.e., if two or more events occurs during 10 ms after the first event, the second and subsequent events are discarded. This limitation is imposed to prevent an external event that is generated in an uncontrolled way, do not block the CPU, since the task has a higher priority over the others.

To configure an external event is necessary to insert a digital input module and perform the configurations described below, in the CPU, through the MT8500 programming tool software.
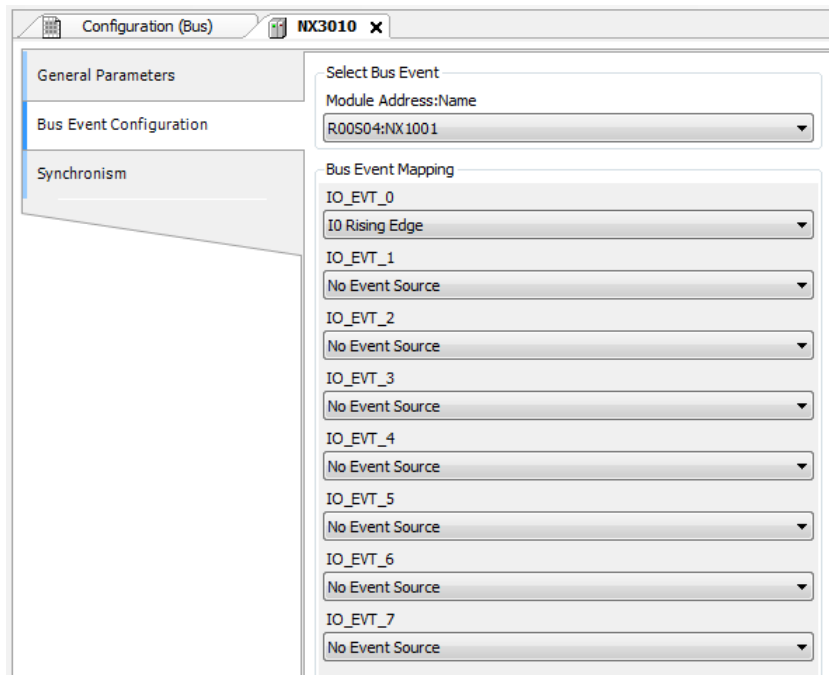
Figure 42: Configuration Screen for External Event in CPU

In the configuration external event tab, within the CPU settings, it is necessary to select which module will be the interruption source, in the field *Module Address: Name*. Then it must be selected which input of this module will be responsible for the event generation (*IO_EVT_0*). In this selection the options described in the figure below can be chosen.
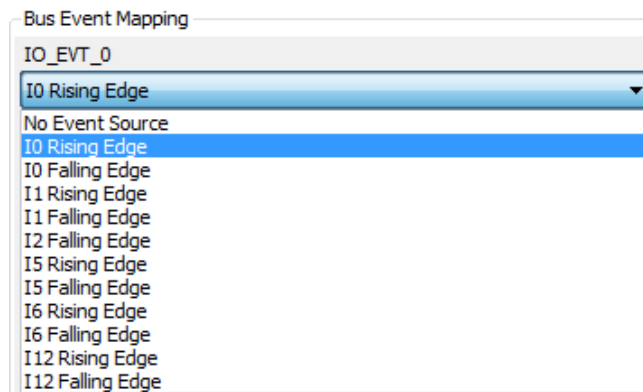
Figure 43: NX1001 Module External Event Source Options

In addition to configuring the CPU it is required to configure the task responsible for executing user-defined actions. In this case the user must use a project profile that supports external events. For further information see the section Project Profiles. In the configuration screen of the *ExternInterruptTask00* (figure below), it is necessary to select the event source in the corresponding field. In this case, *IO_EVT_0* should be selected since the other origin sources (*IO_EVT_1* to *IO_EVT_7*) are not available. In the sequence, the field *POU* should be checked if the right POU is selected, because it will be used by the user to define the actions to be performed when an external event occurs.

altus
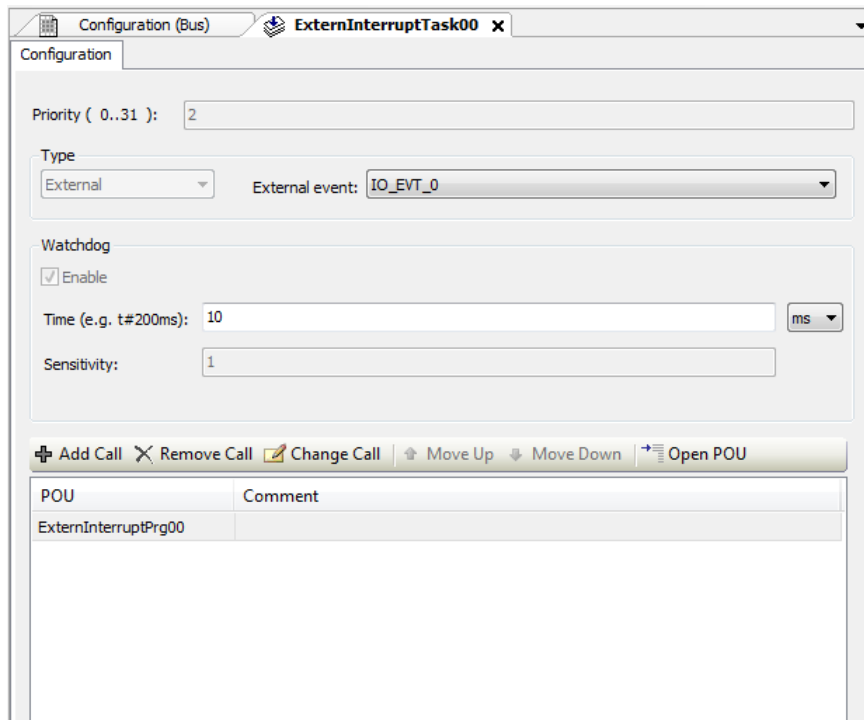
Figure 44: ExternInterruptTask00 Configuration Screen

5.2.3. Time Synchronization

For the time synchronization, Nexto Series CPUs use the SNTP (*Simple Network Time 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, accessed through the CPU, in the device tree:
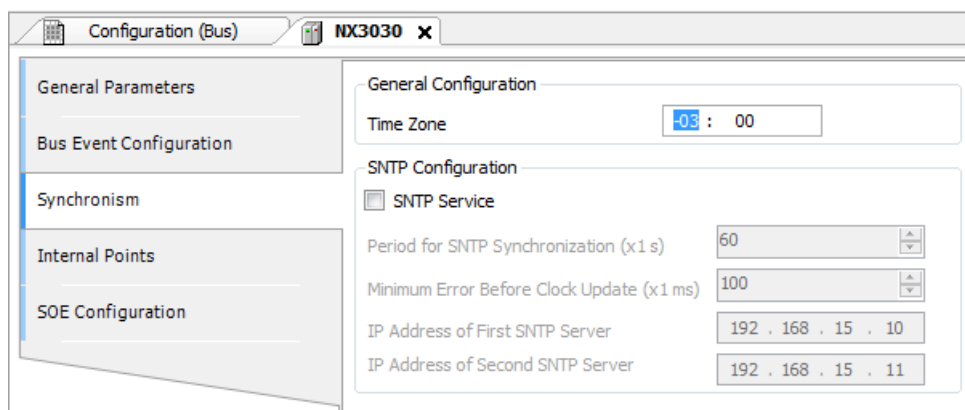
Figure 45: SNTP Configuration

altus

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 Enabled
Period for SNTP Synchronization (x1 s)	Time interval of the synchronization requests (seconds).	60	1 to 255
Minimum Error Before Clock Update (x1 ms)	Offset value acceptable between the server and client (milliseconds).	100	1 to 65519
IP Address of First SNTP Server	IP Address of the primary SNTP server.	192.168.15.10	1.0.0.1 to 223.255.255.254
IP Address of Second Second SNTP Server	IP Address of the secondary SNTP server.	192.168.15.11	1.0.0.1 to 223.255.255.254

Table 45: 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 (DNP3, IEC 60870-5-104 and MasterTool Device Log), while the local time is used by an others CPU's features (GetDateAndTime function, OTD date and time info).

It is allowed to enable more than one sync source on the project, however the device doesn't supports the synchronism from more than one sync source during operation. Therefore there are implicitly defined a priority mechanism. The synchronism through SNTP 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.

5.2.3.1. SNTP

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

The CPU sends the cyclic synchronization requests according to the time set in the *Period for SNTP Synchronization* field. In the first synchronization attempt, just after the service start up, the request is for the first server set in the first server IP address. In case it does not respond, the requests are directed to the second server set in the second server IP address providing a redundancy of SNTP servers. In case the second server does not respond either, the same process of synchronization attempt is performed again but only after the Period of Synchronization having been passed. In other words, at every synchronization period the CPU tries to connect once in each server, it tries the second server in case the first one does not respond. The waiting time for a response from the SNTP server is defined by default in 5 s and it cannot be modified.

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

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

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

altus

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

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

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

5.2.3.2. Daylight Saving Time (DST)

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

For further information, see the section RTC Clock.

5.3. Serial Interfaces Configuration

5.3.1. COM 1

The COM 1 communication interface, is composed by a DB9 female connector for RS-232C standard. It allows the point to point communication (or in network by using a converter) in MODBUS RTU slave or MODBUS RTU master open protocols.

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

When using the MODBUS master/slave protocol, some of these parameters (such as Serial Mode, Data Bits, RX Threshold and Serial Events) are automatically adjusted by MasterTool for the correct operation of this protocol.

Configuration	Description	Default	Options
Serial Type	Serial channel type configuration.	RS-232C	RS-232C
Baud Rate	Serial communication port speed configuration.	115200	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps
Parity	Serial port parity configuration.	None	Odd Even Space Mark None
Data Bits	Sets the serial communication character bits quantity.	8	5, 6, 7 and 8
Stop Bits	Sets the serial port stop bits.	1	1, 1.5 and 2
Serial Mode	Sets the serial port operation mode configuration.	Normal Mode	- Extended Mode: Extended operation mode which delivers information regarding the received data frame. - Normal Mode: Serial communication normal operation mode.

Table 46: RS-232 Standard Serial Configuration

altus

Notes:

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

- One byte for the received data (RX_CHAR : BYTE): Store the five, six, seven or eight bits from the data received, depending on the serial communication configuration.
- One byte for the signal errors (RX_ERROR : BYTE): It has the format described below:

  - Bit 0: 0 - the character in bits 0 to 7 is valid. 1 - the character in bits 0 to 7 is not valid (or it cannot be valid), due to problems indicated in bits 10 to 15.

  - Bit 1: Not used.

  - Bit 2: Not used.

  - Bit 3: UART interruption error. The serial input remained in logic 0 (space) for a time greater than a character (start bit + data bits + parity bit + stop bits).

  - Bit 4: UART frame error. The logic 0 (space) was read when the first stop bit was expected and it should be logic 1 (mark).

  - Bit 5: UART parity error. The parity bit read is not correct according to the calculated one.

  - Bit 6: UART overrun error. Data was lost during the FIFO UART reading. New characters were received before the later ones were removed. This error will only be indicated in the first character read after the overrun error indication. This means some old data were lost.

  - Bit 7: RX line overrun error. This character was written when the RX line was completed, overwriting the unread characters.

- Two bytes for the timestamp signal (RX_TIMESTAMP : WORD): Indicates the silence time, within the 0 to 65535 interval, using 10 us as base. It saturates in 655.35 ms if the silence time is higher than 65535 units. The RX_TIMESTAMP of a character measures the time from a reference which can be any of the three options below:

  - On most of the cases, the end of the later character.

  - Serial port configuration.

  - The end of serial communication using the SERIAL_TX FB, in other words, when the last character is sent on line.

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

Data Bits: The serial interfaces *Data Bits* configuration limits the *Stop Bits* and Communication *Parity* fields. Therefore, the stop bits number and the parity method will vary according to the data bits number.

Data Bits	Stop Bits	Parity
5	1, 1.5	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
6	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
7	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
8	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO

Table 47: Specific Configurations

5.3.1.1. Advanced Configurations

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

Configuration	Description	Default	Options
Advanced Port Parameters			
Handshake	Executes the request control for a command transmission through RS-232C interface.	RTS Off	- RTS: Enabled at the beginning of transmission and restarted, as fast as possible after the end of it. E.g. The RS-232/RS-485 external converter control. - RTS Off: Always disabled. - RTS On: Always enabled. - RTS/CTS: In case the CTS is disabled, the RTS is enabled. Therefore the CTS enabling must be waited until the transmission can start again and the RTS restarted, as fast as possible, at the end of transmission. E.g. the radio modems control using the same modem signal. - Manual RTS: the user is responsible for all control signals.
UART RX Threshold	Bytes quantity which must be received to generate a new UART interruption. 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
Serial Events			
RX on TX	When true, all received bytes during transmission will be discharged instead of going to the RX line. Used to disable the full-duplex operation of the RS-232C interface.	Disabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
RX DCD Event	When true, generates an external event due to DCD signal change.	Enabled	- Enabled: Configuration enabled - Disabled: Configuration disabled

Configuration	Description	Default	Options
RX CTS Event	When true, generates an external event due to CTS signal change.	Enabled	- Enabled: Configuration enabled - Disabled: Configuration disabled

Table 48: RS-232 Standard Serial Advanced Configurations

Notes:

RX on TX: This advanced parameter is valid for RS-232C settings and RS-422.

RX DCD Event: External events such as the DCD signal COM 1 of the CPUs NX3010, NX3020, NX3030, may be associated only to tasks of custom project profile, for further information, please see the MasterTool IEC XE User Manual – MU299609.

RX CTS Event: External events such as the CTS signal COM 1 of the CPUs NX3010, NX3020, NX3030, may be associated only to tasks of custom project profile, for further information, please see the MasterTool IEC XE User Manual – MU299609.

5.4. Ethernet Interfaces Configuration

Nexto CPUs can provide more local Ethernet interfaces. The NX3010 CPU only has NET 1. In addition of the local Ethernet interfaces, the Nexto Series also provides remote Ethernet interfaces through the inclusion of the module NX5000. NX5000 modules have only the NET 1 interface.

5.4.1. Internal Ethernet Interface

5.4.1.1. NET 1

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

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

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
Subnetwork 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 49: NET 1 Configuration

5.4.2. Reserved TCP/UDP Ports

The following TCP/UDP ports of the Ethernet interfaces, both local 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.

altus

Service	TCP	UDP
System Web Page	80	-
SNTP	-	123
SNMP	-	161
MODBUS TCP	502*	-
MasterTool MT8500	1217*	1740:1743
SQL Server	1433	-
MQTT	1883* / 8883*	-
EtherNet/IP	44818	2222
IEC 60870-5-104	2404*	-
OPC UA	4840	-
WEBVISU	8080	-
CODESYS ARTI	11740	-
PROFINET	-	34964
Portainer Docker	9000	-

Table 50: Reserved TCP/UDP ports

* Default port, but user changeable.

5.5. Protocols Configuration

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:

	NX3010
Mapped Points	20480
Mappings (Per Instance / Total)	5120 / 20480
Requests	512
NETs – Client or Server Instances (Per NET / Total)	4 / 4
COM (n) – Master or Slave Instances	1
Control Centers	-

Table 51: Protocols Limits per CPU

Notes:

Mapped Points: It refers to the maximum number of mapped points supported by the CPU. Each mapping supports one or more mapped points, depending on the size of the data when used with variables of type ARRAY.

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 the requests of the communications protocols, declared on devices, may not exceed the maximum number of requests supported by the CPU.

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

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

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 IEC 60870-5-104 Server (does not include master or clients from MODBUS RTU Slave, MODBUS Server and DNP3 Server protocols).

The limitations of the MODBUS protocol for Direct Representation and symbolic mapping for the CPUs can be seen in Tables 52 and 53, respectively.

Limitations	MODBUS RTU Master	MODBUS RTU Slave	MODBUS Ethernet Client	MODBUS Ethernet Server
Mappings per instance	128	32	128	32
Devices per instance	64	1[1]	64	64[2]
Mappings per device	32	32	32	32
Simultaneous requests per instance	-	-	128	64
Simultaneous requests per device	-	-	8	64

Table 52: MODBUS Protocol Limitations for Direct Representation

Notes:

Devices per instance:

- Master or Client Protocols: number of slaves or server devices supported by each Master or Slave 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 number 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.

Mappings per device: The maximum number of mappings per device, despite being listed above, is also limited by the protocol maximum number of mappings. Also to be considered the maximum CPU mappings as in Table 51.

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 to each MODBUS Server device, or may be received simultaneously by each MODBUS client device. MODBUS RTU devices, Master or Slave, do not support simultaneous requests.

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	-
Simultaneous requests per instance	-	-	128	64
Simultaneous requests per device	-	-	8	64

Table 53: 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.

altus

- MODBUS RTU Server Protocol: the limit [2] informed relates to the Ethernet interfaces, which limit the amount of connections that can be established with other devices through a single Ethernet interface. It is not necessary, nor is it possible to declare or configure Clients devices in the instance of the MODBUS Server protocol. All Clients devices will have access to all the mappings made directly in the instance of the MODBUS Server protocol.

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

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

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

5.5.1. Protocol Behavior x CPU State

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

Protocol	Type	CPU operational state					
		STOP			RUN		
		After download, before application starts	After the application goes to STOP (PAUSE)	After an exception	Non redundant or Active	Redundant in Stand-by	After a breakpoint in MainPrg
MODBUS Symbol	Slave/Server	✓	✓	✓	✓	✓	✓
	Master/Client	✗	✗	✗	✓	✓	✓
MODBUS	Slave/Server	✗	✗	✗	✓	✓	✗
	Master/Client	✗	✗	✗	✓	✓	✓
SOE (DNP3)	Outstation	✓	✓	✓	✓	✗	✓
IEC 60870-5-104	Server	✗	✗	✗	✓	✗	✓
EtherCAT	Master	✓	✓	✗	✓	NA	✓
OPC DA	Server	✓	✓	✓	✓	✗	✓
OPC UA	Server	✓	✓	✓	✓	✓	✓
SNTP	Client	✓	✓	✓	✓	✓	✓
HTTP	Server	✓	✓	✓	✓	✓	✓
SNMP	Agent	✓	✓	✓	✓	✓	✓
EtherNet/IP	Scanner	✓	✓	✗	✓	NA	✗
	Adapter	✗	✓	✗	✓	NA	✗

Table 54: Protocol Behavior x CPU State

Notes:

Symbol ✓: Protocol remains active and operating normally.

Symbol ✗: Protocol is disabled.

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

5.5.2. 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 COM 1 or COM 2 (or both, in case of two communication networks). To execute this procedure, see Inserting a Protocol Instance section.
- Configure the serial interface, choosing the transmission speed, the RTS/CTS signals behavior, the parity, the channel stop bits, among others configurations by a double click on the COM 1 or COM 2 serial channel. See Serial Interfaces Configuration section.

5.5.2.1. MODBUS Master Protocol Configuration by Symbolic Mapping

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

- Configure the general parameters of the MODBUS Master protocol, like: transmission delay times and minimum interframe as in Figure 46.
- 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 47.
- Add and configure the MODBUS mappings on Mappings tab as Figure 48, 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 49, 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.

5.5.2.1.1. MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as:
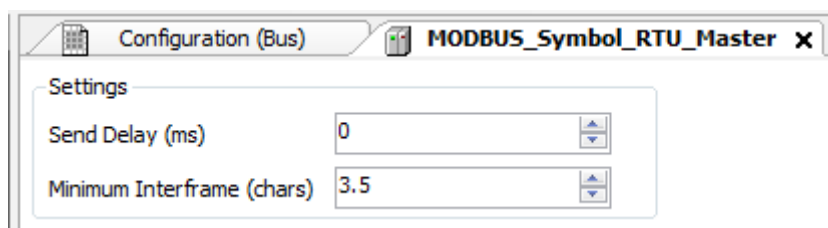
Figure 46: MODBUS RTU Master Configuration Screen

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 55: 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 (where "n" is the configured value in the *%Q Start Address of Diagnostics Area* field):

Direct Repre-sentation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_MASTER_1.*	Size	Description
Diagnostics Bits:			
%QX(n).0	tDiag. bRunning	BIT	The master is running.
%QX(n).1	tDiag. bNotRunning	BIT	The master is not running (see bit: bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the master was interrupted by the user through command bits.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnosis.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnosis.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnosis.
%QX(n).6	tDiag. bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present.
%QX(n).7	tDiag. bDiag_7_reserved	BIT	Reserved
Error Codes:			
%QB(n+1)	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

altus

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_MASTER_1.*	Size	Description
			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:			
%QX(n+2).0	tCommand. bStop	BIT	Stop master.
%QX(n+2).1	tCommand. bRestart	BIT	Restart master.
%QX(n+2).2	tCommand. bResetCounter	BIT	Restart diagnostics statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved
Communication Statistics:			
%QW(n+4)	tStat. wTXRequests	WORD	Counter of request transmitted by the master (0 to 65535).
%QW(n+6)	tStat. wRXNormalResponses	WORD	Counter of normal responses received by the master (0 to 65535).
%QW(n+8)	tStat. wRXExceptionResponses	WORD	Counter of responses with exception codes received by the master (0 to 65535).
%QW(n+10)	tStat. wRXIllegalResponses	WORD	Counter of illegal responses received by master – invalid syntax, not enough received bytes, invalid CRC – (0 to 65535).
%QW(n+12)	tStat. wRXOverrunErrors	WORD	Counter of overrun errors during reception - UART FIFO or RX line – (0 to 65535).
%QW(n+14)	tStat. wRXIncompleteFrames	WORD	Counter of answers with construction errors, parity or failure during reception (0 to 65535).

altus

Direct Repre-sentation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_MASTER_1.*	Size	Description
%QW(n+16)	tStat. wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using RT-S/CTS handshake, during transmission (0 to 65535).
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved

Table 56: MODBUS RTU Master Diagnostics

Note:

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

5.5.2.1.2. Devices Configuration – Symbolic Mapping configuration

The devices configuration, shown on figure below, follows the following parameters:
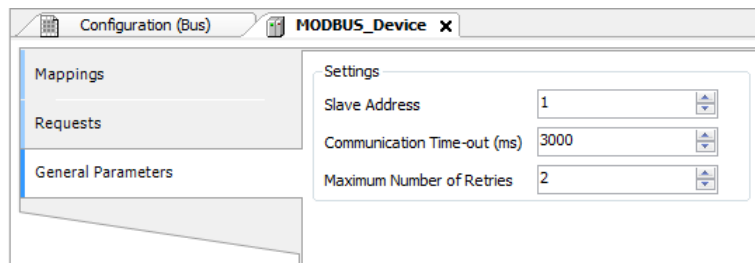
Figure 47: Device General Parameters Settings

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 Re-tries	Defines the numbers of re-tries before reporting a com-munication error	2	0 to 9

Table 57: 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.

5.5.2.1.3. Mappings Configuration – Symbolic Mapping Settings

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:
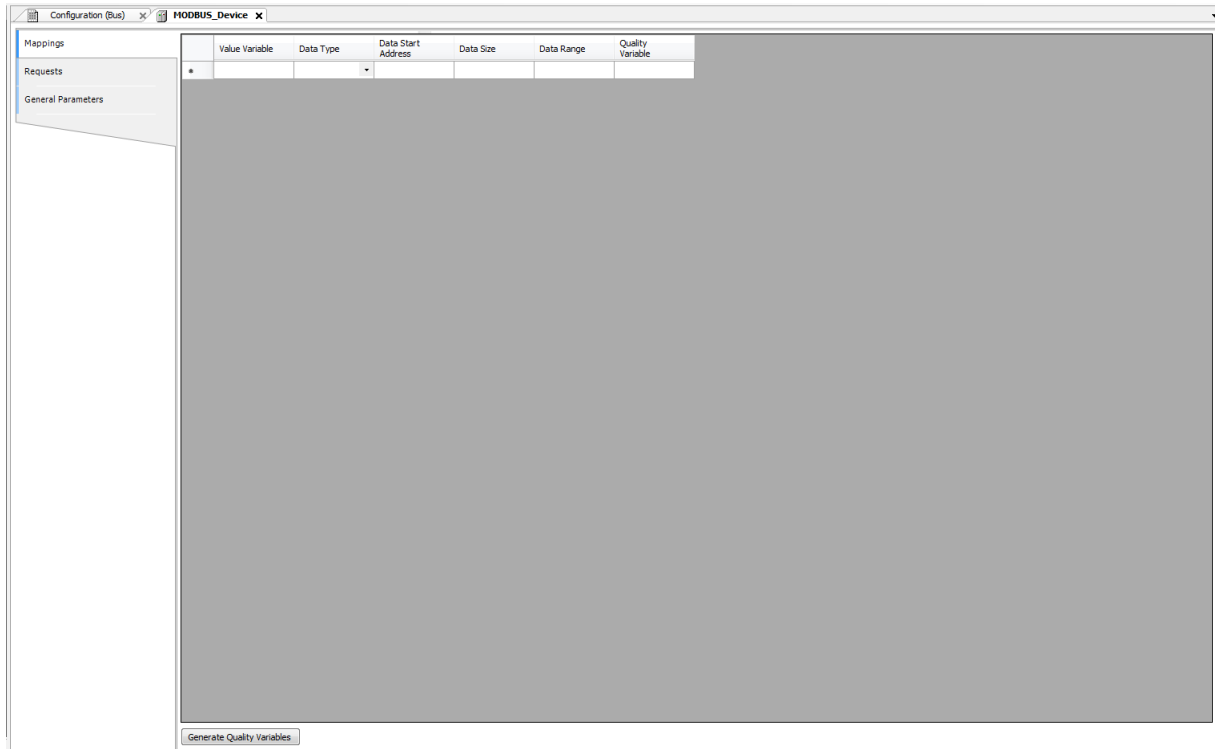
Figure 48: MODBUS Data Mappings Screen

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 58: 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 59: 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.

5.5.2.1.4.   Requests Configuration – Symbolic Mapping Settings

The configuration of the MODBUS requests, viewed in figure below, follow the parameters described in table below:
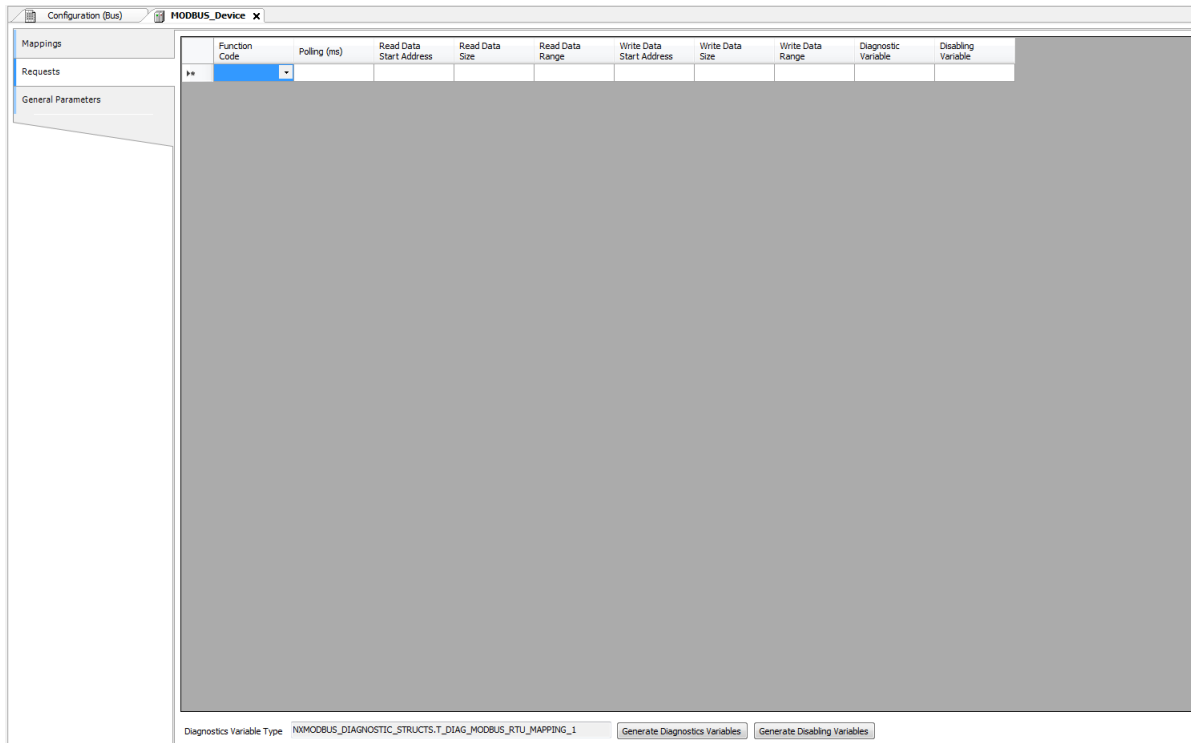
Figure 49: Data Requests Screen MODBUS Master

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 60: 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:

altus

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 61: 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 62 ("n" is the value configured in the *%Q Start Address of Diagnostics Area* field).

Direct Repre- sentation Variable	Diagnostic variable of type T_DIAG_MODBUS _RTU_MAPPING_1.*	Size	Description
Communication status bits:			
%QX(n).0	byStatus. bCommIdle	BIT	Communication idle (waiting to be executed).
%QX(n).1	byStatus. bCommExecuting	BIT	Active communication.
%QX(n).2	byStatus. bCommPostponed	BIT	Communication delayed, 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.
%QX(n).3	byStatus. bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
%QX(n).4	byStatus. bCommOk	BIT	Communication terminated previously was held successfully.
%QX(n).5	byStatus. bCommError	BIT	Communication terminated previously had an error. Check error code.
%QX(n).6	byStatus. bCommAborted	BIT	Not used in MODBUS RTU Master.
%QX(n).7	byStatus. bDiag_7_reserved	BIT	Reserved
Last error code (enabled when bCommError = true):			
%QB(n+1)	eLastErrorCode	MASTER_ERROR _CODE (BYTE)	Informs the possible cause of the last error in the MODBUS mapping. Consult Table 85 for further details.
Last exception code received by master:			
%QB(n+2)	eLastExceptionCode	MODBUS _EXCEP- TION (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:			
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
%QW(n+4)	wCommCounter	WORD	Finished communications counter (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.
%QW(n+6)	wCommErrorCounter	WORD	Finished communications counter (with errors). When the value 65535 is reached, the counter returns to zero.

Table 62: MODBUS Relations Diagnostics

altus

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 63: 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.

5.5.2.2. MODBUS Master Protocol Configuration for Direct Representation (%Q)

To configure this protocol using direct representation (%Q), the following steps must be performed:

- Configure the general parameters of the MODBUS protocol, such as: communication times and direct representation variables (%Q) to receive diagnostics.
- Add and configure devices by setting address, direct representation variables (%Q) to disable the relations, communication time-outs, etc.
- Add and configure MODBUS relations, specifying the data type and MODBUS function, time-outs, direct representation variables (%Q) to receive diagnostics of the relation and other to receive/write the data, amount of data to be transmitted and relation polling.

The descriptions of each configuration are listed below in this section.

5.5.2.2.1. General Parameters of MODBUS Master Protocol - setting by Direct Representation (%Q)

The General parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:
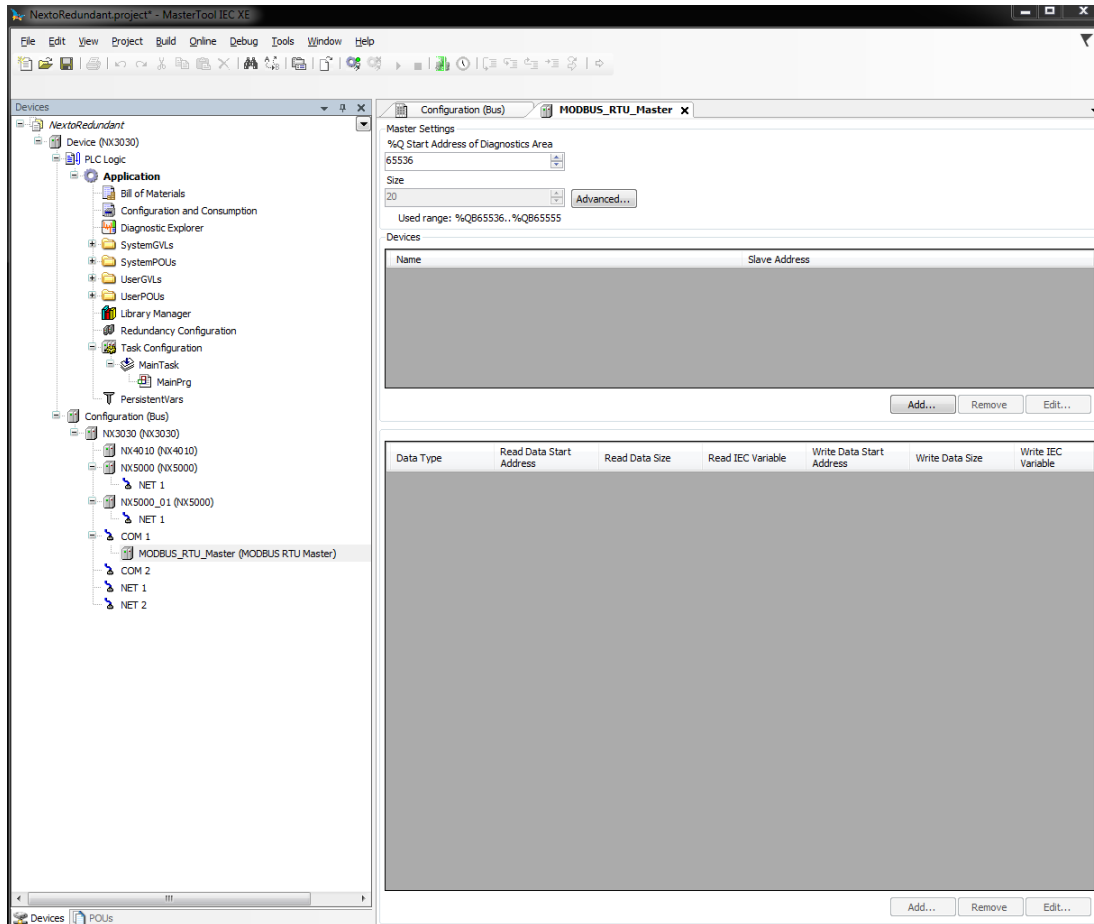
Figure 50: MODBUS RTU Master Setup Screen

Direct representation variables (%Q) for the protocol diagnostic:

Configuration	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Initial address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics area	20	Disabled for editing

Table 64: MODBUS RTU Master Configuration

Notes:

Initial Address of Diagnostics in %Q: this field is limited by the size of outputs variables (%Q) addressable memory of each CPU, which can be found in section Memory.

Default Value: the factory default value cannot be set to the *%Q Start Address of Diagnostics Area* field, because the creation of a Protocol instance may be held at any time on application development. The MasterTool IEC XE software itself allocate a value, from the range of output variables of direct representation (%Q), not used yet.

The diagnostics and MODBUS protocol commands are described in Table 56.

altus

The communication times of the MODBUS Master protocol, found on the button *Advanced...* in the configuration screen are divided into *Send Delay* and *Minimum Interframe*, further details are described in section MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration.

5.5.2.2.2. *Devices Configuration – Configuration for Direct Representation (%Q)*

The configuration of the devices, viewed in figure below, comprises the following parameters:
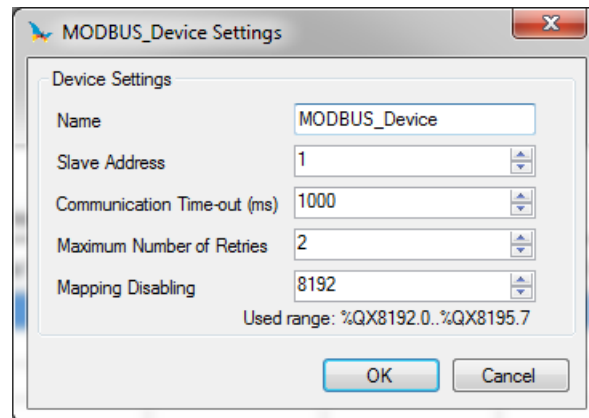
Figure 51: Device Configuration

Configuration	Description	Default Value	Options
Name	Name of the instance	MODBUS_Device	Identifier, according to IEC 61131-3
Slave Address	The MODBUS slave address	1	0 to 255
Communication Time-out (ms)	Sets the time-out of the application level	1000	10 to 65535
Maximum Number of Retries	Sets the number of retries before reporting a communication error	2	0 to 9
Mapping Disabling	Initial address used to disable MODBUS relations	-	0 to 2147483644

Table 65: Device Configuration - MODBUS Master

Notes:

Instance Name: this field is the identifier of the device, which is checked according to IEC 61131-3, i.e. does not allow spaces, special characters and start with numeral character. It's limited in 24 characters.

Mapping Disabling: composed of 32 bits, used to disable, individually, the 32 MODBUS relations configured in *Device Mappings* space. The relation is disabled when the bit, corresponding to relation, is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of outputs variables (%Q) addressable memory of each CPU, which can be found in section Memory.

Default Value: the factory default value cannot be set to the *Mapping Disabling* field, because the creation of a Protocol instance may be held at any time on application development. The MasterTool IEC XE software itself allocate a value, from the range of output variables of direct representation (%Q), not used yet.

For further details on the *Slave Address*, *Communication Time-out* and *Maximum Number of Retries* parameters see notes in section Devices Configuration – Symbolic Mapping configuration.

5.5.2.2.3. Mappings Configuration – Configuration for Direct Representation (%Q)

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:
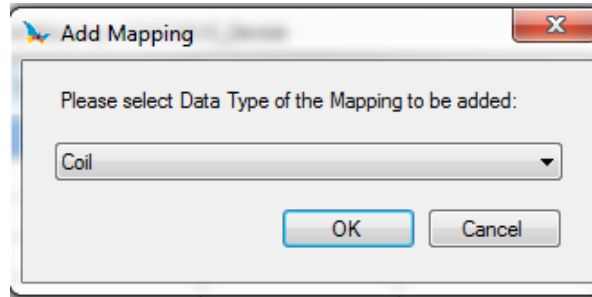
Figure 52: MODBUS Data Type
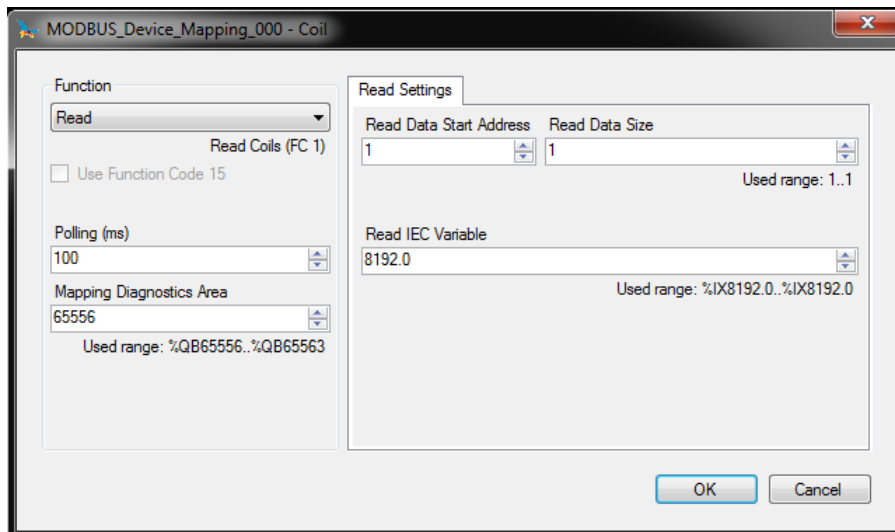
Figure 53: MODBUS Function

In table below, the number of factory default settings and the values for the column Options, may vary according to the data type and MODBUS function (FC).

Configuration	Description	Default Value	Options
Function	MODBUS function type	Read	Read Write Read/Write Mask Write
Polling (ms)	Communication period (ms)	100	0 to 3600000
Mapping Diagnostics Area	Initial address of the MODBUS relation diagnostics (%Q)	-	0 to 2147483640
Read Data Start Address	Initial address of the MODBUS read data	1	1 to 65536
Read Data Size	Number of MODBUS read data	-	Depends on the function used

altus

Configuration	Description	Default Value	Options
Read IEC Variable	Initial address of the read variables (%I)	-	0 to 2147483646
Write Data Start Address	Initial address of the MOD-BUS write data	1	1 to 65536
Write Data Size	Number of MODBUS write data	-	Depends on the function used
Write IEC Variable	Initial address of the write variables (%Q)	-	0 to 2147483647
Mask Write IEC Variables	Initial address of the variables for the write mask (%Q)	-	0 to 2147483644

Table 66: Device Mapping

Notes:

Function: the available data types are detailed in the Table 85 and MODBUS functions (FC) are available in the Table 83.

Polling: this parameter indicates how often the communication set for this relation must be executed. At the end of communication will be awaited a time equal to the configured polling and after, will be performed a new communication as soon as possible.

Mapping Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in the section Memory. The configured MODBUS relations diagnostics are described in Table 62.

Read/Write Data Size: details of the data size supported by each function are described in the notes of the section Requests Configuration – Symbolic Mapping Settings.

> **ATTENTION**
>
> When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

Read IEC Variable: if the MODBUS data type is *Coil* or *Input Status* (bit), the initial address of the IEC reading variables will have the format *%IX10.1*, for example. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the initial address of the IEC reading variables will be *%IW*. This field is limited by the size of input variables addressable memory *(%I)* at CPU, which can be found in the section Memory.

Write IEC Variable: if the MODBUS data type is *Coil*, the initial address of the IEC writing variables will have the format *%QX10.1*, for example. However, if the MODBUS data type is *Holding Register* (16 bits), the initial address of the IEC writing variables will be *%QW*. This field is limited by the size of output variables addressable memory *(%Q)* at CPU, which can be found in the section Memory.

Write Mask: the function *Mask Write* (FC 22), employs a logic between the value already written and the two words that are configured in this field using *%QW(0)* for the AND mask and *%QW(2)* for the OR mask; allowing the user to handle the word. This field is limited by the size of output variables addressable memory *(%Q)* of each CPU, which can be found in the section Memory.

Default Value: the factory default value cannot be set for the *Mapping Diagnostics Area*, *Read IEC Variable*, *Write IEC Variable* and *Mask Write IEC Variables* fields, since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables *(%Q)*, still unused. Factory default cannot be set to the *Read/Write Data Size* fields, as they will vary according to the MODBUS data type selected.

> **ATTENTION**
>
> Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS RTU Master 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.

altus

5.5.3. 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.

There are two ways to configure this protocol. The first one makes use of direct representation (%Q), in which the variables are defined by your address. The second one, through symbolic mapping, where the variables are defined by your name.

Independent of the configuration mode, the steps to insert an instance of the protocol and configure the serial interface are equal. 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 COM 1 or COM 2 (or both, in cases of two communication networks). 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 Interfaces Configuration section.

5.5.3.1. MODBUS Slave Protocol Configuration via Symbolic Mapping

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

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

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

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as.
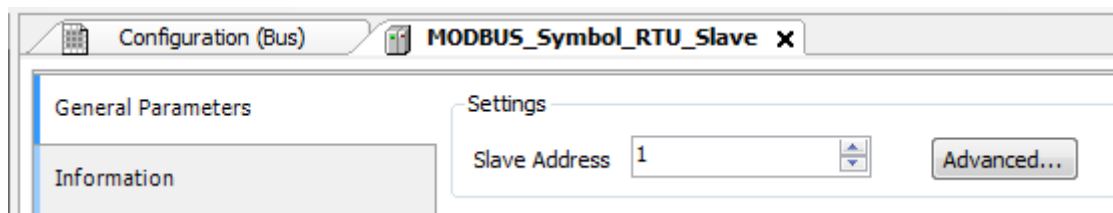
Figure 54: MODBUS RTU Slave Configuration Screen

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

Table 67: 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.
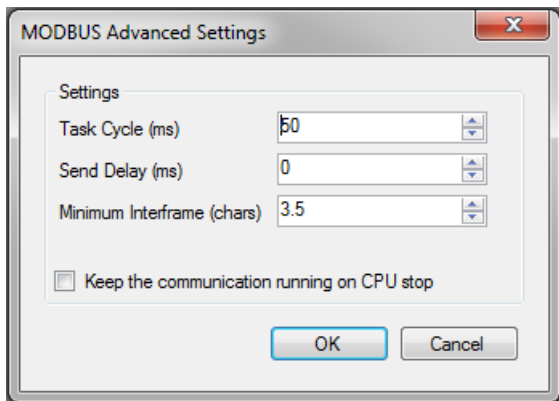
Figure 55: Modbus Slave Advanced Configurations

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
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 68: 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 (where "n" is the configured value in the *%Q Start Address of Diagnostics Area*):

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_SLAVE_1.*	Size	Description
Diagnostic Bits:			
%QX(n).0	tDiag. bRunning	BIT	The slave is in execution mode.
%QX(n).1	tDiag. bNotRunning	BIT	The slave is not in execution (see bit: bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the slave was interrupted by the user through command bits.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnosis.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnosis.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnosis.
%QX(n).6	tDiag. bModuleFailure	BIT	Discontinued diagnosis.
%QX(n).7	tDiag. bDiag_7_reserved	BIT	Reserved.
Error codes:			
%QB(n+1)	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

altus

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_SLAVE_1.*	Size	Description
			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:			
%QX(n+2).0	tCommand. bStop	BIT	Stop slave.
%QX(n+2).1	tCommand. bRestart	BIT	Restart slave.
%QX(n+2).2	tCommand. bResetCounter	BIT	Restart diagnostics statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved.
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved.
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved.
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved.
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved.
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
Communication Statistics:			
%QW(n+4)	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).
%QW(n+6)	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). Exception codes: 1: the function code (FC) is legal, but not supported. 2: relation not found in these MODBUS data. 3: illegal value for this field. 128: the master/client hasn't right for writing or reading. 129: the MODBUS relation is disabled.
%QW(n+8)	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).

altus

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_SLAVE_1.*	Size	Description
%QW(n+10)	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).
%QW(n+12)	tStat. wRXOverrunErrors	WORD	Counter of frames with overrun errors during reception – UART FIFO or RX line – (0 to 65535).
%QW(n+14)	tStat. wRXIncompleteFrames	WORD	Counter of frames with construction errors, parity or failure during reception (0 to 65535).
%QW(n+16)	tStat. wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using the RTS/CTS handshake, during the transmission (0 to 65535).
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved.

Table 69: MODBUS RTU Slave Diagnostic

Note:

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

5.5.3.1.2. *Configuration of the Relations – Symbolic Mapping Setting*

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

Figure 56: 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	-	-

Configuration	Description	Default	Options
Data Size	MODBUS data size	-	1 to 65536
Data Range	Data address range configured	-	-

Table 70: 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 71: 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.

5.5.3.2. MODBUS Slave Protocol Configuration via Direct Representation (%Q)

To configure this protocol using Direct Representation (%Q), you must perform the following steps:

- Configure the general parameters of MODBUS slave protocol, such as: communication times, address and direct representation variables (%Q) to receive diagnostics and control relations.
- Add and configure MODBUS relations, specifying the MODBUS data type, direct representation variables (%Q) to receive/write the data and amount of data to communicate.

The descriptions of each setting are listed below, in this section.

5.5.3.2.1. *General Parameters of MODBUS Slave Protocol – Configuration via Direct Representation (%Q)*

The general parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:
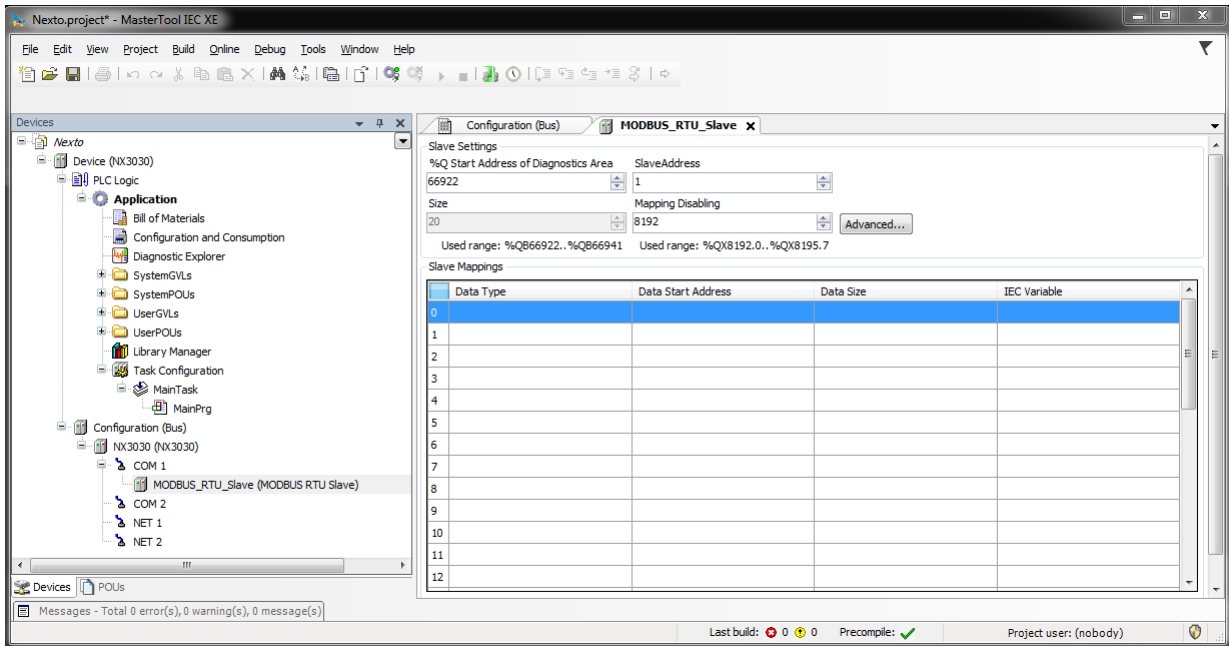
Figure 57: MODBUS RTU Slave Configuration Screen by Direct Representation

Address and direct representation variables (%Q) to control relations and diagnostics:

Configuration	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Initial address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics area	-	Disabled for editing
Slave Address	MODBUS slave address	1	1 to 255
Mapping Disabling	Initial address used to disable MODBUS relations	-	0 to 2147483644

Table 72: Address and Direct Representation Variables Settings

Notes:

%Q Start Address of Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found in section Memory.

Slave Address: it is important to note that the Slave accepts requests broadcast, when the master sends a command with the address set to zero. Moreover, in accordance with standard MODBUS, the valid address range for slaves is 1 to 247. The addresses 248 to 255 are reserved.

Mapping Disabling: composed of 32 bits, used to disable, individually, the 32 MODBUS relations configured in *Slave Mappings* space. The relation is disabled when the corresponding bit is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found on Memory section.

Default Value: the factory default value cannot be set for the *%Q Start Address of Diagnostics Area* and *Mapping Disabling* fields, since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused.

The MODBUS Slave by Direct Representation protocol stops communicating while the CPU is in STOP or stopped at a breakpoint.

The MODBUS protocol diagnostics and commands are described in the Table 69.

The communication times of the MODBUS Slave protocol, found on the button *Advanced...* of the configuration screen, are described in Table 68.

5.5.3.2.2. Mappings Configuration – Configuration via Direct Representation (%Q)

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:

altus
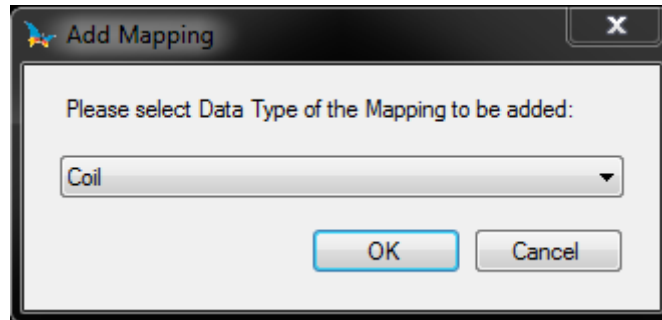
Figure 58: Adding MODBUS Relations
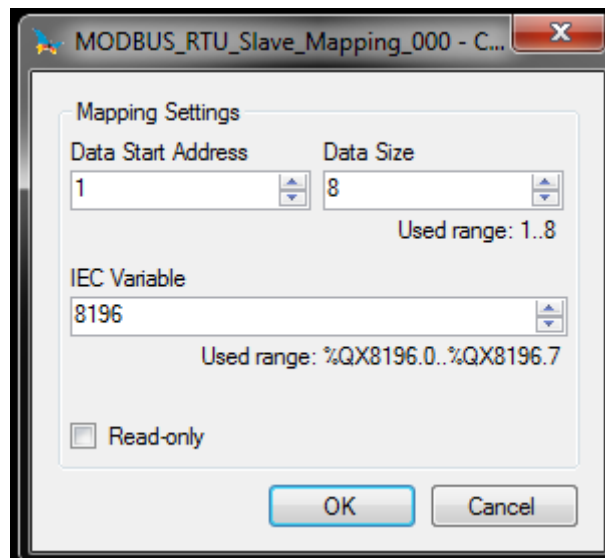
Figure 59: Configuring the MODBUS Relation

Configuration	Description	Default Value	Options
Data Type	MODBUS data type	Coil	Coil (1 bit) Holding Register (16 bits) Input Register (16 bits) Input Status (1 bit)
Data Start Address	Initial address of the MOD-BUS data	1	1 to 65536
Data Size	Number of MODBUS data	-	1 to 65536
IEC Variable	Initial address of variables (%Q)	-	0 to 2147483647
Read-only	Only allows reading	Disabled	Enabled or disabled

Table 73: Slave Mappings

Notes:

Options: the values written in the column *Options* may vary according with the configured MODBUS data.

Data Size: the value of *Data Size* defines the maximum amount of data that a MODBUS relation 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, i.e. when selected *Coil* or *Input Status*, the Data Size field must be a multiple of eight. Also, the maximum amount must not exceed the size of output addressable memory and not assign the same values used in the application.

> **ATTENTION**
>
> When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

IEC Variable: in case the MODBUS data type is *Coil* or *Input Status* (bit), the IEC variables initial address will be in the format *%QX10.1*. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the IEC variables initial address will be in the format *%QW*. This field is limited by the memory size of the addressable output variables (%Q) from each CPU, which can be seen on Memory section.

Read-only: when enabled, it only allows the communication master to read the variable data. It does not allow the writing. This option is valid for the writing functions only.

Default Value: the default value cannot be defined for the *IEC Variable* field since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused. The default cannot be defined for the *Data Size* field as it will vary according to selected MODBUS data type.

In the previously defined relations, the maximum MODBUS data size can be 65535 (maximum value configured in the *Data Size* field). However, the request which arrives in the MODBUS RTU Slave must address a subgroup of this mapping and this group must have, at most, the data size depending on the function code which is defined below:

- Read Coils (FC 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (FC 3): 125
- Read Input Registers (FC 4): 125
- Write Single Coil (FC 5): 1
- Write Single Holding register (FC 6): 1
- Force Multiple Coils (FC 15): 1968
- Write Holding Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Holding Registers (FC 23):
  - Read: 121
  - Write: 121

> **ATTENTION**
>
> Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of a Slave 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.

5.5.4. MODBUS Ethernet

The multi-master communication allows the Nexto CPUs to read or write MODBUS variables in other controllers or HMIs compatible with the MODBUS TCP protocol or MODBUS RTU via TCP. The Nexto CPU can, at the same time, be client and server in the same communication network, or even have more instances associated to the Ethernet interface. It does not matter if they are MODBUS TCP or MODBUS RTU via TCP, as described on Table 51.

The figure below represents some of the communication possibilities using the MODBUS TCP protocol simultaneously with the MODBUS RTU via TCP protocol.
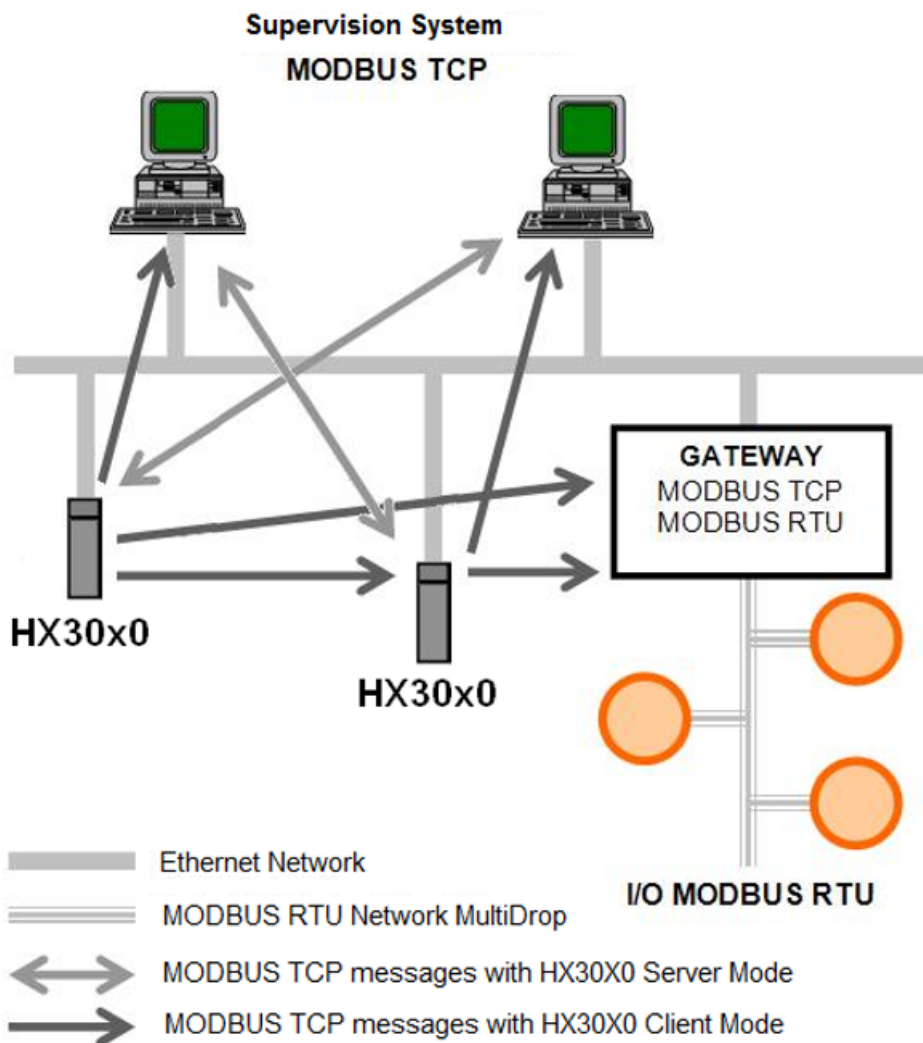
Figure 60: MODBUS TCP Communication Network

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 74: 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 75: 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 Interfaces Configuration.

5.5.5. 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).

There are two ways to configure this protocol. The first one makes use of *direct representation (%Q)*, in which the variables are defined by your address. The second one, through *symbolic mapping*, where the variables are defined by your name.

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.

5.5.5.1. MODBUS Ethernet Client Configuration via Symbolic Mapping

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

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

5.5.5.1.1. MODBUS Client Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol configuration initial screen (figure below), are defined as:
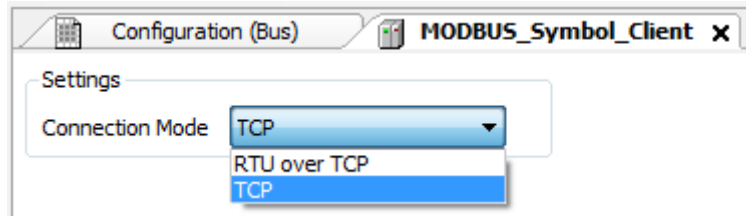
Figure 61: MODBUS Client General Parameters Configuration Screen

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

Table 76: 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 (where "n" is the configured value in the *%Q Start Address of Diagnostics Area*):

Direct Repre-sentation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_CLIENT_1.*	Size	Description
Diagnostic Bits:			
%QX(n).0	tDiag. bRunning	BIT	The client is in execution mode.
%QX(n).1	tDiag. bNotRunning	BIT	The client is not in execution mode (see bit bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled, as the client was interrupted by the user through command bits.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnostics.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnostics.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnostics.
%QX(n).6	tDiag. bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present.
%QX(n).7	tDiag. bAllDevicesCommFailure	BIT	Indicates that all devices configured in the Client are in failure.
%QB(n+1)	byDiag_1_reserved	BYTE	Reserved.

altus

Direct Repre-sentation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_CLIENT_1.*	Size	Description
Command bits, automatically initialized:			
%QX(n+2).0	tCommand. bStop	BIT	Stop client.
%QX(n+2).1	tCommand. bRestart	BIT	Restart client.
%QX(n+2).2	tCommand. bResetCounter	BIT	Restart the diagnostic statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved.
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved.
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved.
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved.
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved.
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
Communication Statistics:			
%QW(n+4)	tStat. wTXRequests	WORD	Counter of number of requests transmitted by the client (0 to 65535).
%QW(n+6)	tStat. wRXNormalResponses	WORD	Counter of normal answers received by the client (0 to 65535).
%QW(n+8)	tStat. wRXExceptionResponses	WORD	Counter of answers with exception code (0 to 65535).
%QW(n+10)	tStat. wRXIllegalResponses	WORD	Counter of illegal answers received by the client – invalid syntax, invalid CRC or not enough bytes received (0 to 65535).
%QW(n+12)	tStat. wDiag_12_reserved	WORD	Reserved.
%QW(n+14)	tStat. wDiag_14_reserved	WORD	Reserved.
%QW(n+16)	tStat. wDiag_16_reserved	WORD	Reserved.
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved.

Table 77: MODBUS Client Protocol Diagnostics

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

5.5.5.1.2. *Device Configuration – Configuration via Symbolic Mapping*

The devices configuration, shown on figure below, follows the following parameters:
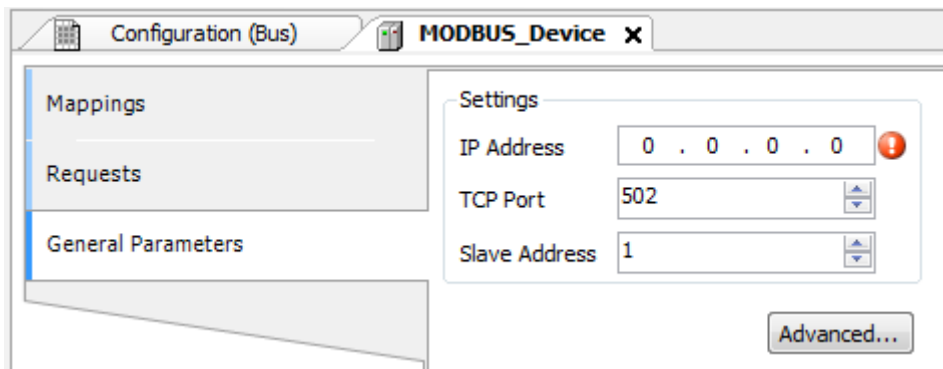
Figure 62: Device General Parameters Settings

Configuration	Description	Default	Options
IP Address	Server IP address	0.0.0.0	1.0.0.1 to 223.255.255.255
TCP Port	TCP port	502	2 to 65534
Slave Address	MODBUS Slave address	1	0 to 255

Table 78: 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	Connection is closed after an inactive time of (s): 10 to 3600.	Connection is closed after a time-out. 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	3600

Table 79: MODBUS Client Advanced Configurations

altus

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:

- Connection is closed after a time-out or Connection is never closed in normal situations: Those options 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.

5.5.5.1.3. Mappings Configuration – Configuration via Symbolic Mapping

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:
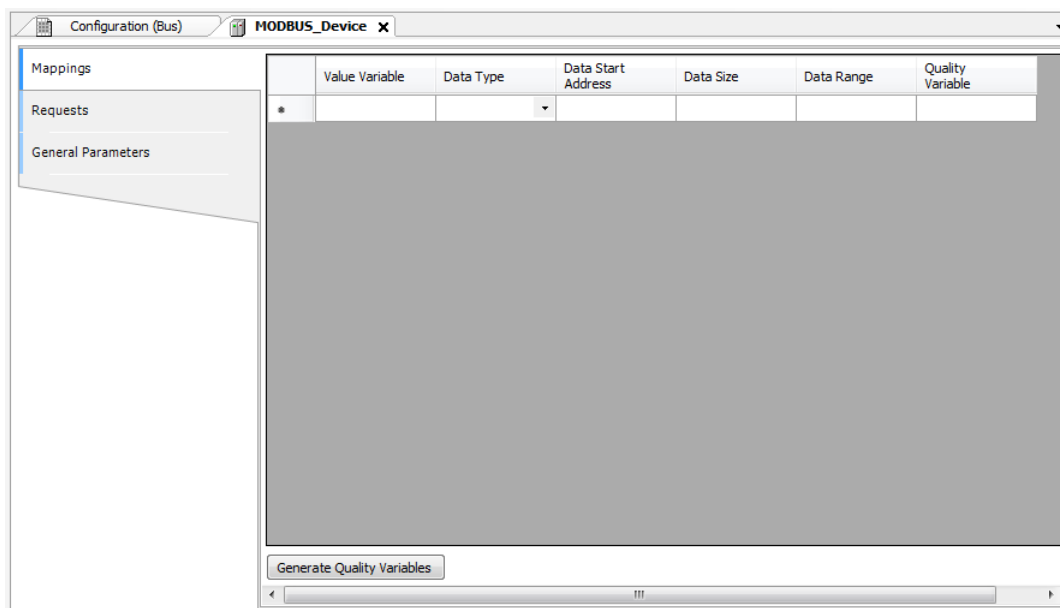
Figure 63: MODBUS Data Type

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 80: 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 81: 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.

5.5.5.1.4. Requests Configuration – Configuration via Symbolic Mapping

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

altus
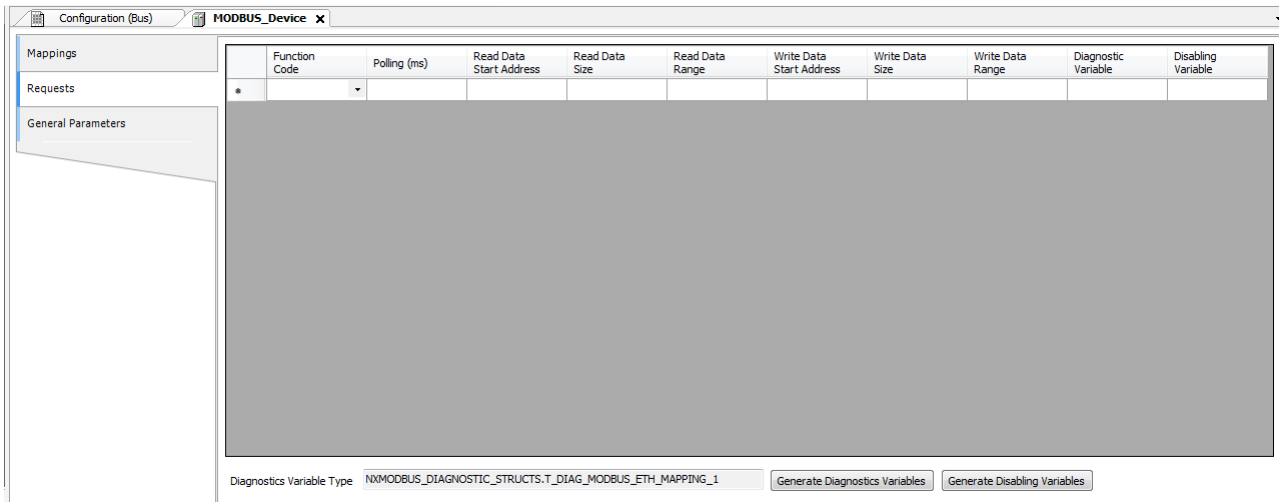
Figure 64: MODBUS Data Request Screen

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

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

Table 82: 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 83: 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 62 ("n" is the value configured in the *%Q Start Address of Diagnostics Area* field).

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_MAPPING_1.*	Size	Description
Communication Status Bits:			
%QX(n).0	byStatus. bCommIdle	BIT	Communication idle (waiting to be executed).
%QX(n).1	byStatus. bCommExecuting	BIT	Active communication.
%QX(n).2	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 timeout. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true.
%QX(n).3	byStatus. bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
%QX(n).4	byStatus. bCommOk	BIT	Communication terminated previously was held successfully.
%QX(n).5	byStatus. bCommError	BIT	Communication terminated previously had an error. Check error code.
%QX(n).6	byStatus. bCommAborted	BIT	Previously terminated communication was interrupted due to connection failure.
%QX(n).7	byStatus. bDiag_7_reserved	BIT	Reserved.
Last error code (enabled when bCommError = true):			
%QB(n+1)	eLastErrorCode	MASTER_ERROR _CODE (BYTE)	Informs the possible cause of the last error in the MODBUS mapping. Consult Table 85 for further details.
Last exception code received by master:			
%QB(n+2)	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)*

altus

Direct Rep-resentation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_MAPPING_1.*	Size	Description
Communication statistics:			
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
%QW(n+4)	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.
%QW(n+6)	wCommErrorCounter	WORD	Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero.

Table 84: 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 85: MODBUS Relations Error Codes

altus

> **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.

5.5.5.2. MODBUS Ethernet Client configuration via Direct Representation (%Q)

To configure this protocol using direct representation (%Q), the following steps must be performed:

- Configure the general parameters of the MODBUS protocol, such as: communication times and direct representation variables (%Q) to receive diagnostics.
- Add and configure devices by setting address, direct representation variables (%Q) to disable the relations, communication time-outs, etc.
- Add and configure MODBUS relations, specifying the data type and MODBUS function, time-outs, direct representation variables (%Q) to receive diagnostics of the relation and other to receive/write the data, amount of data to be transmitted and relation polling.

The descriptions of each configuration are listed below in this section.

5.5.5.2.1. General parameters of MODBUS Protocol Client - configuration for Direct Representation (%Q)

The General parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:
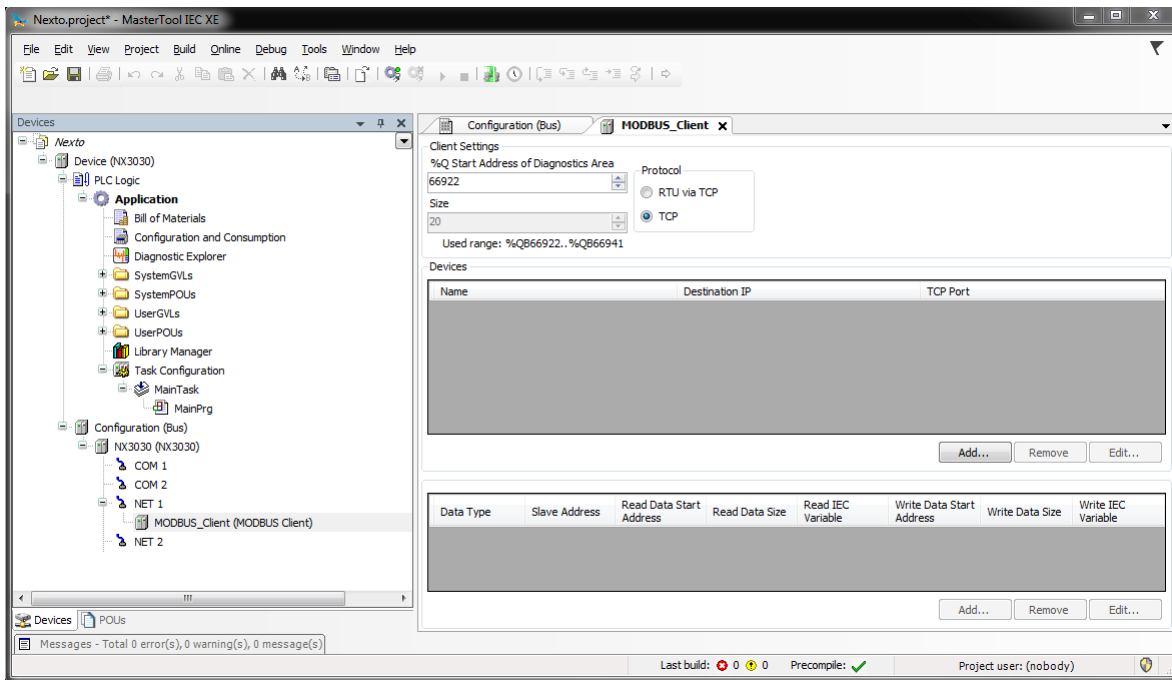
Figure 65: MODBUS Client Setup Screen

Protocol selection and direct representation variables (%Q) for diagnostics:

Setting	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Initial address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics	20	Disabled for editing
Protocol	Protocol selection	TCP	RTU via TCP TCP

Table 86: MODBUS Client settings

Notes:

%Q Start Address of Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in section Memory.

Default Value: the default value cannot be defined for the *%Q Start Address of Diagnostics Area* field since the creation of a protocol instance can be made at any moment within the application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused.

The diagnostics and MODBUS commands are described in Table 77.

5.5.5.2.2. *Device Configuration – Configuration via Direct Representation (%Q)*

The configuration of the devices, viewed in figure below, comprises the following parameters:
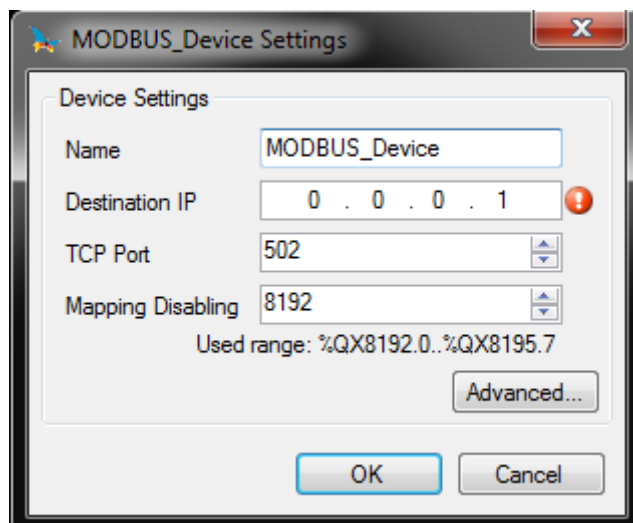
Figure 66: Configuring MODBUS Client

Configuration	Description	Factory default	Options
Name	Name of the instance	MODBUS_Device	Identifier, according to IEC 61131-3
Destination IP	IP address of the server	0. 0. 0.1	1.0.0.1 to 223.255.255.255
TCP Port	TCP Port	502	2 to 65534
Mapping Disabling	Initial address used to disable MODBUS relations	-	Any address of the %Q area, limited by the CPU model

Table 87: Configuration of Client Devices

Notes:

Instance Name: this field is the identifier of the device, which is checked according to IEC 61131-3, i.e. it does not allow spaces, special characters and starting with numeral character. It is limited to 24 characters.

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.

Mapping Disabling: composed of 32 bits, it is used to disable, individually, the 32 MODBUS relations configured in *Device Mappings* space. The relation is disabled when the corresponding bit is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in section Memory.

Default Value: factory default cannot be set for the *Mapping Disabling* field, since the creation of a protocol instance can be made at any moment within the application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused.

Communication Time-out: the settings present on the button *Advanced...* on the TCP connection, are described in the notes of the section Device Configuration – Configuration via Symbolic Mapping.

5.5.5.2.3. Mapping Configuration – Configuration via Direct Representation (%Q)

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:
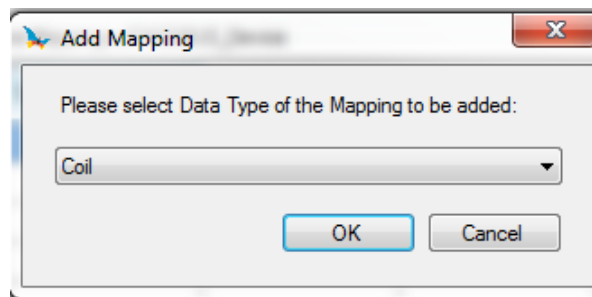
Figure 67: MODBUS Data Type
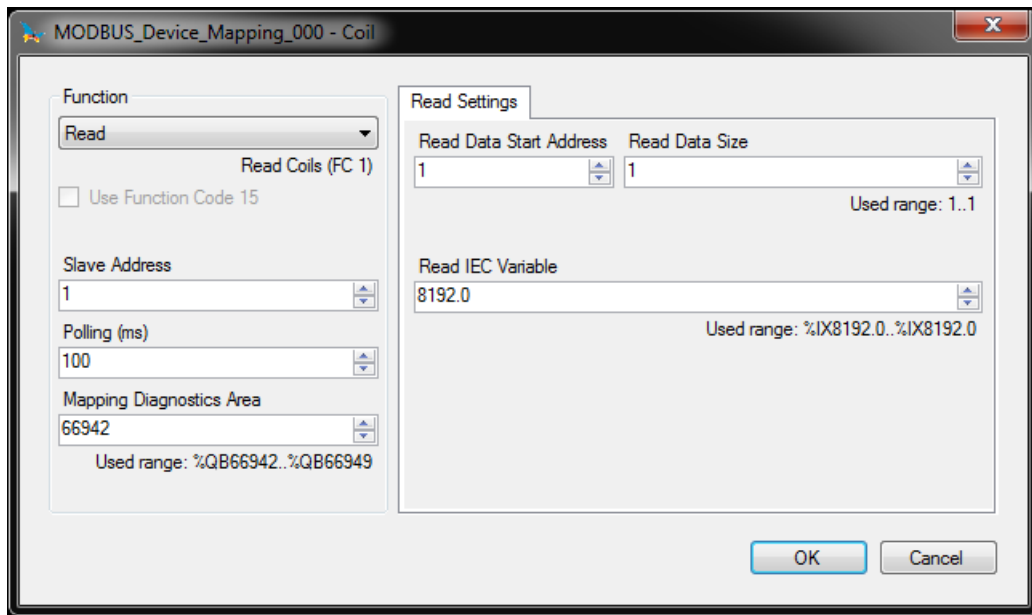
Figure 68: MODBUS Function

In table below, the number of factory default settings and the values for the column Options, may vary according to the data type and MODBUS function (FC).

Configuration	Description	Default Value	Options
Function	MODBUS function type	Read	Read Write Read/Write Mask Write
Slave Address	MODBUS slave address	1	0 to 255
Polling (ms)	Period of communication (ms)	100	0 to 3600000
Mapping Diagnostics Area	Starting address of MODBUS interface diagnostics	-	0 to 2147483640
Read Data Start Address	Starting address of the read MODBUS data	1	1 to 65536
Read Data Size	Number of read MODBUS data	-	Depends on the function used
Read IEC Variable	Starting address of the read variables (%I)	-	0 to 2147483647
Write Data Start Address	Starting address of MODBUS writing data	1	1 to 65536
Write Data Size	Number of MODBUS writing data	-	Depends on the function used
Write IEC Variable	Starting address of the write variables (%Q)	-	0 to 2147483647
Mask Write IEC Variables	Starting address of variables for write mask (%Q)	-	0 to 2147483644

Table 88: Device Mapping

Notes:

Device Mappings Table: the number of settings and values described in the column Options may vary according to the data type and MODBUS function.

Slave Address: typically, the address 0 is used when the server is a MODBUS RTU or MODBUS RTU via TCP Gateway, and the same broadcasts the request to all network devices. When the address 0 is used, the client doesn't waits for a response and its use serves only to written commands. Moreover, in accordance with MODBUS standard, the valid address range for slaves is 0 to 247, and addresses 248 to 255 are reserved.

Polling: this parameter indicates how often the communication set for this relation must be executed. At the end of communication will be awaited a time equal to the configured polling and after, will be performed a new communication as soon as possible.

Mapping Diagnostic Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in the section Memory . The configured MODBUS relations diagnostics are described in Table 62.

Size of the Read and Write Data: details of the size of the data supported by each function are described in the notes of Requests Configuration – Symbolic Mapping Settings section.

> **ATTENTION**
>
> When accessing the communication data memory is between devices with different endian-ism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

Read IEC Variable: in case the MODBUS data type is *Coil* or *Input Status* (bit), the IEC variables initial address will be in the format *%IX10.1*. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the IEC variables initial address will be in the format *%IW*. This field is limited by the memory size of the addressable input variables *(%I)* from each CPU, which can be seen on Memory section.

Write IEC Variable: in case the MODBUS data type is *Coil* (bit), the IEC variables initial address will be in the format *%QX10.1*. However, if the MODBUS data type is *Holding Register* (16 bits), the IEC variables initial address will be in the format *%QW*. This field is limited by the memory size of the addressable output variables *(%Q)* from each CPU, which can be seen on Memory section.

Write Mask of IEC Variables: the *Mask Write Register* function (FC 22) employs a logic between the value already written and the two words that are configured in this field using *%QW(0)* for the AND mask and *%QW(2)* for the OR mask; allowing the user to handle the word. This field is limited by the size of output variables addressable memory *(%Q)* of each CPU, which can be found in the section Memory.

Default Value: the factory default value cannot be set for the *Mapping Diagnostics Area*, *Read IEC Variable*, *Write IEC Variable* and *Mask Write IEC Variables* fields, since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables *(%Q)*, still unused. Factory default cannot be set to the *Read/Write Data Size* fields, as they will vary according to the MODBUS data type selected.

> **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.

5.5.5.3. 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.

altus

5.5.6. 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).

There are two ways to configure this protocol. The first one makes use of *direct representation (%Q)*, in which the variables are defined by your address. The second one, through *symbolic mapping*, where the variables are defined by your name.

The procedure to insert an instance of the protocol is found in detail in the MasterTool IEC XE User Manual – MU299609.

5.5.6.1. MODBUS Server Ethernet Protocol Configuration for Symbolic Mapping

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

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

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

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

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as.
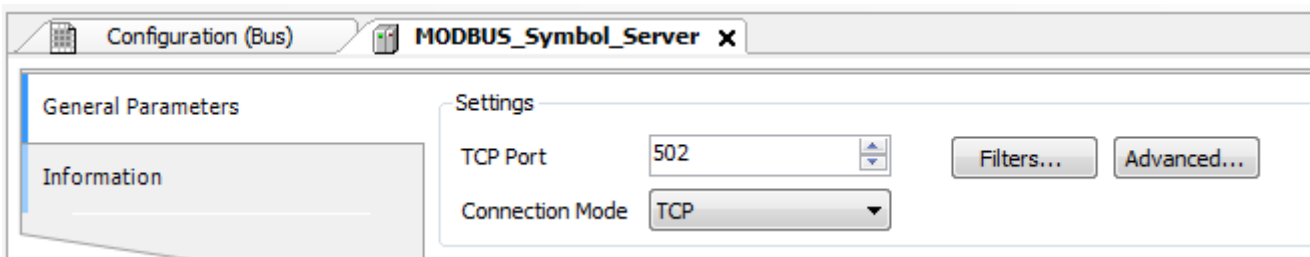
Figure 69: MODBUS Server General Parameters Configuration Screen

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

Table 89: MODBUS Server General Configurations

Notes:

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

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

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

Table 90: 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*.
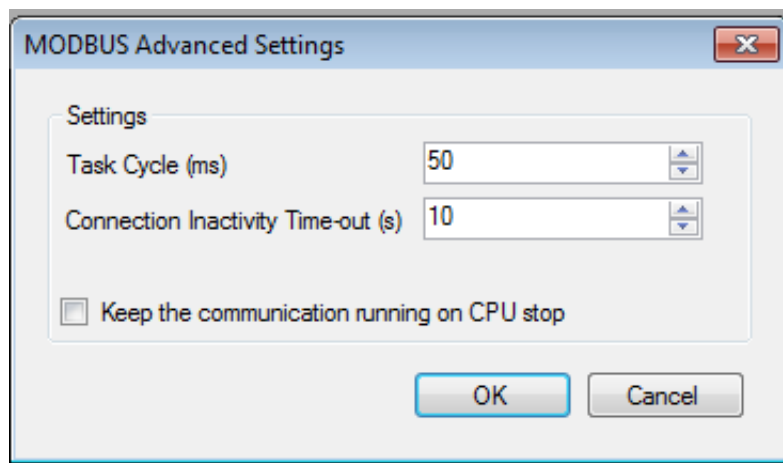
Figure 70: MODBUS Server Advanced Settings Configuration Screen

altus

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

Table 91: MODBUS Server Advanced Configurations

Notes:

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

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

5.5.6.1.2. 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 (n is the value configured in the *%Q Start Address of Diagnostics Area* field):

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_SERVER_1 .*	Size	Description
Diagnostic bits:			
%QX(n).0	tDiag. bRunning	BIT	The server is running.
%QX(n).1	tDiag. bNotRunning	BIT	The server is not running (see bit bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled, because the server was interrupted by the user through the command bit.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnostic.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnostic.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnostic.

altus

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_SERVER_1 .*	Size	Description
%QX(n).6	tDiag. bModuleFailure	BIT	Discontinued diagnostic.
%QX(n).7	tDiag. bDiag_7_reserved	BIT	Reserved.
%QB(n+1)	byDiag_1_reserved	BYTE	Reserved.
Command bits, restarted automatically:			
%QX(n+2).0	tCommand. bStop	BIT	Stop the server.
%QX(n+2).1	tCommand. bRestart	BIT	Restart the server.
%QX(n+2).2	tCommand. bResetCounter	BIT	Reset diagnostics statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved.
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved.
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved.
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved.
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved.
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
Communication statistics:			
%QW(n+4)	tStat. wActiveConnections	WORD	Number of established connections between client and server (0 to 64).
%QW(n+6)	tStat. wTimeoutClosedConnections	WORD	Connections counter, between the client and server, interrupted after a period of inactivity - time-out (0 to 65535).
%QW(n+8)	tStat. wClientClosedConnections	WORD	Connections counter interrupted due to customer request (0 to 65535).
%QW(n+10)	tStat. wRXFrames	WORD	Ethernet frames counter received by the server. An Ethernet frame can contain more than one request (0 to 65535).
%QW(n+12)	tStat. wRXRequests	WORD	Requests received by the server counter and answered normally (0 to 65535).
%QW(n+14)	tStat. wTXExceptionResponses	WORD	Requests received by the server counter and answered with exception codes (0 to 65535). The exception codes are listed below: 1: the function code (FC) is legal, but not supported. 2: relation not found in these data MODBUS. 3: illegal value for the address. 128: the master/client has no right to read or write. 129: MODBUS relation is disabled.

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_SERVER_1 .*	Size	Description
%QW(n+16)	tStat. wRXIllegalRequests	WORD	Illegal requests counter (0 to 65535).
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved.

Table 92: MODBUS Server Diagnostics

Note:

Counters: all counters of the MODBUS Ethernet Server Diagnostics return to zero when the limit value 65535 is exceeded.

5.5.6.1.3. Mapping Configuration – Configuration via Symbolic Mapping

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

Figure 71: 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 93: 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.

5.5.6.2. MODBUS Server Ethernet Protocol Configuration via Direct Representation (%Q)

To configure this protocol using *Direct Representation (%Q)*, the user must perform the following steps:

- Configure the general parameters of MODBUS Server Protocol, such as: communication times, address and direct representation variables (%Q) to receive the diagnostics and control relation.
- Add and configure MODBUS relations, specifying the MODBUS data type, direct representation variables (%Q) to receive/write the data and amount of data to be reported.

The descriptions of each configuration are listed below in this section.

5.5.6.2.1. General Parameters of MODBUS Server Protocol – Configuration via Direct Representation (%Q)

The general parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:
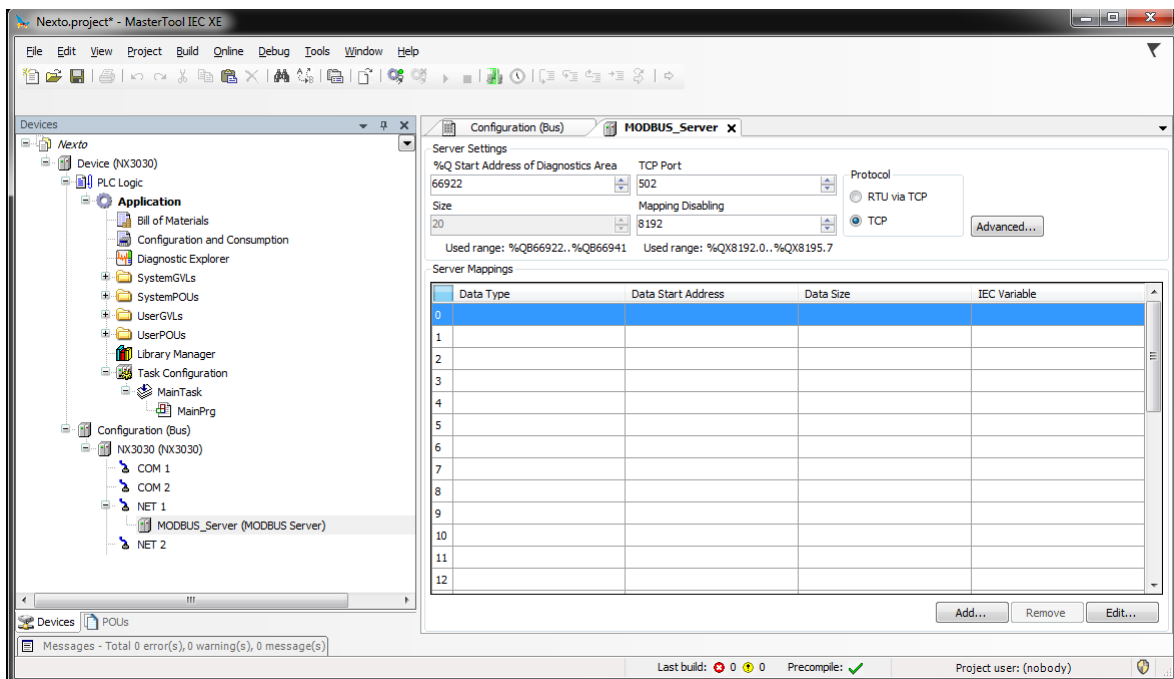
Figure 72: MODBUS Server Setup Screen

TCP port, protocol and direct representation variables (%Q) to control relations and diagnostics:

Configuration	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Starting address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics	20	Disabled for editing
TCP Port	TCP Port	502	2 to 65534
Mapping Disabling	Starting address used to disable MODBUS relations	-	0 to 2147483644
Protocol	Protocol selection	TCP	RTU via TCP TCP

Table 94: Settings to control relations and diagnostics

Notes:

%Q Start Address of Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in section Memory.

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.

Mapping Disabling: composed of 32 bits, used to disable, individually, the 32 MODBUS relations configured in *Server Mappings* space. The relation is disabled when the corresponding bit is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found on Memory section.

Default Value: the factory default value cannot be set to the *%Q Start Address of Diagnostics Area* and *Mapping Disabling* fields, because the creation of a Protocol instance may be held at any time on application development. The MasterTool IEC XE software itself allocate a value, from the range of output variables of direct representation (%Q), not used yet.

The communication times of the MODBUS Server protocol, found on the *Advanced...* button of the configuration screen, are divided into: *Task Cycle (ms)* and *Connection Inactivity Time-out (s)*. Further details are described in MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping section.

The diagnostics and MODBUS commands are described in Table 92.

5.5.6.2.2.   Mapping Configuration – Configuration via Direct Representation (%Q)

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:
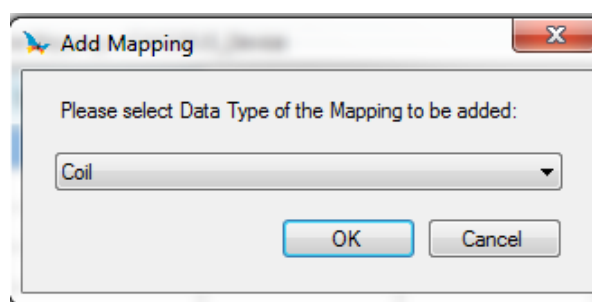
Figure 73: MODBUS Data Type

altus
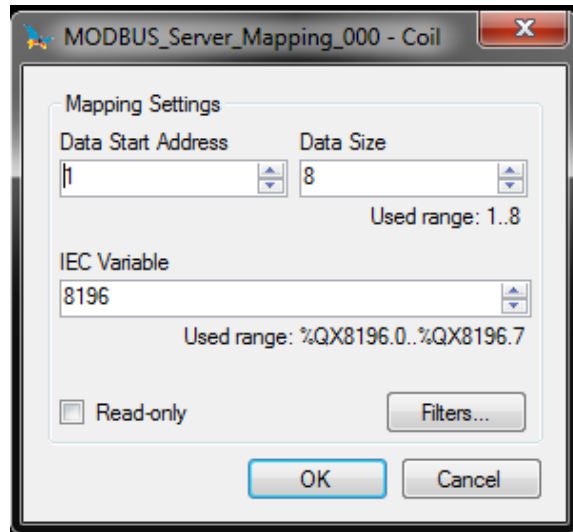
Figure 74: MODBUS Server Function

Configuration	Description	Default	Options
Data Type	MODBUS data type	Coil	Coil (1 bit) Holding Register (16 bits) Input Status (1 bit) Input Register (16 bits)
Data Start Address	MODBUS data initial address	1	1 to 65536
Data Size	MODBUS data quantity	8	1 to 65536 (Holding Register and Input Register) 8 to 65536 (Coil and Input Status)
IEC Variable	Variables initial address (%Q)	-	0 to 2147483647
Read-only	Allow reading only	Disabled	Enabled or Disabled

Table 95: Server Mappings

Notes:

Options: the values written in the column *Options* may vary according with the configured MODBUS data.

Data Size: the *Data Size* value sets the maximum amount of data that a MODBUS relation can access from the initial address. Thus, 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 set MODBUS data type, that is, when selected *Coil* or *Input Status*, the field *Data Size* must be a number multiple of 8. It is also important to take care so the maximum value is not greater than the addressable output memory size and the attributed values aren't the same already used during the application.

> **ATTENTION**
>
> When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

IEC Variable: in case the MODBUS data type is *Coil* or *Input Status* (bit), the IEC variables initial address will be in the format for example *%QX10.1*. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the IEC variables initial address will be in the format *%QW*. This field is limited by the memory size of the addressable output variables (%Q) from each CPU, which can be seen on the Memory section.

Read-only: when enabled, it only allows the communication master to read the variable data. It does not allow the writing. This option is valid for the writing functions only.

altus

Default: the default cannot be defined for the *IEC Variable* field as the creation of a protocol instance can be made at any moment within the application development, making the MasterTool IEC XE software allocate a value itself from the direct representation output variables range (%Q) still not used. The default cannot be defined for the *Data Size* field as it will vary according to selected MODBUS data type.

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

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

Table 96: 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.

In the previously defined relations, the maximum MODBUS data size can be 65536 (maximum value configured in the *Data Size* field). However, the request which arrives in the MODBUS Ethernet Server must address a subgroup of this mapping and this group must have, at most, the data size depending on the function code which is defined below:

- Read Coils (FC 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (FC 3): 125
- Read Input Registers (FC 4): 125
- Write Single Coil (FC 5): 1
- Write Single Holding register (FC 6): 1
- Force Multiple Coils (FC 15): 1968
- Write Holding Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Holding Registers (FC 23):
  - Read: 121
  - Write: 121

> **ATTENTION**
>
> Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of an Ethernet MODBUS Server 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.

5.5.7. OPC DA Server

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

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

When it comes to project development, to configure the communication and exchange information between the systems is extremely simple using OPC DA technology. Using other drivers, based on addresses, it's necessary to create tables to relate tags from the supervision system with variables from the programmable controller. When the data areas are changed during the project, it's necessary to remap the variables and create new tables with the relations between the information on the PLC with the Supervisory Control And Data Acquisition system (SCADA).
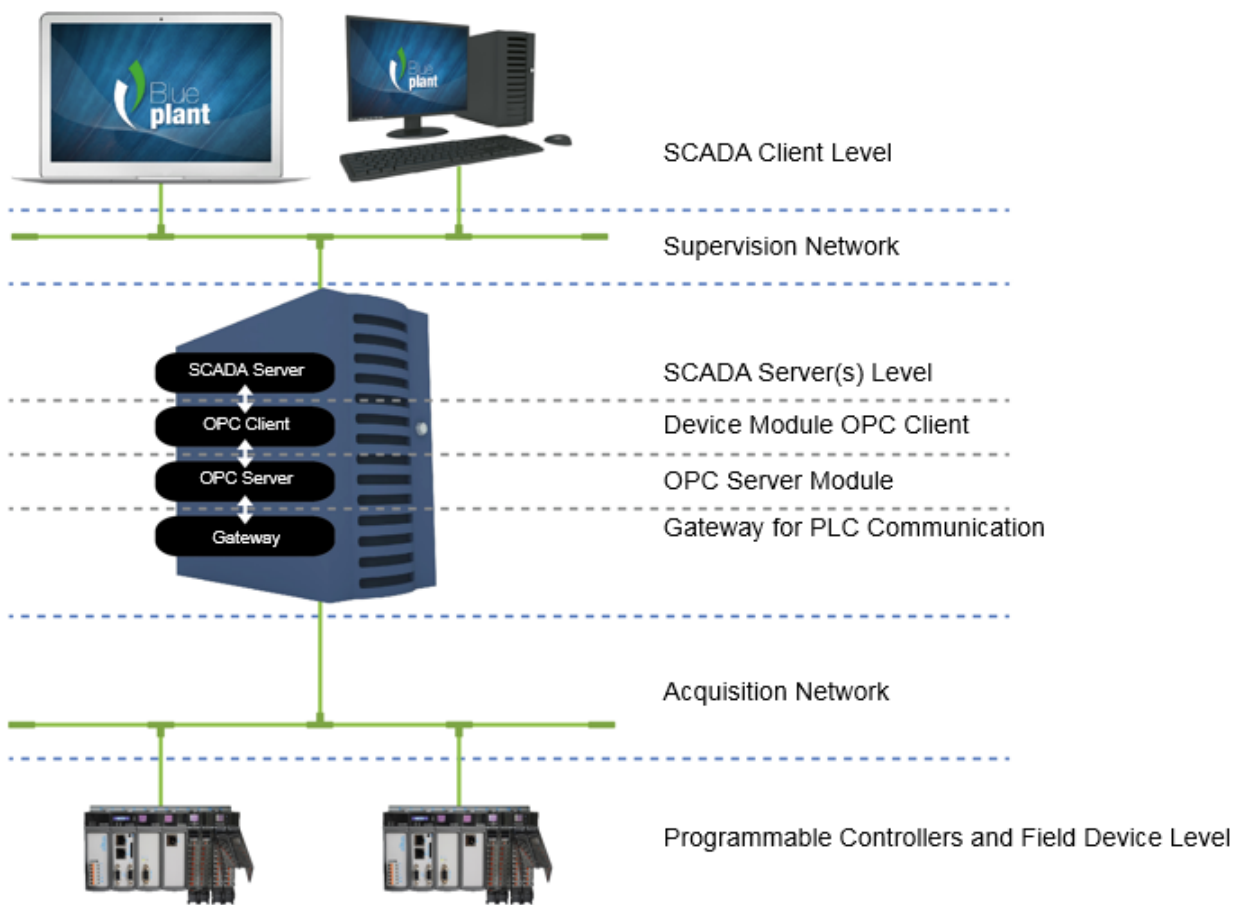
Figure 75: OPC DA Architecture

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.
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 97: 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 local Ethernet interface of the Nexto CPUs. The Ethernet expansion modules do not support this functionality.

5.5.7.1. Creating a Project for OPC DA Communication

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

altus

> **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.
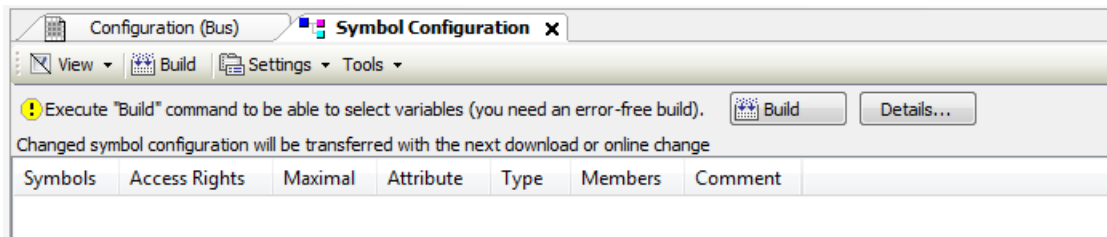
Figure 76: Symbol Configuration Object

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.

Field	Description
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.

<div align="center">Table 98: Symbol Configuration object screen fields description</div>

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 76 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.
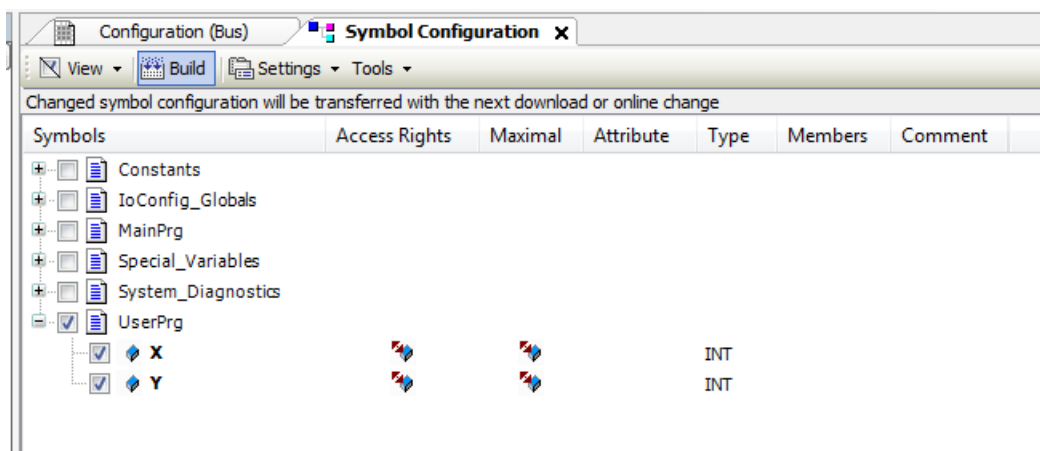
<div align="center">Figure 77: Selecting Variables on the Symbol Configuration</div>

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.

5.5.7.2. 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.
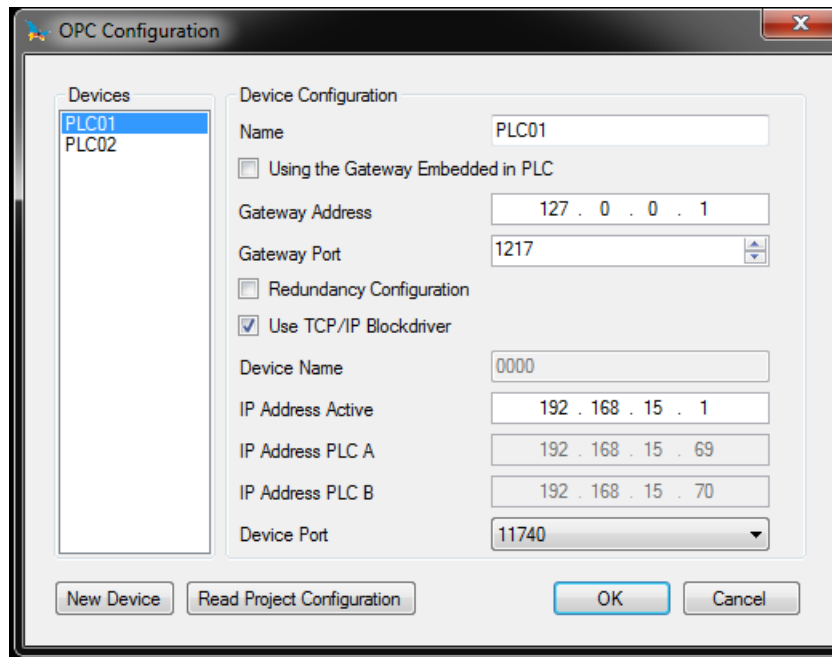
Figure 78: OPC DA Server Settings

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 99: Configuration Parameter of each PLC for the OPC DA Server

altus

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.

5.5.7.2.1. Importing a Project Configuration

Using the button *Read Project Configuration*, as shown in Figure 78, 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.

5.5.7.3. OPC DA Communication Status and Quality Variables

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

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

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

Table 100: 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 101: Description of the OPC Quality value

Table 101 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.

altus

5.5.7.4. Limits of Communication with OPC DA Server

The table below presents the OPC DA Server configuration limits.

Maximum number of variables communicating with a single PLC	-
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 102: 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 102.

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.

5.5.7.5. Accessing Data Through an OPC DA Client

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

> **ATTENTION**
>
> The same way that in MasterTool 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*.
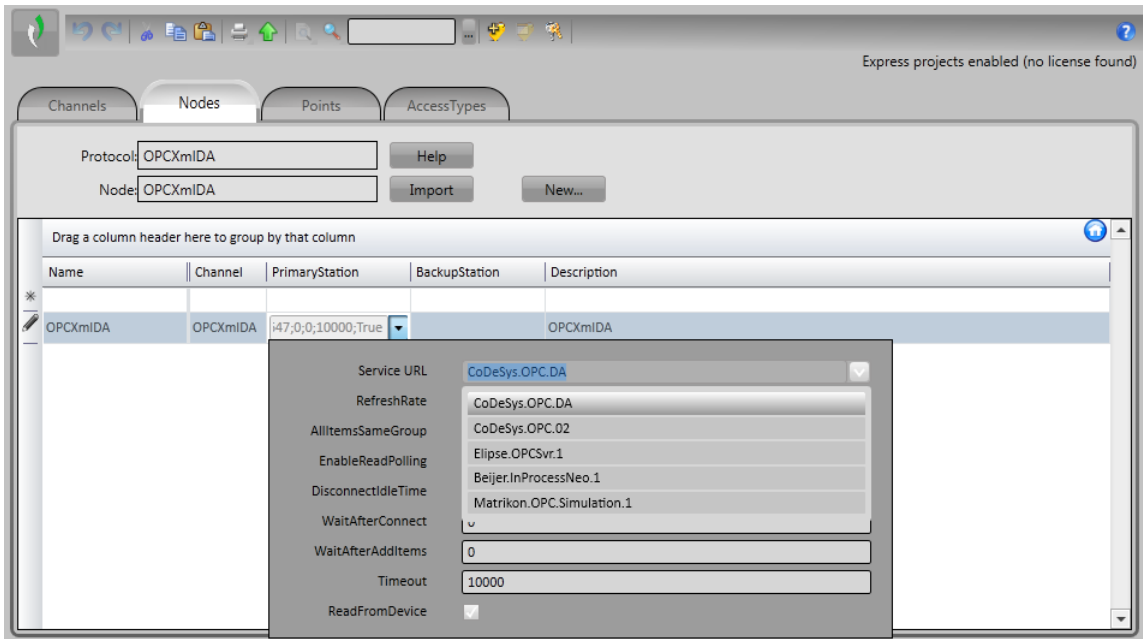
Figure 79: Selecting the OPC DA Server in the Client Configuration

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.
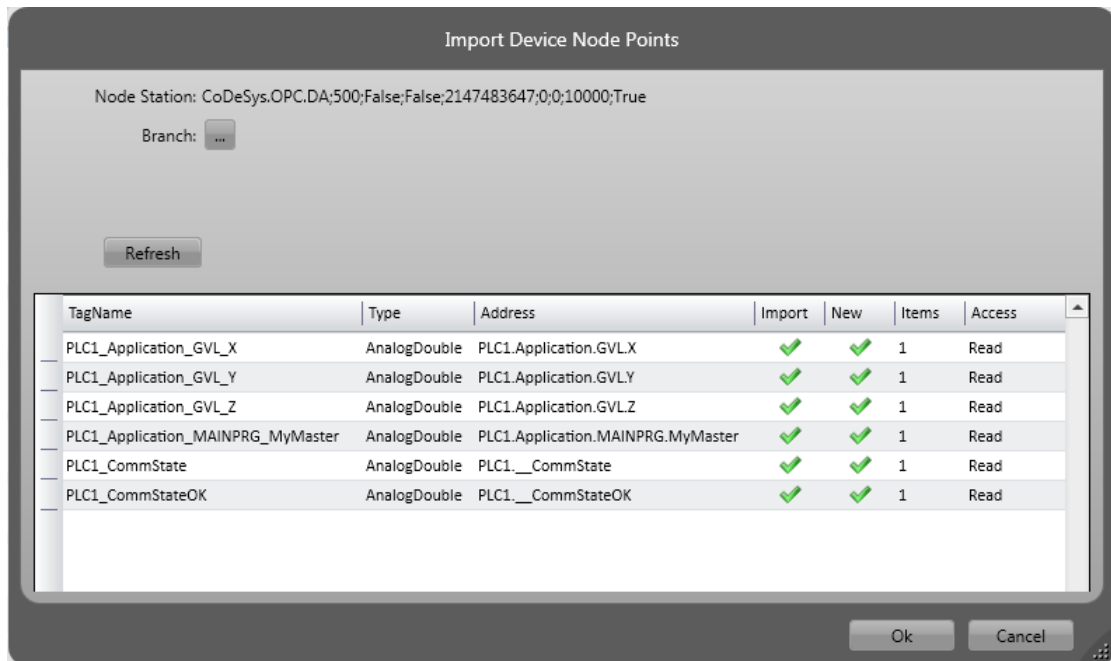
Figure 80: Symbols list consulted by the OPC Client

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.

5.5.8. OPC UA Server

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

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

When it comes to project development, configuring communication and exchanging information between systems is extremely simple using OPC UA technology. Using other address-based drivers, it is necessary to create tables to relate the supervision system tags and programmable controller variables. When data areas change during project development, it is necessary to redo the mappings and new tables with the relationships between the PLC information and the SCADA system.
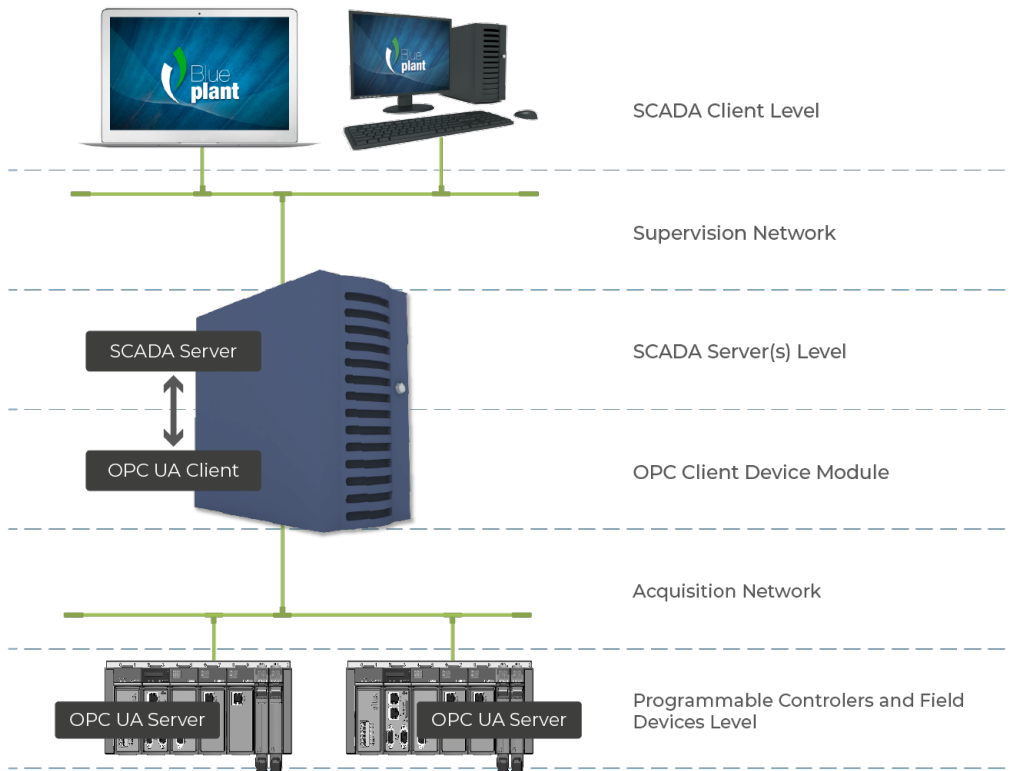
Figure 81: OPC UA Architecture

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 103: 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 local Ethernet interfaces of the Nexto CPUs. Ethernet expansion modules do not support this functionality.

5.5.8.1. Creating a Project for OPC UA Communication

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

altus
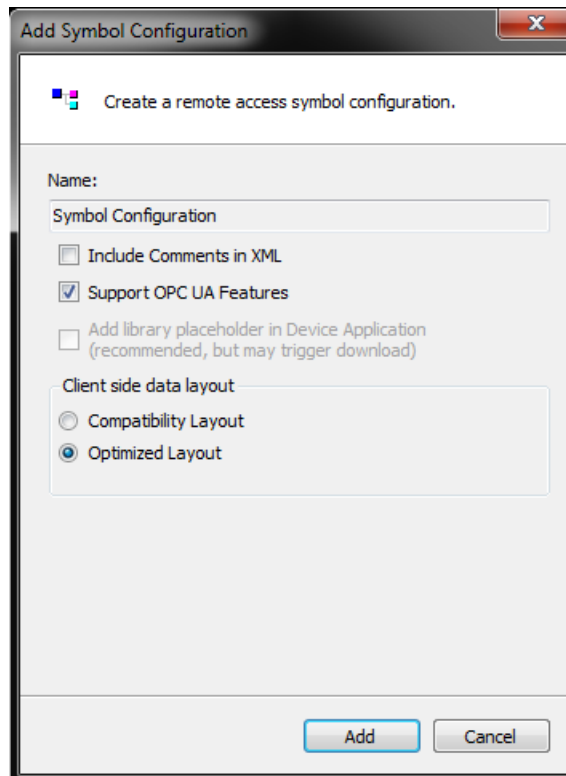
Figure 82: Symbol Configuration Object

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.
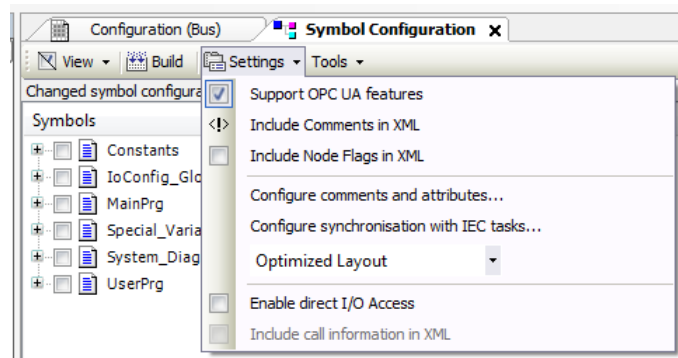
Figure 83: Enabling OPC UA in Object Symbol Configuration

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.

5.5.8.2. Types of Supported Variables

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

The following types of simple variables are supported:

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

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

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

5.5.8.3. Limit Connected Clients on the OPC UA Server

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

5.5.8.4. Limit of Communication Variables on the OPC UA Server

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

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

5.5.8.5. Encryption Settings

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

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

1. Define an active path for communication with the controller (no login required);
2. From the *View* menu, select *Security Screen*;
3. Click the *Devices* tab on the left side of this screen;
4. Click the icon ⟳ to perform a refresh;
5. Click on the *Device* icon, below which will open several certificates (*Own Certificates, Trusted Certificates, Untrusted Certificates, Quarantined Certificates*);
6. Click the icon to generate a certificate and select the following parameters:

   - *Key length* (bit): 3072
   - *Validity period* (days): 365 (can be modified if desired)

7. Wait while the certificate is calculated and transferred to the controller (this may take a few minutes);
8. Reboot the controller.
9. On the OPC UA client, perform the necessary procedures to connect to the OPC UA server and generate a certificate with the *Basic256Sha256* profile (see specific OPC UA client manual for details);

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

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

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

5.5.8.6. Main Communication Parameters Adjusted in an OPC UA Client

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

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

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

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

5.5.8.6.1. Endpoint URL

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

opc.tcp://192.168.17.2:4840

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

The TCP port should always be 4840.

5.5.8.6.2. Publishing Interval (ms) e Sampling Interval (ms)

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

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

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

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

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

5.5.8.6.3.   Lifetime Count e Keep-Alive Count

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

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

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

In situations where the variables of a *subscription* do not change, it could be a long time without the transmission of *PublishResponses* and consequently *PublishRequests* that succeed, causing an undesired deactivation of the *subscription*. To prevent this from happening, the *Keep-Alive Count* parameter was created. If there are no *subscription* data changes for a time equal to *Keep-Alive Count* multiplied by *Publishing Interval*, the server will send a small empty *Publish Response* packet indicating that no variable has changed. This empty *Publish Response* will authorize the client to immediately send the next *PublishRequest*.

The *Keep-Alive Count* value must be less than the *Lifetime Count* value to prevent unwanted deactivation of the *subscription*. It is suggested that *LifeTime Count* be at least 3 times larger than *Keep-Alive Count*.

5.5.8.6.4.   Queue Size e Discard Oldest

These parameters must be maintained with the following fixed values, which are usually the default values on the clients:

- Queue Size: 1
- Discard Oldest: enable

According to the OPC UA standard, it is possible to define these parameters for each variable. However, many clients allow you to define common values for all variables configured in a *subscription*.

Queue Size must be retained with value 1 because there is no event support in this implementation of the OPC UA server, so it is unnecessary to define a queue. Increasing the value of *Queue Size* may imply increase communication bandwidth and CPU processing, and this should be avoided.

Discard Oldest must be maintained with the *enable* value, so that the *Publish Response* package always reports the most recent change of value detected for each variable.

5.5.8.6.5.   Filter Type e Deadband Type

These parameters must be maintained with the following fixed values, which are usually the default values in the clients:

- *Filter Type: DataChangeFilter*
- *Deadband Type: none*

According to the OPC UA standard, it is possible to define these parameters for each variable. However, many clients allow you to define common values for all variables configured in a *subscription*.

The *Filter Type* parameter must be of *DataChangeFilter*, indicating that value changes in the variables should cause it to be transmitted in a *Publish Response* package.

Deadband Type should be kept in *"none"* because there is no implementation of *deadbands* for analog variables. In this way, any change of the analog variable, however minimal, causes its transmission in a *Publish Response* package.

To reduce processing power and Ethernet communication bandwidth, you can deploy *deadbands* on your own as follows:

- Do not include the analog variable in a *subscription*;
- Instead, include in a *subscription* an auxiliary variable linked to the analog variable;
- Copy the analog variable to the auxiliary variable only when the user-managed *deadband* is extrapolated.

5.5.8.6.6.   PublishingEnabled, MaxNotificationsPerPublish e Priority

It is suggested that the following parameters be maintained with the following values, which are usually the default values in the clients:

- *PublishingEnabled: true*
- *MaxNotificationsPerPublish: 0*
- *Priority: 0*

altus

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.

5.5.8.7. Accessing Data Through an OPC UA Client

After configuration of the OPC UA Server the data available in all PLCs can be accessed via a Client OPC UA. In the configuration of the OPC UA Client, the address of the correct OPC UA Server must be selected. In this case the address *opc.tcp://ip-address-of-device:4840*. The figure below shows the server selection in the SCADA BluePlant client software driver.

> **ATTENTION**
>
> 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*.
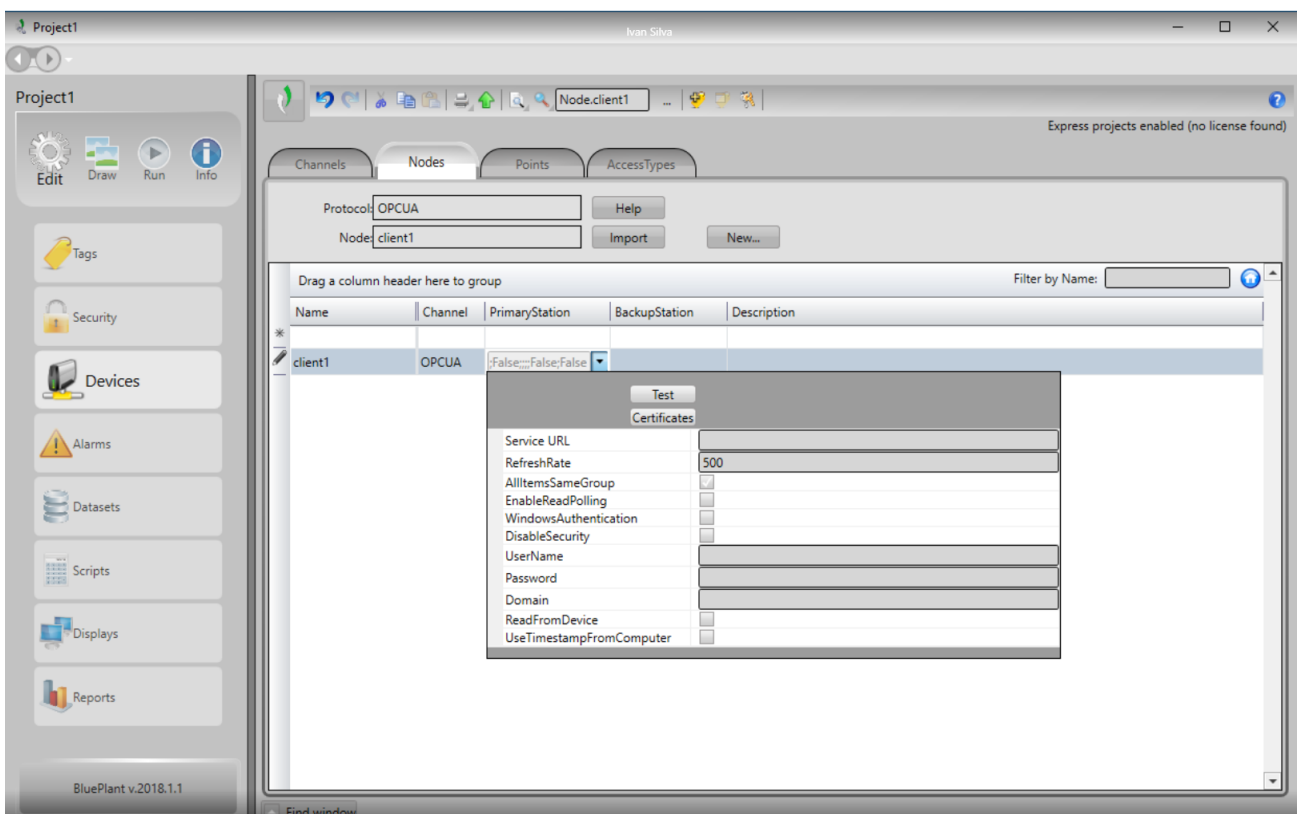
Figure 84: Selecting OPC UA Server in Client Configuration

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.
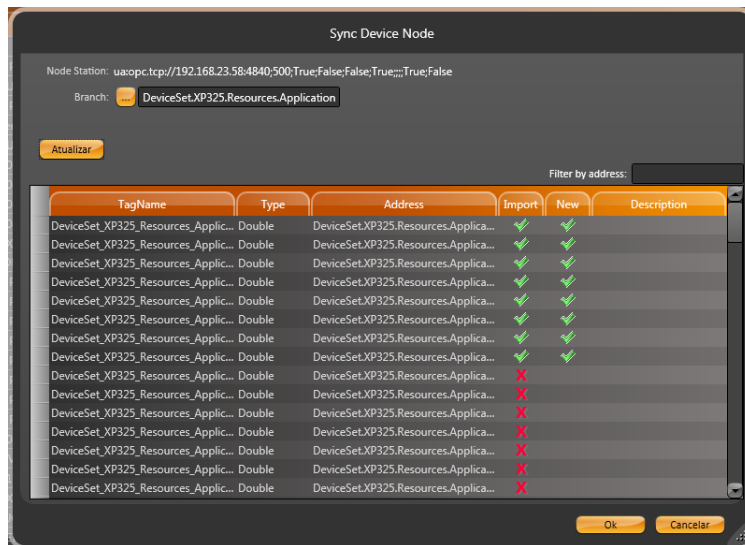
Figure 85: List of Symbols Browsed by OPC UA

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.

5.5.9. 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.

5.5.9.1. 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*.
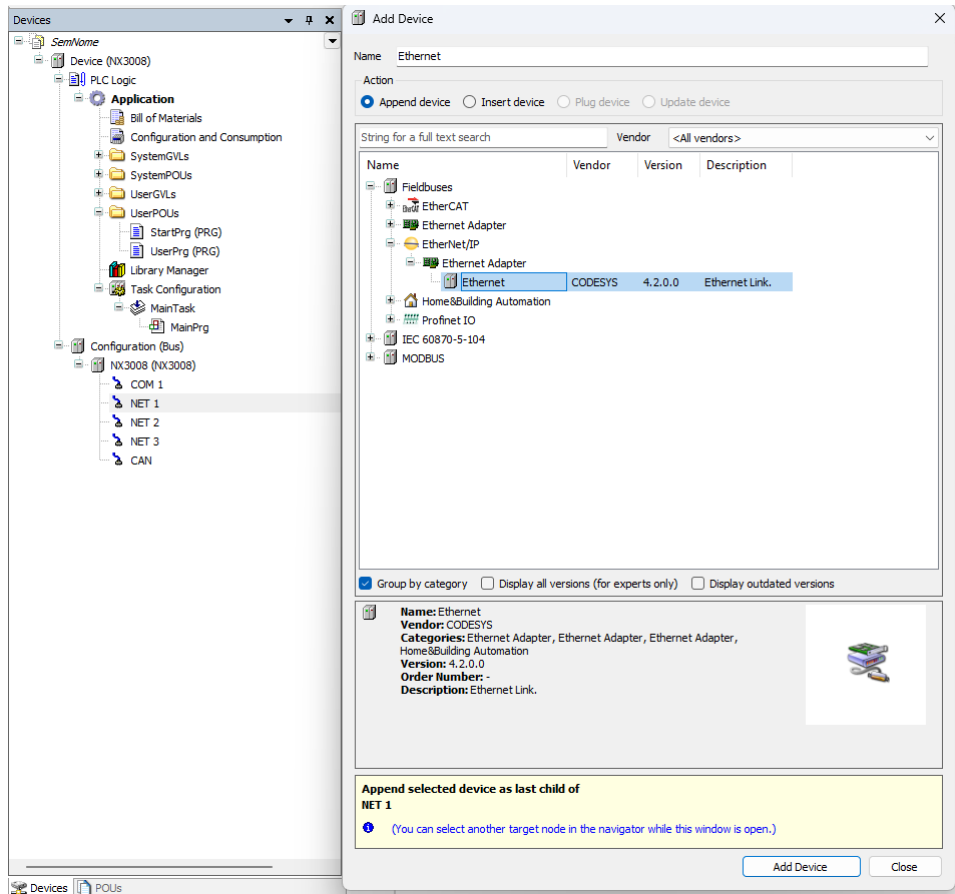
Figure 86: Adding an Ethernet Adapter
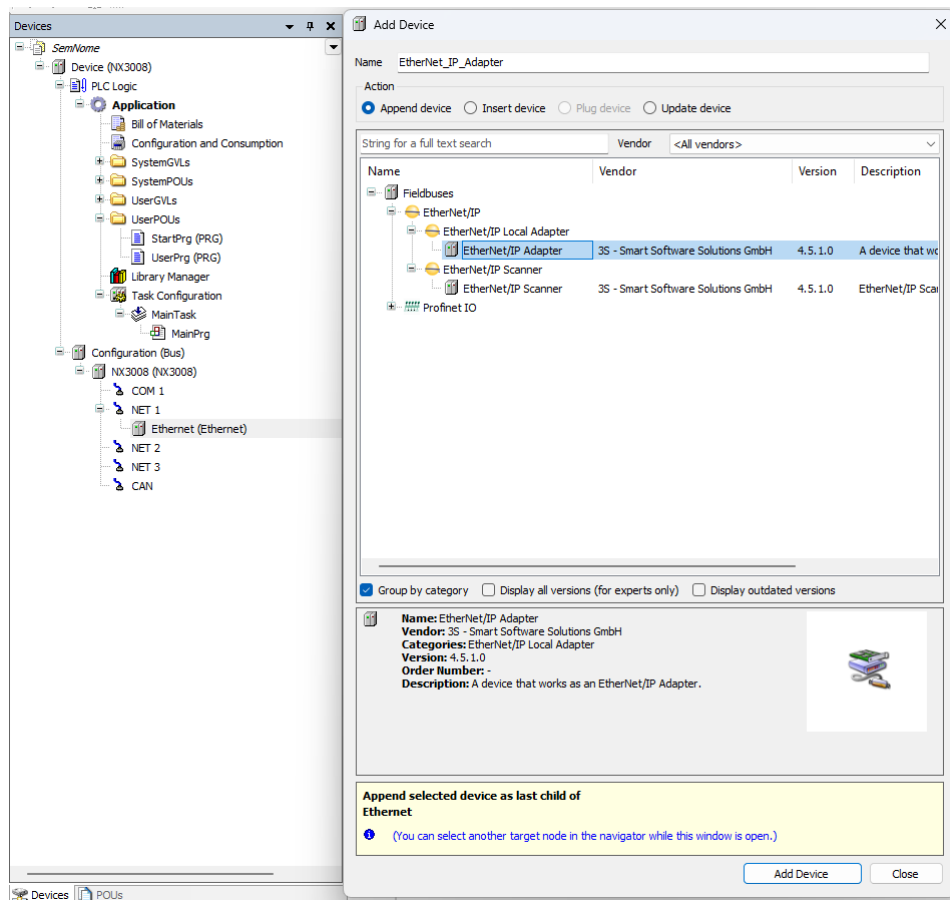
Figure 87: Adding an EtherNet/IP Adapter or Scanner

5.5.9.2. EtherNet/IP Scanner Configuration
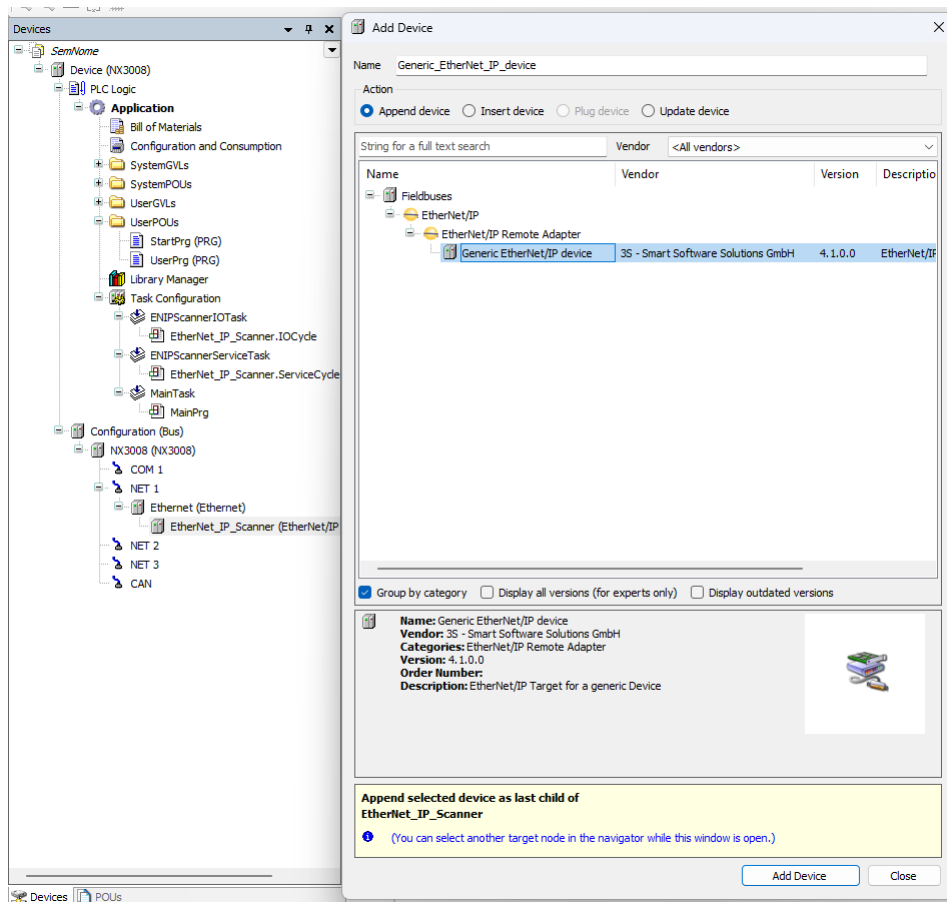
Figure 88: Adding an EtherNet/IP Adapter Under the Scanner

5.5.9.2.1. 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".
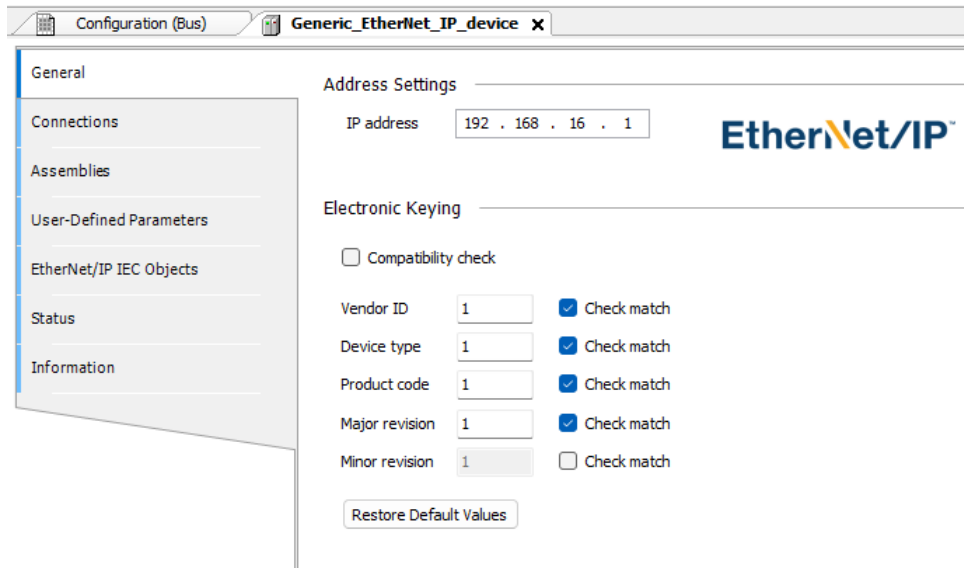
Figure 89: EtherNet/IP General Tab

5.5.9.2.2. 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.
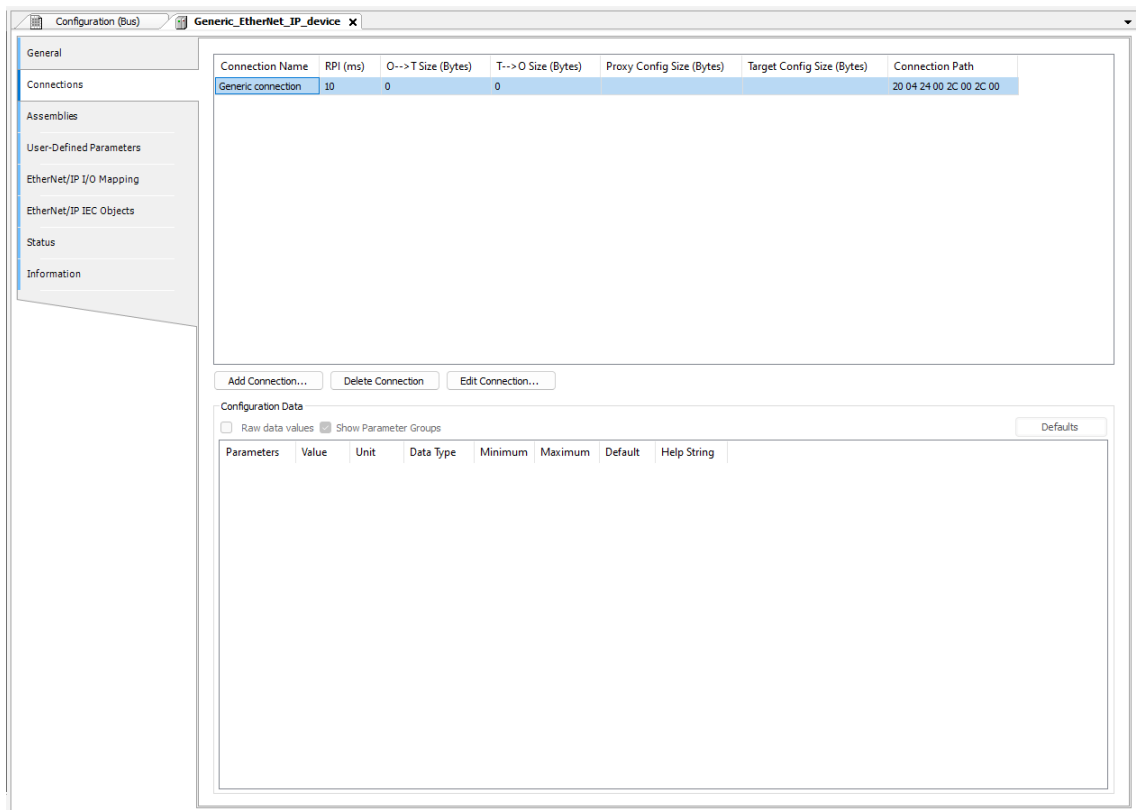
Figure 90: EtherNet/IP Connection Tab

altus

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 104: 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.
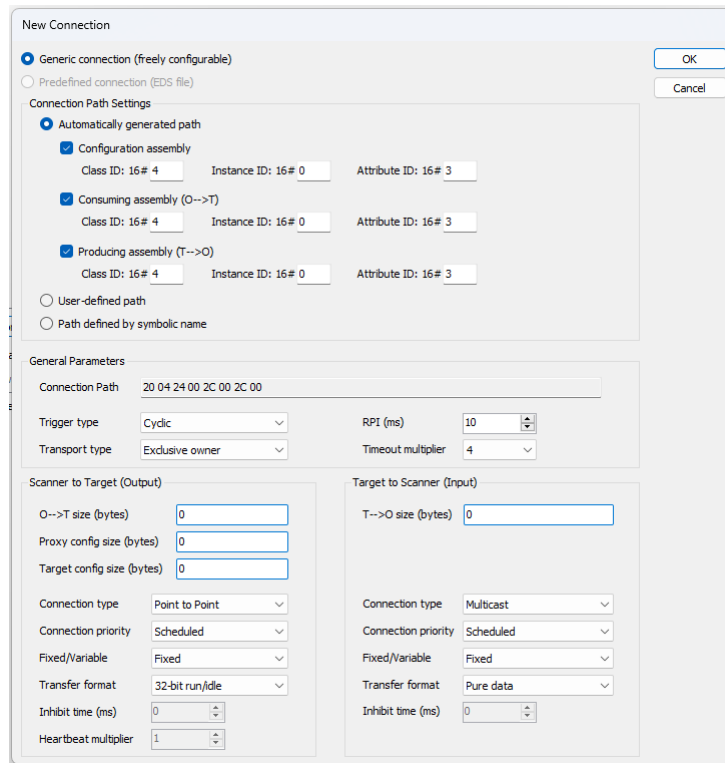
Figure 91: EtherNet/IP New Connection's Window

5.5.9.2.3. *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.
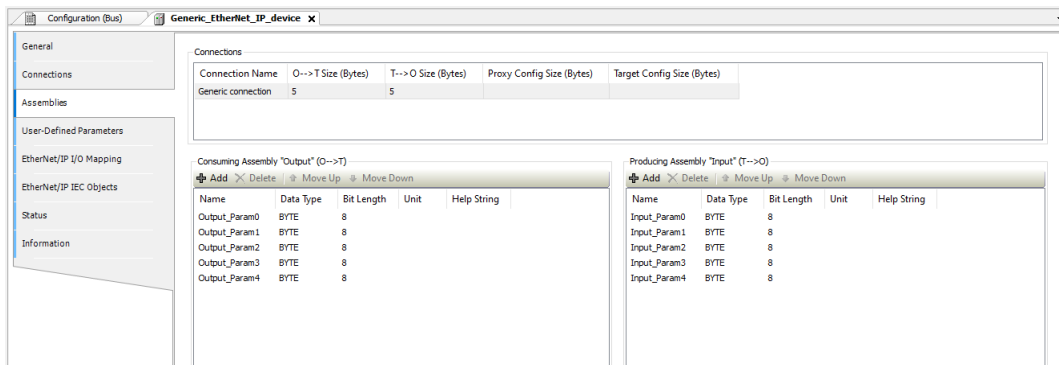
Figure 92: EtherNet/IP Assemblies

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 105: 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 106: EtherNet/IP "Add Input/Output" window

5.5.9.2.4. *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.

5.5.9.3. **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.

5.5.9.3.1. *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.

altus
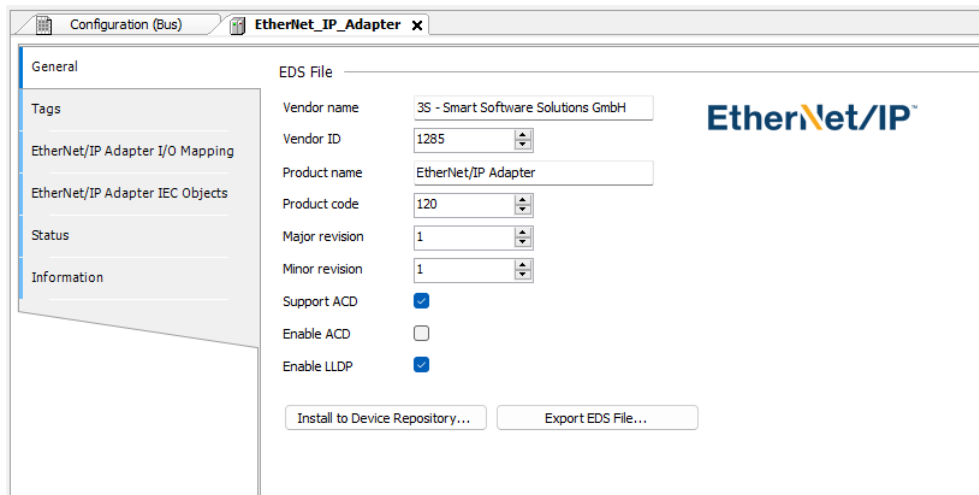
Figure 93: EtherNet/IP General Tab

5.5.9.3.2. *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.

5.5.9.4. EtherNet/IP Module Configuration
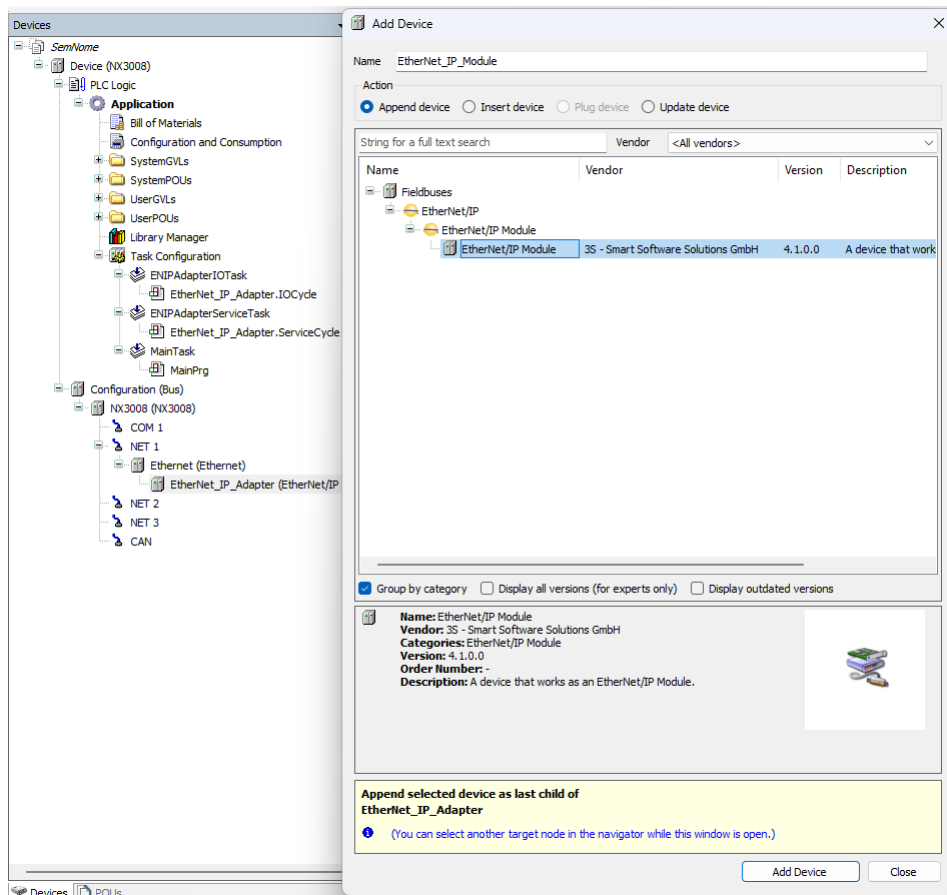
Figure 94: Adding an EtherNet/IP Module under the Adapter

5.5.9.4.1. *Assemblies*

The parameters of the module's *General* tab follow the same rules as described in the 105 and 106 tables.
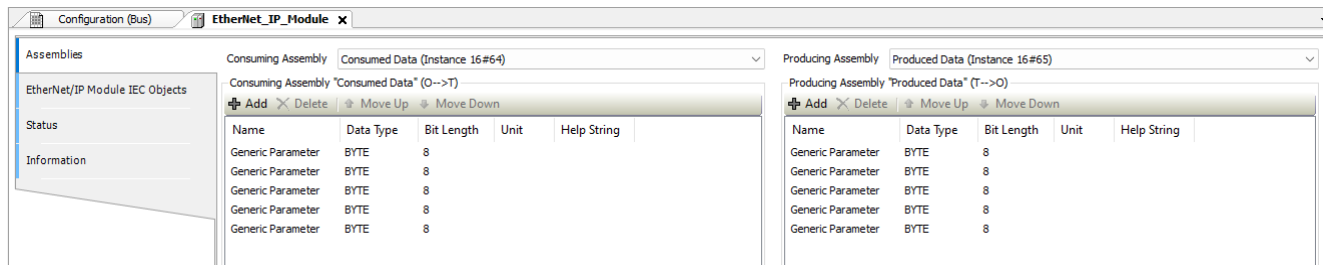
Figure 95: EtherNet/IP Module Assemblies tab

5.5.9.4.2. *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.

5.5.10. PROFINET Controller

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

5.6. Communication Performance

5.6.1. 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.

5.6.1.1. CPU's Local Interfaces

For a device MODBUS Ethernet Server, we can assert that the device is capable to answer a x number of requisitions per second. Or, in other words, the Server is able to transfer n bytes per second, depending on the size of each requisition. As smaller is the cycle time of the MODBUS Server task, higher is the impact of the number of connections in his answer rate. However, for cycle times smaller than 20 ms this impact is not linear and the table below must be viewed for information.

The table below exemplifies the number of requisitions that a MODBUS Server inserted in a local 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 107: Communication Rate of a MODBUS Server at Local 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:

$N = C \times (1 / T)$

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;

$N = 1 \times (1 / (0,05))$

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.

5.6.2. OPC DA Server

Communication performance with OPC DA Server was tested by creating POUs with 1,000 INT variables each. All scenarios were tested with *Single* profile and MainTask Interval at 100 ms. The communication was enabled by the *attribute 'symbol' := 'readwrite'*, to make the data available to the OPC DA Server. The measurements were made while MasterTool was disconnected from the CPU, and MainTask duration was made to last 5%, 50% and 80% of the configured Interval, as seen in table below.

At the OPC Client's side, a SCADA system driver was used. Configured update time was 50 ms. Performance results in these conditions are described in table below.

Total quantity of variables in the PLC's project	Variable update time at OPC DA Client		
	5% of CPU Busy	**50% of CPU Busy**	**80% of CPU Busy**
1000	600 ms	800 ms	1400 ms
2000	800 ms	900 ms	2800 ms
5000	1000 ms	2000 ms	6500 ms
10000	2000 ms	4000 ms	13700 ms
15000	3200 ms	6400 ms	20000 ms
20000	4000 ms	8100 ms	25000 ms

Table 108: Communication Rate of an OPC DA Server

5.6.3. 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.

5.7. System Performance

In cases where the application has only one MainTask user task responsible for the execution of a single Program type programming unit called MainPrg (as in Single Profile), the PLC consumes a certain amount of time for the task to be processed. At that time we call it as *Execution Time*.

In an application the average application *Execution Time* can be known using the MasterTool IEC XE in the *Device* item of its *Devices Tree* as follows:

PLC Logic-> Application-> Task Configuration in the *Monitor* tab, *Average Cycle Time* column.

The user must pay attention to the *Cycle Time* so that it does not exceed 80% of the interval set in the MainTask user task. For example, in an application where the interval is 100 ms, an appropriate *Cycle Time* is up to 80 ms. This is due to the fact that the CPU needs time to perform other tasks such as communication processing, processing of the display and memory card, and these tasks take place within the range (the remaining 20% of *Cycle Time*).

ATTENTION

For very high cycle times (typically higher than 300 ms), even that the value of 80% is respected, it may occur a reduction in the display response time and of the diagnostics key. In case the 80 percentage is not respected and the running time of the user task is closer or exceeds the interval set for the MainTask, the screen and the diagnosis button cannot respond once its priority in the system running is lower than the user tasks. In case an application with errors is loaded in the CPU, it may be necessary to restart it without loading this application as shown in the System Log section.

ATTENTION

The CPU's system logs of the Nexto Series, starting from firmware version 1.4.0.33 now reloaded in case of a CPU reset or a reboot of the *Runtime System*, that is, you can view the older logs when one of these conditions occurs.

5.7.1. I/O Scan Time

For a project that uses digital I/O modules, being them inserted into the bus and declared in the project, the MainTask time will increase according to the number of modules. The table below illustrates the average time that is added to the MainTask:

Declared Modules in the Bus	Added Time in the MainTask Cycle Time (μs)
5	300
10	700
20	1000

Table 109: I/O Scanning Time

In projects that use remote I/Os, for example, using the NX5001 PROFIBUS-DP Master module, the manual of the respective module has to be consulted for information about performance and influences of the module in the execution of user tasks.

5.7.2. Memory Card

Data transfers involving the memory card is performed by the CPU in the background, as this gives priority to the execution of user application and communication processing. Thus, the transfer of files to the card may suffer an additional significant time, depending on the Cycle Time of the user application.

The time required to read/write files on the card will be directly affected by the Cycle Time of the user application since this application has priority in execution.

Further information about the use of the memory card see Memory Card section.

5.8. 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:
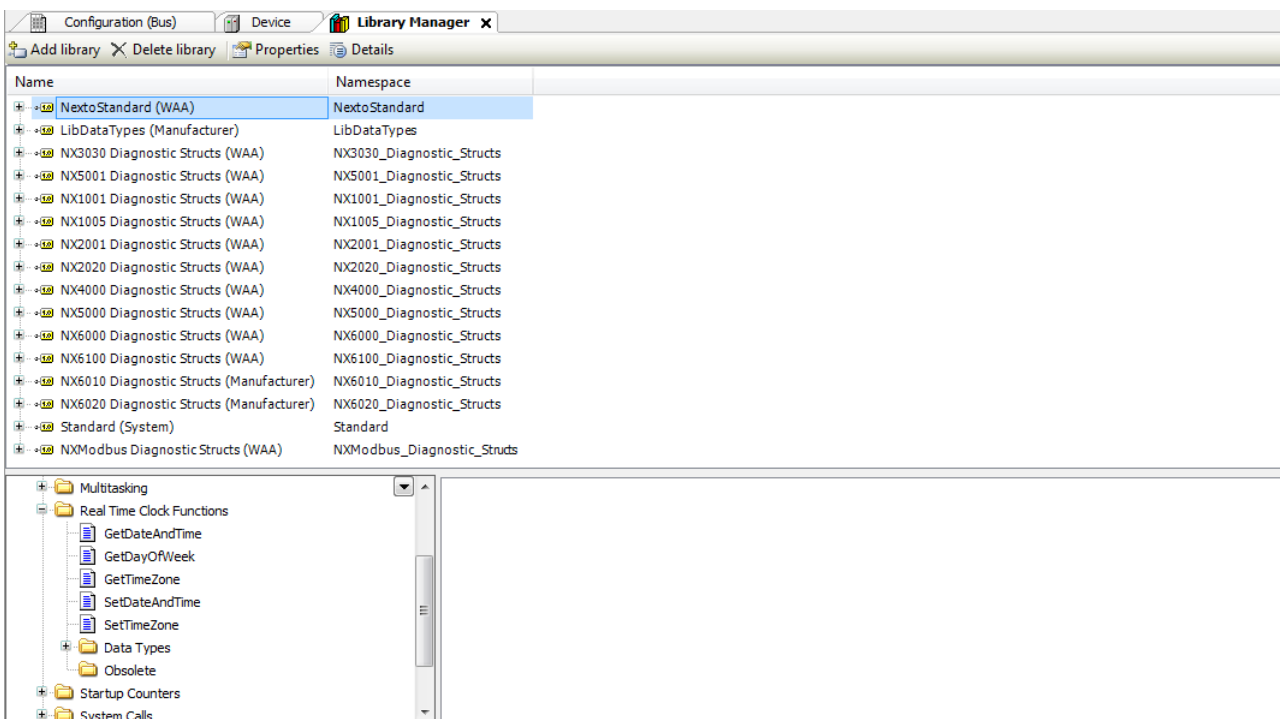
Figure 96: Clock Reading and Writing Blocks

> **ATTENTION**
>
> Function blocks of RTC Reading and Writing, previously available in 2.00 MasterTool IEC XE or older become obsolete from 2.00 or newer, the following blocks are no longer used: *NextoGetDateAndTime*, *NextoGetDateAndTimeMs*, *NextoGetTimeZone*, *NextoSetDateAnd-Time*, *NextoSetDateAndTimeMs* and *NextoSetTimeZone*.

5.8.1. 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.

5.8.1.1. Function Blocks for RTC Reading

The clock reading can be made through the following functions:

5.8.1.1.1. GetDateAndTime
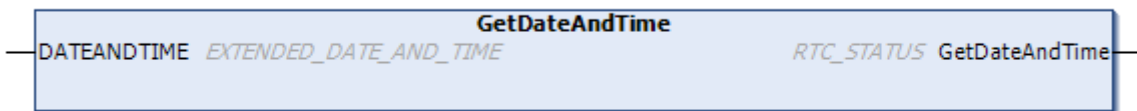
Figure 97: Date and Hour Reading

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 119.

Table 110: Input Parameters of GetDateAndTime

Output Parameters	Type	Description
GETDATEANDTIME	RTC_STATUS	Returns the function error state, see Table 121.

Table 111: 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
```

5.8.1.1.2. *GetTimeZone*

The following function reads the Time Zone configuration, this function is directly related with time in Time Zone at SNTP synchronism service:
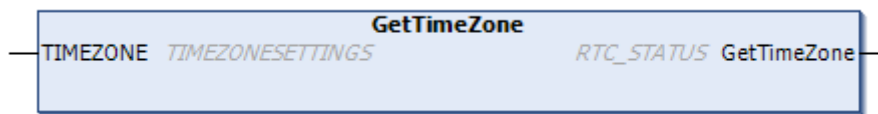
Figure 98: Configuration Reading of Time Zone

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

Table 112: Input Parameters of GetTimeZone

Output Parameters	Type	Description
GetTimeZone	RTC_STATUS	Returns the function error state, see Table 121.

Table 113: 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
```

5.8.1.1.3. *GetDayOfWeek*

GetDayOfWeek function is used to read the day of the week.
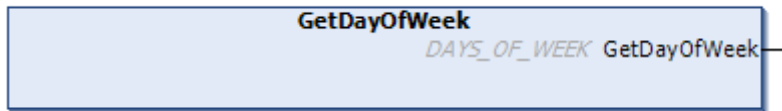
Figure 99: Day of Week Reading

Output Parameters	Type	Description
GetDayOfWeek	DAYS_OF_WEEK	Returns the day of the week. See Section 120.

Table 114: 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();
```

5.8.1.2. RTC Writing Functions

The clock settings are made through function and function blocks as follows:

5.8.1.2.1. *SetDateAndTime*

SetDateAndTime function is used to write the settings on the clock. Typically the precision is on the order of hundreds of milliseconds.
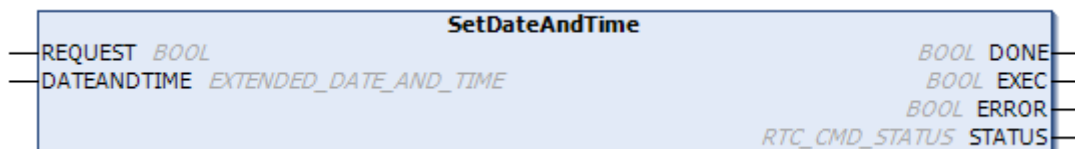
Figure 100: Set Date And Time

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 119.

Table 115: 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_CMD_STATUS	Returns the error occurred during the configuration. See Table 121.

Table 116: 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
```

altus

> **ATTENTION**
>
> If you try to write time values outside the range of the RTC, the values are converted to valid values, provided they do not exceed the valid range of 01/01/2000 to 12/31/2035. For example, if the user attempts to write the value 2000 ms, it will be converted to 2 seconds, write the value 100 seconds, it will be converted to 1 min and 40 seconds. If the type value of 30 hours, it is converted to 1 day and 6 hours, and so on.

5.8.1.2.2. *SetTimeZone*

The following function block makes the writing of the time zone settings:
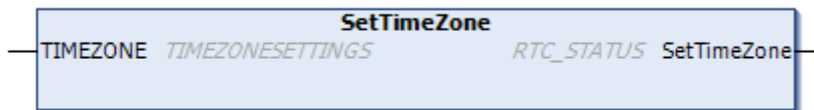
Figure 101: Writing of the Time zone Settings

Input parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	Structure with time zone to be configured. See Table 122.

Table 117: SetTimeZone Input Parameters

Output parameters	Type	Description
SetTimeZone	RTC_STATUS	Returns the error occurred during the reading/setting. See Table 121.

Table 118: 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 perform the clock should be used time and date values within the following valid range: 00:00:00 hours of 01/01/2000 to 12/31/2035 23:59:59 hours, otherwise , is reported an error through the *STATUS* output parameter. For details of the *STATUS* output parameter, see the section RTC_STATUS.

5.8.2. RTC Data Structures

The reading and setting function blocks of the Nexto Series CPUs RTC use the following data structures in its configuration:

5.8.2.1. 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 119: EXTENDED_DATE_AND_TIME

5.8.2.2. 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 120: DAYS_OF_WEEK Structure

altus

5.8.2.3. 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.
	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 121: RTC_STATUS

5.8.2.4. 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 122: 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.

5.9. 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.

altus
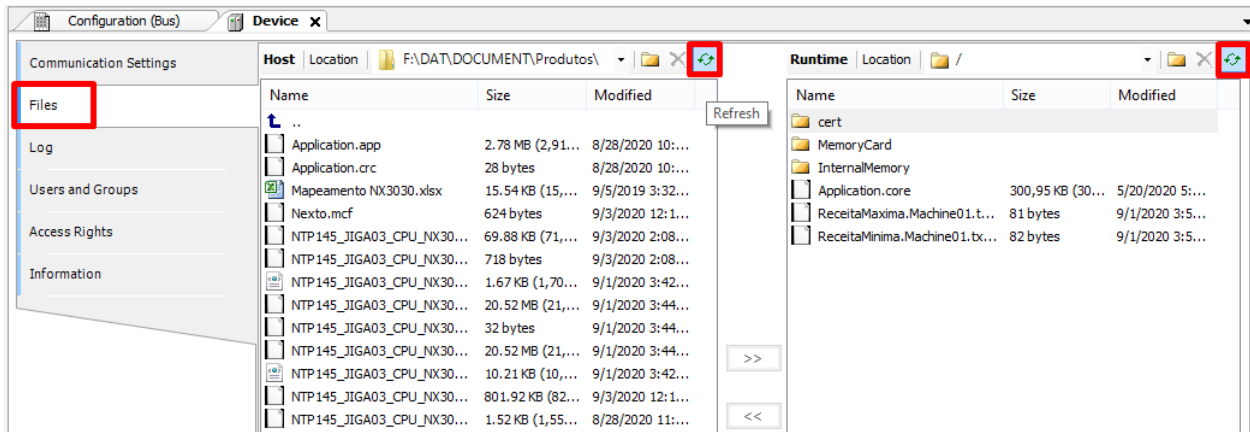
Figure 102: User Files Access

After updating the CPU column of files, the root directory of files stored in the CPU will be shown. Then it will be possible to select the folder where the files will be transferred to. The *"InternalMemory"* folder is a default folder to be used to store files in the CPU's internal memory, since it is not possible to transfer files to the root directory. If necessary, the user can create other folders in the root directory or subfolders inside the *"InternalMemory"* folder.

The *"MemoryCard"* folder is the directory where the memory card is mounted, if it is inserted into the CPU. Files which are transferred to the *"MemoryCard"* are being transferred directly into the memory card. As new features are being added to the product, some folders may appear and which should be ignored by the user.

> **ATTENTION**
>
> In the case where the memory card is inserted after the CPU startup, an username and password will be requested to perform the MasterTool IEC XE access and/or file transfers to the memory card or vice versa. The standard user with privileges to access the CPU is *"Owner"* and the default password for that user is *"Owner"*.

In order to perform a file transfer from the microcomputer to the CPU just select the desired file in the left column and press the "»" key located in the center of the screen, as shown in figure below. The download time will vary depending on file size and cycle time (execution) of the current application of the CPU and may take several minutes.

The user does not need to be in *Run* Mode or connected to the CPU to perform the transfers, since it has the ability to connect automatically when the user performs the transfer.
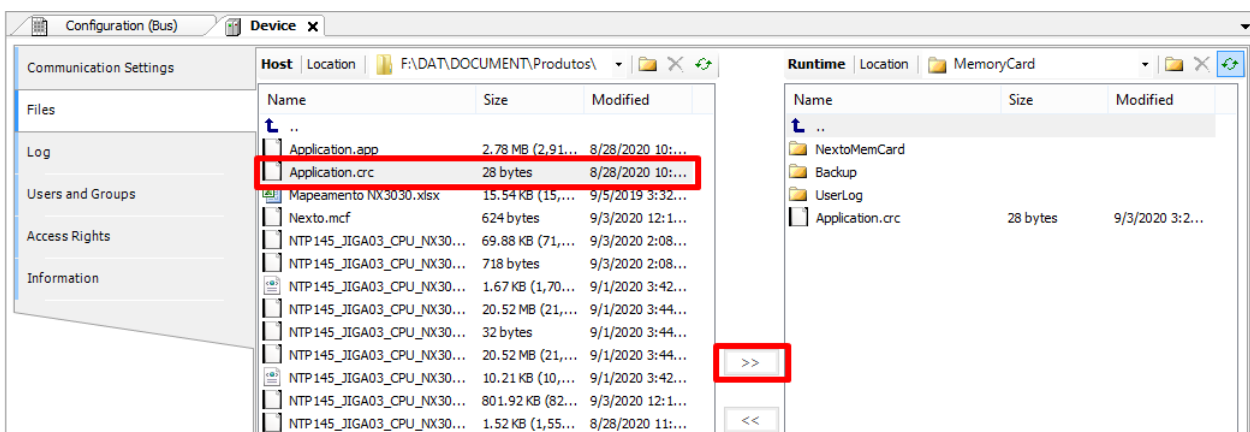
Figure 103: Files Transference

> **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.
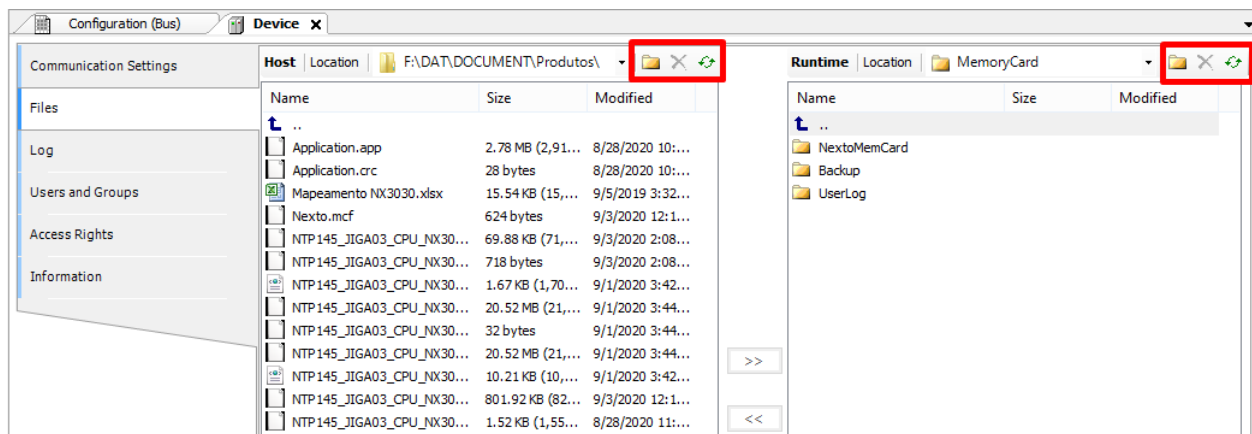
Figure 104: Utilization Options

> **ATTENTION**
>
> For a CPU in Stop Mode or with no application, the transfer rate to the internal memory is approximately 150 Kbytes/s.

5.10. Memory Card

Among other memories, Nexto Series CPUs allow the user the utilization of a memory card. It is defined according the features described in Memory Card Interface section which stores, among other files, the project and application in the CPU internal memory.

When the card is inserted in the CPU and it presents a file type different from FAT32, it automatically identifies those files and questions the user if he wants to format the files. In negative case the user cannot use the card, as it is not mounted. A message informing the format is not recognized is presented and the card presence is not displayed either. If the user decides to format the files, the CPU takes a few minutes to execute the operation, depending on the cycle time (execution) of the application which is running in the CPU. Once the memory card is mounted, the CPU will read its general information, leaving access to the memory card slower in the first few minutes. This procedure is done only when the card is inserted or in case of the CPU reset.

> **ATTENTION**
>
> It is recommended to format the memory card directly in the Nexto CPU in order to avoid possible use problems, mounting time increase or even the incorrect functioning.
> It is not recommended to remove the memory card or de-energize the CPU during the formatting or during the files transfer as it can cause the loss of data as well as irreversible damages.

5.10.1. Project Preparation

To use the functionality, during the project configuration, in the MasterTool IEC XE software, the user must enable the option to copy the CPU project to the memory card and/or memory card to the CPU and to configure passwords. These passwords will be requested by the CPU when executing the respective transfer. For information about the table, see section Project Parameters.

> **ATTENTION**
>
> If the CPU has no application, the "*Memory Card*" Menu will be available to allow the transfer of the project from memory card to the CPU without requiring any kind of previous preparation of the CPU.

To use the feature you must perform the following steps.

Navigate to the *Online* menu and execute the command *Create Boot Application*, remembering that you cannot be logged into the CPU to perform this procedure. After you run this command, two files are created in the folder where the project is saved. One with the extension "*app*" and one with the extension "*crc*".

After generating the files in the previous item, you must navigate to the CPU *General Parameters* settings and click the *Memory Card...* button. A new screen will open as shown in figure below. In this screen you must enable the desired transfer operation(s) and, if necessary, set the password(s) with numeric characters only. The use of password is not required.

To complete the setup operation you must click the *Find File...* button and then locate the file with "*.crc*" extension generated in the previous step.
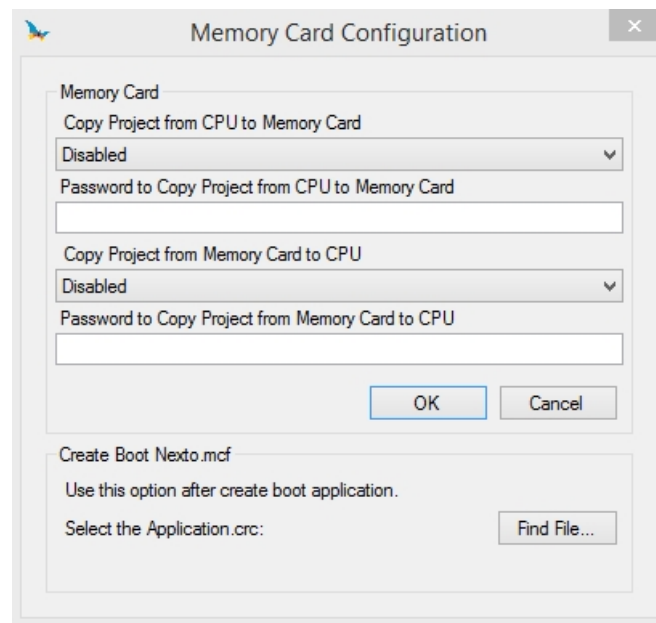
Figure 105: Memory Card Settings

Following these steps, MasterTool IEC XE will send all files needed to perform the send and receive operations of projects via memory card. If the card is mounted, the password will be written to it. Otherwise, the password set in MasterTool will be requested if the user tries to transfer the CPU project to the card.

5.10.2. Project Transfer

To transfer the project from CPU to the memory card or vice versa, the user, in addition to enabling in MasterTool IEC XE software to use the functionality, will have to access the menu *Memory Card* in the CPU, using the diagnostics key, and select the desired transfer option.

> **ATTENTION**
>
> The transfer of the project to the memory card should only be done using the CPU diagnostics key.

Afterwards, you will be prompted for the password if the user has set during application setup. Then with a short press on the diagnostics key the digits are incremented and with a long press are confirmed. In the confirmed sixth digit, the CPU will consist of the password and start the process.

After transferring the memory card to the CPU, if there is a RUN application it will be kept in STOP for safety reasons. To put the CPU in RUN, it must be rebooted.

When the passwords of both the application that is in the CPU and the application that is on the memory card are the same, it is not required to enter the passwords in the CPU menu to perform the application transfers. For more information on using the diagnostics key, see section One Touch Diag.

To remove the memory card, simply press and hold the *MS* key and wait until the memory card icon disappears from the graphic display status screen.

> **ATTENTION**
>
> If the memory card is removed without have been unmounted through CPU's menu, during a file transference, this process can cause the loss of card data as well as corrupt the files in it. This process may cause the need of another card formatting when it'll be inserted on the CPU again.

> **ATTENTION**
>
> If there is any file at memory card root named "*NextoMemCard*" or "*Backup*", it will be deleted to create the system folders with the same name, used by the CPU to store the project application and the project archive. Folders with these names will not be overwritten.

5.10.3. MasterTool Access

The memory card access is connected to the same user memory screen in the MasterTool IEC XE software, being it mounted in the folder called *MemoryCard*. *NextoMemCard* and *Backup* folders are created into the memory card every time the latter is inserted into the CPU. In case these folders already exist, the system will recognize them and will not overwrite the folders.

In the *NextoMemCard* on the memory card, you will find the application files, in this window you still have the option to save your project in a preferred directory (if you have sent the source code). In MasterTool in the option "*File / Project Archive / Extract Archive...*" you can open in MasterTool the saved application, which is located in the directory previously chosen.

The *Backup* folder is not used by the user.
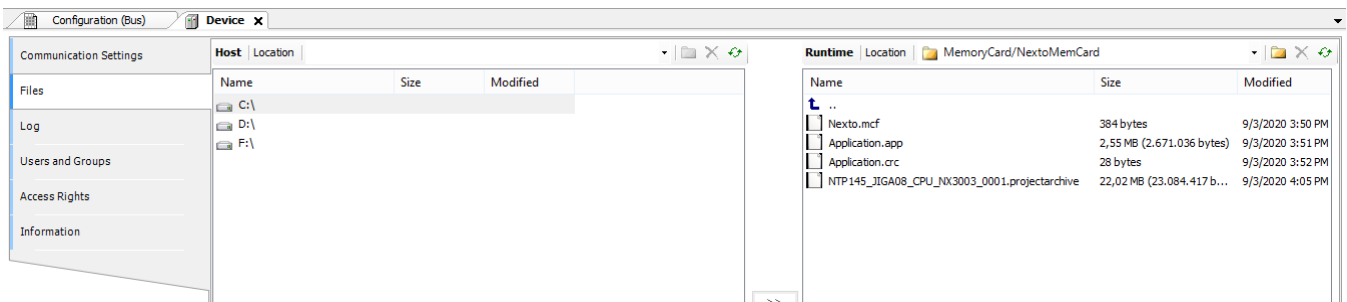
Figure 106: Directory with Memory Card Inserted with Project

> **ATTENTION**
>
> The files transference time depends on the interval time difference minus the average execution time of the task (s) in execution (available time until the next task cycle), it means, the bigger this difference for each task in an application, the faster will be the transference of a data from the memory card to a CPU/MasterTool IEC XE or vice-versa.
> Transferring files to the memory card will be slower than the transfer to the internal memory of the CPU. For a CPU in Stop Mode or with no application, the transfer rate is close to 100 Kbytes/s.

5.11.   CPU's Informative and Configuration Menu

The access to the *Informative Menu*, the Nexto CPU *Configuration* and the detailed diagnostics, are available through levels and to access the menu information, change level and modify any configuration, a long touch is required on the diagnostic button and to navigate through the items on the same level, a short touch on the diagnostic button is required. See One Touch Diag section to verify the functioning and the difference between the diagnostics button touch types.

The table below shows the menu levels and each screen type available in the CPUs, if they are informative, configurable or to return a level.

Level 1	Level 2	Level 3	Type
HARDWARE	TEMPERATURE	-	Informative
	CONTRAST	CONTRAST LEVEL	Configurable
	DATE AND TIME	-	Informative
	BACK	-	Return level
LANGUAGES	ENGLISH	>ENGLISH	Configurable
	PORTUGUES	>PORTUGUES	Configurable
	ESPANOL	>ESPANOL	Configurable
	BACK	-	Return level
NETWORK	NET 1 IP ADDR.		Informative
	NET 1 MASK		Informative
	BACK		Return level
SOFTWARE	FIRMWARE		Informative
	BOOTLOADER	-	Informative
	AUX. PROC.		Informative
	BACK		Return level
MEM. CARD	CPU > MEMCARD	CPU PASSWORD	Configurable
	MEMCARD > CPU	MC PASSWORD	Configurable
	FORMAT	CONFIRM ?	Configurable
	BACK	-	Return level
BACK	-	-	Return level

Table 123: CPU Menu Levels

Notes:

Memory Card: The memory card is only available in the menu, if it is connected in the Nexto CPU.

Password: The memory card data access password is only necessary in case it is configured in the MasterTool IEC XE software. You cannot edit the password via menu.

As shown on Table 123, between the available options to visualize and modify are the main data necessary to user, as:

- Information about the hardware resources:

  - TEMPERATURE – CPU Internal temperature (Ex.: 36 C 97 F)
  - CONTRAST – Contrast setting of the CPU frontal display
  - DATE AND TIME – Date and time set in the CPU (Ex.: 2001.01.31 00:00)

altus

- ■ Changing the menu language on the CPU:
  - PORTUGUES – Changes the language to Portuguese
  - ENGLISH – Changes the language to English
  - ESPANOL – Changes the language to Spanish
- ■ Visualization of information about the network set in the device:
  - NET 1 IP ADDR. – Address (Ex.: 192.168.0.1)
  - NET 1 MASK – Subnet mask (Ex.: 255.255.255.0)
- ■ Information about the software versions:
  - FIRMWARE – CPU software version (Ex.: 1.0.0.0)
  - BOOTLOADER – CPU bootloader version (Ex.: 1.0.0.0)
  - AUX. PROC. – CPU auxiliary processor version (Ex.: 1.0.0.0)
- ■ Access to the Memory Card data:
  - MEMCARD > CPU – Transference of the memory card project to the CPU
  - CPU > MEMCARD – Transference of the CPU project to the memory card
  - FORMAT – Formats the card to the FAT32 files system

The figure below describes an example of how to operate the Nexto CPUs menu through the contrast adjust menu procedure from the *Status* screen. Besides to make the configuration easy, it is possible to identify all screen levels and the touch type to navigate through them, and to modify other parameters as *Language* and the *Memory Card*, using the same access logic. The short touch shows the contrast is being incremented (clearer) and in the next touch after its maximum value, it returns to the minimum value (less clear). The long touch shows the confirmation of the desired contrast and its return to the previous level.
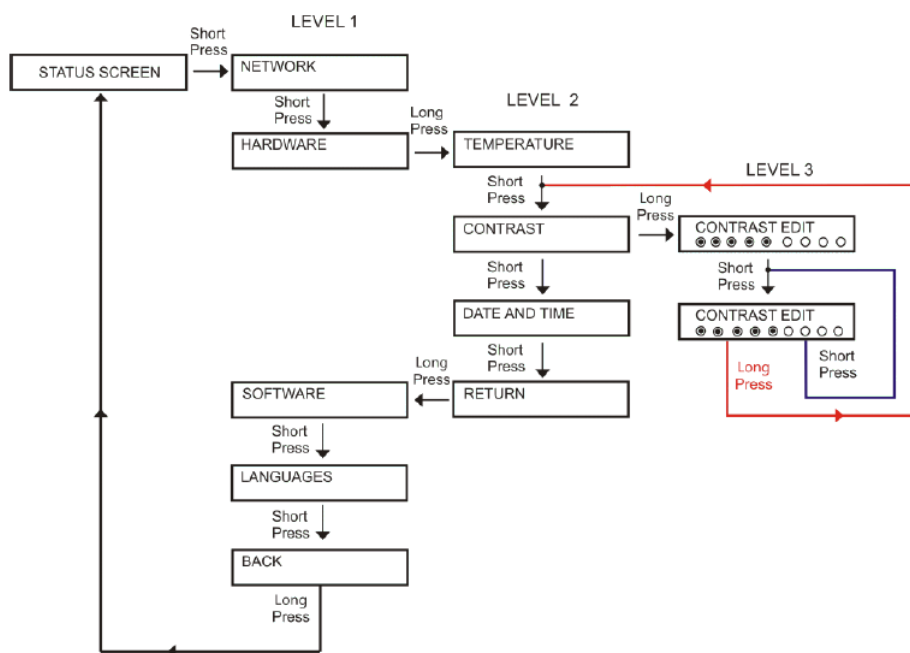
Figure 107: Contrast Adjust

Besides the possibility of the Nexto CPUs menu to be closed through a long touch on the screen diagnostic button *BACK* from level 1, there are also other output conditions that are described below:

- ■ Short touch, at any moment, in the other modules existent on the bus, make the CPU disconnect from the menu and show the desired module diagnostic.
- ■ Idle time, at any level, superior to 5 s.

5.12. Function Blocks and Functions

5.12.1. Special Function Blocks for Serial Communication

The special function blocks for serial interfaces make possible the local access (COM 1 AND COM 2) 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.

altus

Data type	Options	Description
SERIAL_MODE	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).
	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	

altus

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	
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:	

altus

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.

altus

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 local 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 124: Serial Function Blocks Data types

5.12.1.1. 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 108: 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 125: SERIAL_CFG Input Parameters

172

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 126: 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.
```

5.12.1.2. SERIAL_GET_CFG

The function block is used to capture the desired serial port configuration.

Figure 109: 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 127: 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.

altus

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 128: 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.
```

5.12.1.3. 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.

altus

Figure 110: 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 129: 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 130: SERIAL_GET_CTRL Output Parameters

altus

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;
```

5.12.1.4. 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 111: 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 131: 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 132: 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.
```

altus

5.12.1.5. 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.
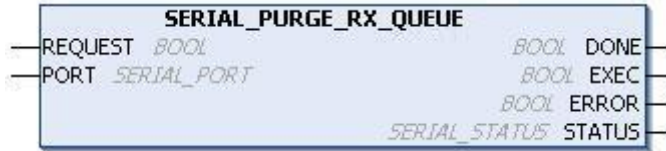
Figure 112: Block Used to Clean the RX Queue

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 133: 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 134: SERIAL_PURGE_RX_QUEUE Output Parameters

altus

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.
```

5.12.1.6. 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 113: 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 135: 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 136: 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;
```

5.12.1.7. 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 Interfaces Configuration section.

Figure 114: 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.

altus

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 137: 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 138: 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;
```

5.12.1.8.   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.

altus

Figure 115: 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 139: 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 140: SERIAL_SET_CTRL Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

altus

```
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.
```

5.12.1.9.   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).
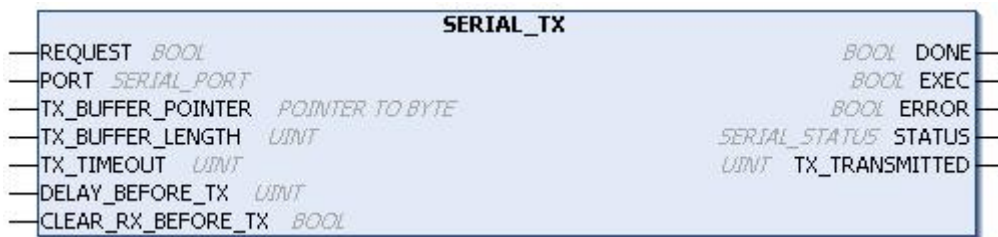
Figure 116: Block for Values Transmission by the Serial

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).

altus

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 141: 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 142: SERIAL_TX Output Parameters

altus

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;
```

5.12.2. 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 Main-Task.

> **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.

5.12.2.1. REFRESH_INPUT

This function block is used to update the specified module inputs without the necessity to wait for the cycle to be completed. It is important to notice that the filters configured in the MasterTool IEC XE and the update time of the module inputs will have to be considered in effective time of the inputs update in the application developed by the user.

> **ATTENTION**
>
> The *REFRESH_INPUT* function must only be used in MainTask.
> To update inputs in other tasks, the option *Enable I/O update per task* must be selected, for further information about this option, consult Table 41.

altus

> **ATTENTION**
>
> *REFRESH_INPUT* function does not support inputs that have been mapped to symbolic variables. For proper operation it is necessary that the input is mapped to a variable within the memory direct representation of input variables (%I).

> **ATTENTION**
>
> The *REFRESH_INPUT* function updates only the direct variables %I that are declared in the "Bus: I/O Mapping" tab of the module addressed in the respective rack/slot of the function. In the case of communication modules/interfaces (MODBUS, Profibus, etc.), the update does not include the direct variables of the device mappings.

```
REFRESH_INPUT
byRackNumber
bySlotNumber
```
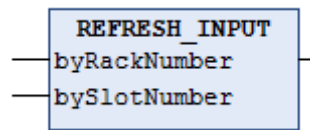
Figure 117: Block for Input Updating

Input parameters	Type	Description
byRackNumber	BYTE	Rack number.
bySlotNumber	BYTE	Position number where the module is connected.

Table 143: REFRESH_INPUT Input Parameters

Possible *TYPE_RESULT*:

- **OK_SUCCESS:** Execution success.
- **ERROR_FAILED:** This error is returned if the function is called for a module that has only outputs, or also if the option *Always update variables* (located in the module's configuration screen, tab *I/O Mapping*) is not checked.
- **ERROR_NOTSUPPORTED:** The called routine is not supported by the product.
- **ERROR_PARAMETER:** Invalid / unsupported parameter.
- **ERROR_MODULE_ABSENT:** The module is absent in the bus.
- **ERROR_MODULE_NOTCONFIGURED:** The module is not configured in the application.
- **ERROR_MODULE_NOTRUNNING:** The module is not running (is not in operational state).
- **ERROR_MODULE_COMMFAIL:** Failure in the communication with the module.
- **ERROR_MODULE_NOTFOUND:** The module was not found in the application or is not supported.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Info: TYPE_RESULT;
byRackNumber: BYTE;
bySlotNumber: BYTE;
END_VAR
//INPUTS:
byRackNumber := 0;
bySlotNumber := 10;
//FUNCTION:
Info := REFRESH_INPUT (byRackNumber, bySlotNumber); //Function call.
//Variable "Info" receives possible function errors.
```

altus

5.12.2.2. REFRESH_OUTPUT

This function block is used to update the specified module outputs. It is not necessary to wait until the cycle is finished. It is important to notice that the update time of the module outputs will have to be considered in the effective time of the outputs update in the application developed by the user.

> **ATTENTION**
>
> The *REFRESH_OUTPUT* function must only be used in MainTask.
> To update outputs in other tasks, the option *Enable I/O update per task* must be selected, for further information about this option, consult Table 41.

> **ATTENTION**
>
> *REFRESH_OUTPUT* function does not support inputs that have been mapped to symbolic variables. For proper operation it is necessary that the input is mapped to a variable within the memory direct representation of input variables (%Q).

> **ATTENTION**
>
> The *REFRESH_OUTPUT* function updates only the direct variables %Q that are declared in the "Bus: I/O Mapping" tab of the module addressed in the respective rack/slot of the function. In the case of communication modules/interfaces (MODBUS, Profibus, etc.), the update does not include the direct variables of the device mappings.
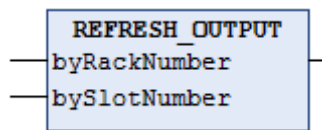
Figure 118: Block for Output Updating

Input parameters	Type	Description
byRackNumber	BYTE	Rack number.
bySlotNumber	BYTE	Position number where the module is connected.

Table 144: REFRESH_OUTPUT Input Parameters

Possible *TYPE_RESULT*:

- **OK_SUCCESS:** Execution success.
- **ERROR_FAILED:** This error is returned if the function is called for a module that has only inputs, or also if the option *Always update variables* (located in the module's configuration screen, tab *I/O Mapping*) is not checked.
- **ERROR_NOTSUPPORTED:** The called routine is not supported by the product.
- **ERROR_PARAMETER:** Invalid / unsupported parameter.
- **ERROR_MODULE_ABSENT:** The module is absent in the bus.
- **ERROR_MODULE_NOTCONFIGURED:** The module is not configured in the application.
- **ERROR_MODULE_NOTRUNNING:** The module is not running (is not in operational state).
- **ERROR_MODULE_COMMFAIL:** Failure in the communication with the module.
- **ERROR_MODULE_NOTFOUND:** The module was not found in the application or is not supported.

Utilization example in ST language:

altus

```
PROGRAM UserPrg
VAR
Info: TYPE_RESULT;
byRackNumber: BYTE;
bySlotNumber: BYTE;
END_VAR
//INPUTS:
byRackNumber := 0;
bySlotNumber := 10;
//FUNCTION:
//Function call.
Info := REFRESH_OUTPUT (byRackNumber, bySlotNumber);
//Variable "Info" receives possible function errors.
```

5.12.3. PID Function Block

> **ATTENTION**
>
> The PID function block described up to previous revision L of this manual became obsolete and was removed from this manual.
>
> The PID, PID_INT and PID_REAL function blocks described up to revision C of MP399609, also became obsolete and were also removed from newer versions of that manual. Users that need description of these obsolete function blocks due to maintenance reasons must use revision C of MP399609.
>
> Function blocks PID, PID_INT and PID_REAL must not be used in new projects. These function blocks were replaced by newer function blocks with additional resources, like auto-tuning and support to cascade, override and feed-forward controls. These new function blocks are described in MU214610, and are available after version 1.1.0.0 of library *NextoPID*.

5.12.4. 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).

5.12.4.1. 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 119 depicts the *TOF_RET* block and Figure 120 shows its graphic behavior.

Figure 119: 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 145: 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 146: TOF_RET Output Parameters
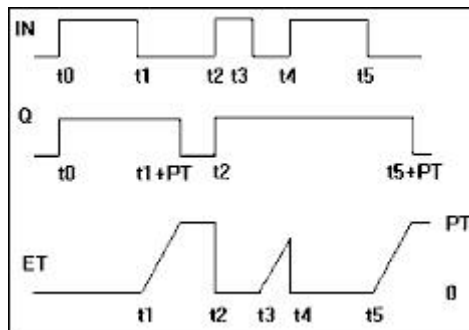
Figure 120: TOF_RET Block Graphic Behavior

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
```

5.12.4.2. 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 121 depicts the *TON_RET* block and Figure 122 shows its graphic behavior.

Figure 121: 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 147: 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 148: TON_RET Output Parameters
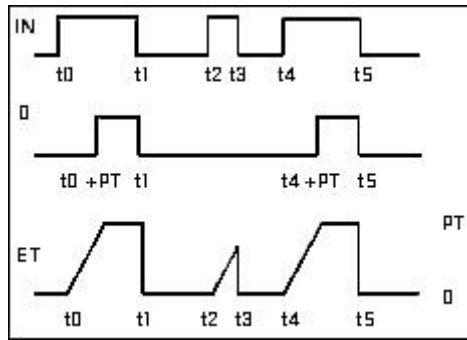
Figure 122: TON_RET Block Graphic Behavior

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
```

5.12.4.3. 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 123 depicts the *TP_RET* and Figure 124 shows its graphic behavior.

Figure 123: 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 149: TP_RET Input Parameters

altus

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 150: TP_RET Output Parameters
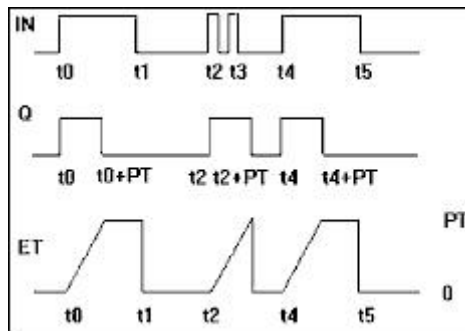
Figure 124: TP_RET Block Graphic Behavior

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
```

5.12.5. User Log

Feature that allows the user to create own records and write to log files on the memory card present in the CPU. The files are generated in a specific directory of the memory card in the CSV format, allowing viewing in text editors and spreadsheets. The separator was the semicolon character. For more information about the use of the memory card, see section Memory Card.

There are two functions available, one for log information and another to remove all records. The following is a description of the types of data used by the functions:

altus

Data type	Option	Description
USER_LOG_EVENT_TYPES	USER_LOG_EVENT_ERROR	The user is free to use the best indication according to log message severity.
	USER_LOG_EVENT_DEBUG	
	USER_LOG_EVENT_INFO	
	USER_LOG_EVENT_WARN	
USER_LOG_MESSAGE		Log message with 150-character max.
USER_LOG_ERROR_CODES	USER_LOG_OK	The operation was performed successfully.
	USER_LOG_FAILED	The operation was not performed successfully. The reason for the failure can be checked in the system logs – see section System Log.
	USER_LOG_BUFFER_FULL	Messages are being added beyond the processing capacity.
	USER_LOG_NO_MEMORY	At the time, there were no resources to perform the operation.
	USER_LOG_FILE_SYSTEM_ERROR	There was an error while accessing the memory card or there is no available space. Error information can be verified in the logs of system – see section System Log.
	USER_LOG_NO_MEMORY_CARD	There is a memory card present in the CPU.
	USER_LOG_MEMORY_CARD_FULL	There is no free space available on the memory card.
	USER_LOG_PROCESSING	The resource is busy executing the last operation, for example, deleting all log files.

Table 151: Data Type for User Log

The following are described the two functions available in the *LibLogs* library on MasterTool IEC XE. To perform the procedure of inserting a library, see the MasterTool IEC Programming Manual – MP399609, section Libraries.

> **ATTENTION**
>
> The User Logs are available only until version 1.3.0.20 of Nexto Series CPUs. In the same way to use this feature is necessary version 1.40 or higher of MasterTool IEC XE.

5.12.5.1. UserLogAdd

This function is used to add a new user log message, adding in a new line to the log file on the memory card. The message must have a maximum length of 150 characters, and the event type of the message. Application variables can be registered using conversion to string and concatenation with the main message. The date and time information in UTC (timestamp) is automatically added in the message with a resolution of milliseconds where the event was registered. The date and time information is also used in the formation of the names of the log files.

The *UserLogAdd* function can be used to enter multiple messages within a single task and also in different application tasks. However independent of each execution of the function in the application, being on the same task or on different tasks, all use the same feature to record the desired messages. For this reason it is recommended that the addition of messages using the *UserLogAdd* function in the application be held every 50 ms to prevent the return of buffer overload. If the function is performed in periods shorter than the indicated, but respect the average time of 50 ms between each message addition at the end of the interval for the task, also prevents the return of buffer overload. So that the logs are added correctly, it is important to respect time limits when the card is inserted and at startup of the CPU, mentioned in section Memory Card. After the operation the function returns the options for the given type *USER_LOG_ERROR_CODES* as Table 151. When the function return is not *USER_LOG_OK*, the message was not registered and the function *UserLogAdd* should be re-executed with the

altus

desired message. In case of return of consecutive writing failures, the memory card can be damaged. The replacement by a healthy memory card ensures that the latest logged messages will be recorded on the card that is not damaged, since the CPU is not restarted.

The figure below represents the function *UserLogAdd* and table below the input parameters:
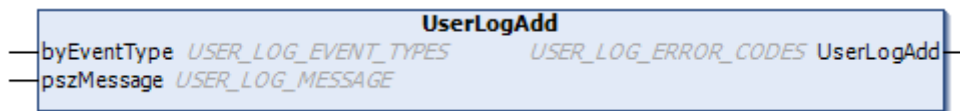
Figure 125: UserLogAdd Function

Input Parameters	Type	Description
byEventType	BYTE	This variable specifies the event type of the log being added as options for the USER_LOG_EVENT_TYPES data type.
pszMessage	USER_LOG_MESSAGE	This variable should contain the set of characters that compose the message to be added to the log file. The message must contain a maximum of 150 characters.

Table 152: UserLogAdd Input Parameters

The log files are generated and organized on the memory card in a specific directory path depending on the CPU serial number and the firmware version installed. For example, for a CPU with serial number 445627 and firmware version 1.4.0.4, the location where the log files should be written to the memory card is *MemoryCard/UserLog/445627/1.4.0.4/*.

The names of the log files are formed by the date and time (timestamp) of the first message. Except when there's a problem to use this name, for example, another existing file with the same name, in this situation it is used the instant date and time. The file name follows the following pattern: *year/month/day/hour/minute/second/millisecond.CSV*. In case of file access problem due to defective sector not enabling to continue writing, will be added to the name of this file the extension *"corrupted"* and a new file will be created. The amount of logs per file is not fixed, varying depending on the size of messages. The amount of created files is limited to 1024 with maximum size of 1 MB each, so the memory card requires 1 GB of free space. When it reaches the limit of 1024 files created on the memory card, during CPU operation, the oldest files are removed so that files with latest logs are preserved, even in cases of partial manual removal of the files in the directory where the files are being written.

The viewing of the log files can be performed through worksheets or conventional text editors. The concatenated information, for improved visualization, may use semicolons between the strings of the message to separate them. One must be careful in formatting cells with floating point values.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
  eLogError : USER_LOG_ERROR_CODES;
  sMessage :USER_LOG_MESSAGE;
END_VAR

IF (m_rTemperature > MAX_TEMPERATURE_ACCEPT) THEN
  sMessage := 'Temperature higher than expected: ';
  sMessage := concat(sMessage, REAL_TO_STRING(m_rTemperature));
  sMessage := concat(sMessage, '°');
  eLogError := UserLogAdd(USER_LOG_EVENT_WARN, sMessage);
//eLogError variable gets possible function errors.
END_IF
```

Log file content example: (*UserLog-201308271506245738.csv*)

```
Model; NX3008
Serial number; 445627
Firmware version; 1.4.0.4

27/08/2013 15:06:24.5738; WARN; Overtemperature: 25°
27/08/2013 16:37:45.3476; WARN; Overtemperature: 25°
28/08/2013 09:10:55.4201; WARN; Overtemperature: 26°
```

5.12.5.2. UserLogDeleteAll

The *UserLogDeleteAll* function performs the deletion of log files present in the directory created specifically for the CPU in which is inserted the memory card, i.e. are only deleted the logs contained in the directory named with the CPU firmware version that exists within the directory with the CPU serial version. The log files deleted are only files that exist at the time of memory card mounting and the generated by the *UserLogAdd* function. Logs of other CPUs and files added manually by the user during execution are not deleted. The figure below represents the function *UserLogDeleteAll*.
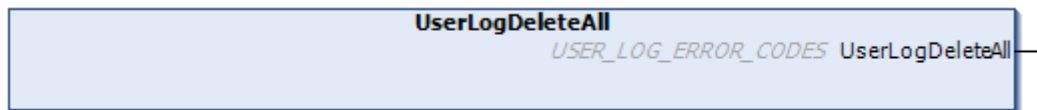
Figure 126: UserLogDeleteAll Function

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
eLogError : USER_LOG_ERROR_CODES;
END_VAR

IF (m_DeleteLogs = TRUE) THEN
eLogError := UserLogDeleteAll();
m_DeleteLogs := FALSE;
//eLogError variable gets possibles function errors.
END_IF
```

> **ATTENTION**
>
> The *UserLogDeleteAll* function's return does not indicate operation completed, just confirmation of execution that can take a large amount of time if there are hundreds of log files in the directory. The function to record the new user log is unavailable right now, returning the *USER_LOG_PROCESSING* option for any operation. The result of the operation can also be checked in the system log.

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 with each object of this tree is called OID (*Object Identifier*).

For all equipment with SNMP, it is mandatory to support MIB-II. In this MIB are described key information for managing Ethernet networks.

5.13.2. SNMP nas UCPs Nexto

The CPUs of the Nexto Series behave as agents in SNMP communication. The information made available through SNMP cannot be manipulated or accessed through the user application, requiring an external SNMP manager to perform access. The table below contains the objects available in the Nexto CPUs. This feature is available after firmware version 1.4.0.33 of the Nexto Series CPUs supports the protocols SNMPv1, SNMPv2c and SNMPv3, and support the MIB-II, where objects are described in RFC-1213.

OID	Name	Description
1.3.6.1.2.1.1	System	Contains name, description, location and other equipment identification information.
1.3.6.1.2.1.2	Interfaces	Contains information of the machine's network interfaces. The ifTable (OID 1.3.6.1.2.1.2.2) has the indexes 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 153: MIB II Objects – Nexto Series SNMP Agent

By default, the SNMP agent is activated, i.e., the service is initialized at the time the CPU is started. The access to the agent information is via the Ethernet interfaces of the Nexto Series CPUs on UDP port 161. So when the service is active, the agent information can be accessed through any one of the Ethernet interfaces available. It is not possible to provide agent information through Ethernet interfaces NX5000 modules. In figure below an example is shown SNMP manager, in which some values are read.

altus
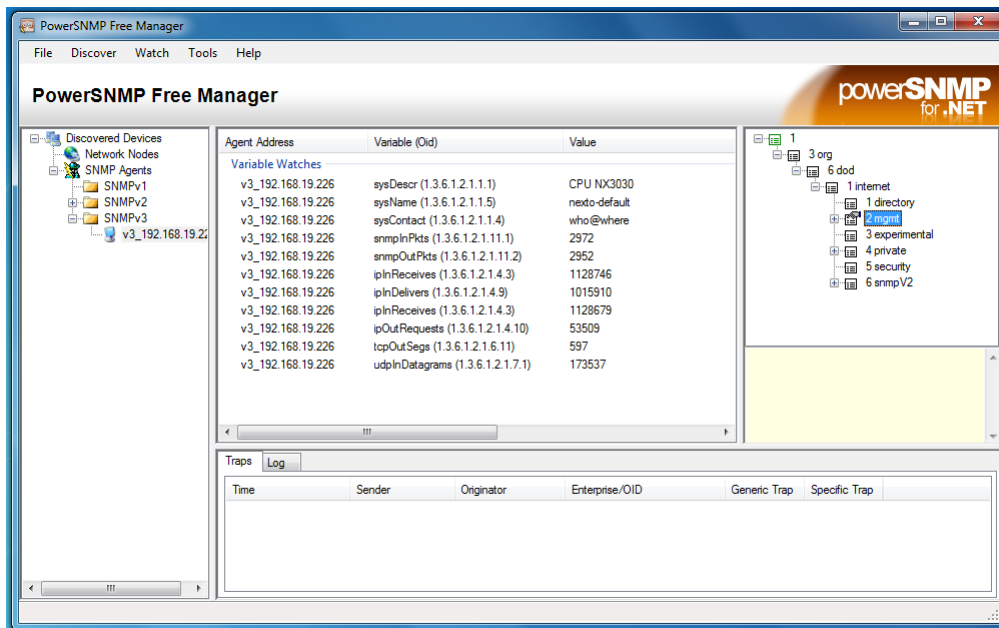
Figure 127: SNMP Manager Example

For SNMPv3, in which there is user authentication and password to requests via SNMP protocol, is provided a standard user described in the User and SNMP Communities section.

If you want to disable the service, change the SNMPv3 user or communities for SNMPv1 / v2c predefined, you must access the System Web Page of the CPU. For details, see the Configuration SNMP section.

5.13.3.   Private MIB

The Private MIB has been discontinued since June 2019.

5.13.4.   Configuration SNMP

SNMP settings can be changed through the System Web Page, in the *CPU Management* tab in 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. Will open a form where you must complete the old username and password, and the new username and password. The other user information SNMPv3 cannot be changed.

To change the data of SNMPv1/v2c communities, the process is similar, just click the *Change* button below the information community. A new screen will open where the new data to the *rocommunity* and *rwcommunity* fields will be inserted. If you fail any of the fields blank, their community will be disabled. That way, if the user leaves the two fields blank, 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 reconfigure the same according to 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.
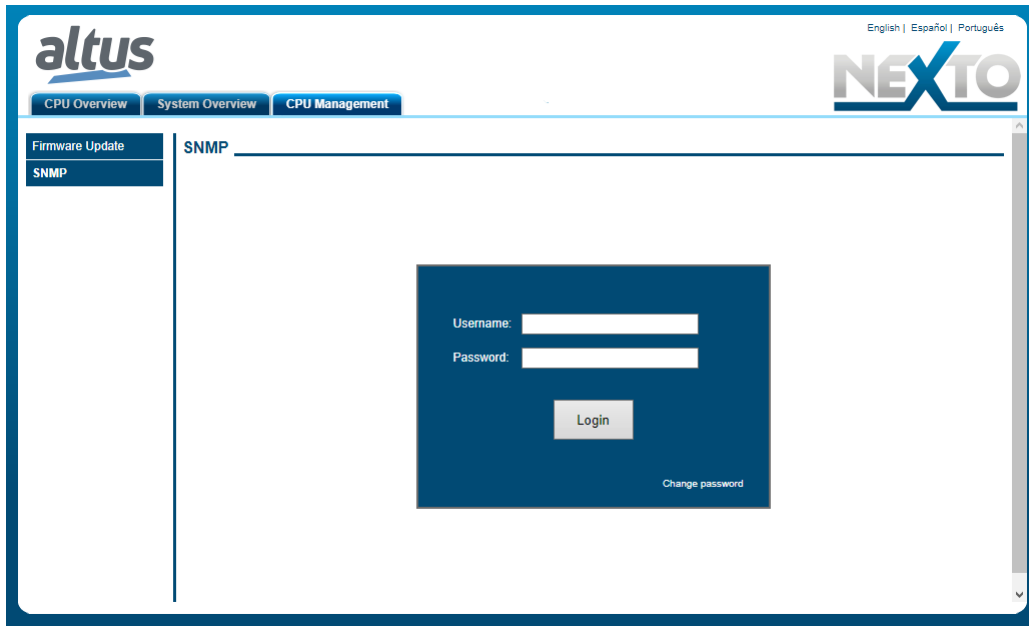
Figure 128: SNMP Login screen

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.
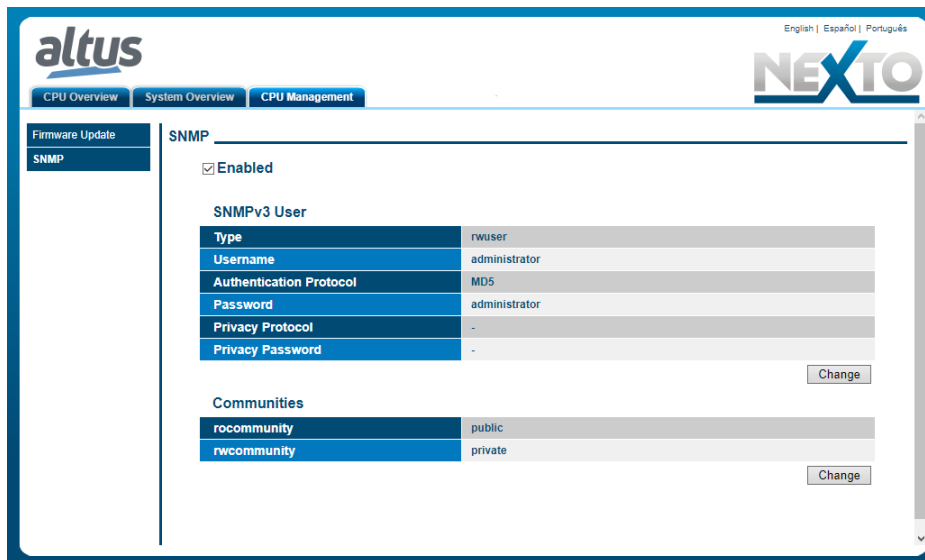
Figure 129: SNMP status configuration screen

ATTENTION

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

5.13.5. User and SNMP Communities

To access the SNMPv1 / v2c of the Nexto Series CPUs, there are two communities, according to table below.

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

Table 154: SNMPv1/v2c Default Communities info

It's possible to access SNMPv3 using default user, see table below:

Username	Type	Authentication Protocol	Password	Privacy Protocol	Privacy Password
administrator	rwuser	MD5	administrator	-	-

Table 155: SNMPv3 Default 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]@$*_.
V3 Password	8	30	[0-9][a-z][A-Z]@$*_.

Table 156: SNMP settings limits

altus

6.   Maintenance

One feature of the Nexto Series is the abnormality diagnostic generation, whether they are failures, errors or operation modes, allowing the operator to identify and solve problems which occurs in the system easily.

The Nexto CPUs permit many ways to visualize the diagnostics generated by the system, which are:

- One Touch Diag
- Diagnostics via LED
- Diagnostics via System Web Page
- Diagnostics via Variables
- Diagnostics via Function Blocks

The first one is an innovating feature of Nexto Series, which allows a fast access to the application abnormal conditions. The second is purely visual, generated through two LEDs placed on the panel (DG and WD) and also through the LEDs placed in the RJ45 connector (exclusive for Ethernet connection). The next feature is the graphic visualization in a WEB page of the rack and the respective configured modules, with the individual access allowed of the operation state and the active diagnostics. The diagnostics are also stored directly in the CPU variables, either direct representation (%Q) or attributed (AT variable), and can be used by 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.   Module Diagnostics

6.1.1.   One Touch Diag

The One Touch Diag (OTD), or single touch diagnostics, is an exclusive feature the Nexto Series brings for the programmable controllers. With this new concept the user can verify the diagnostics of any module connected to the system straight on the CPU graphic display with a single touch on the module Diagnostic Switch. This is a powerful diagnostic tool which can be used offline (with no need of supervisory or programming software) making easier to find and solve quickly possible problems.

The diagnostics key is placed on the CPU upper part, in an easy access place and, besides giving active diagnostics, allows the access to the navigation menu, described in the Configuration – CPU's Informative and Configuration Menu section.

The figure below shows the CPU switch placement:
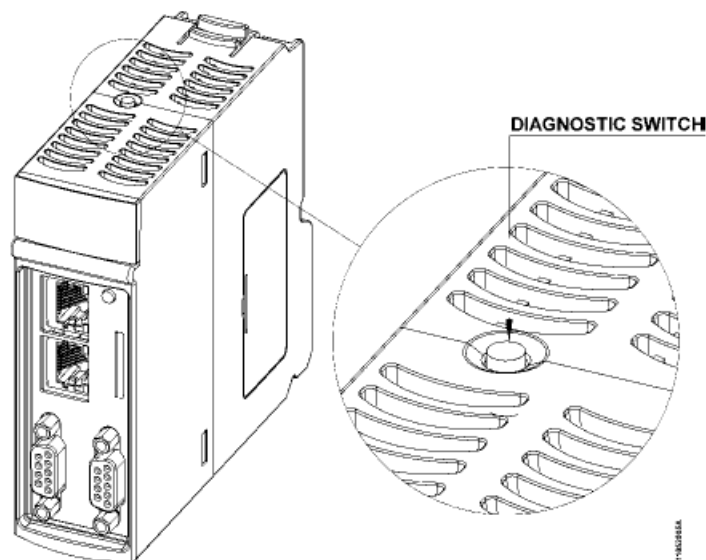
Figure 130: Diagnostic Switch

With only a short touch, the CPU starts to show the bus diagnostics (when active, otherwise shows the "NO DIAG" message). Initially, the Tag is visualized (configured in the module properties in the MasterTool IEC XE software, following

altus

the IEC 61131-3 standard), in other words, the name attributed to the CPU, and after that all diagnostics are shown, through CPU display messages. This process is executed twice on the display. Everything occurs automatically as the user only has to execute the first short touch and the CPU is responsible to show the diagnostics. The diagnostics of other modules present on the bus are also shown on the CPU graphic display by a short press in the diagnostic module button, in the same presentation model of diagnostics.

The figure below shows the process starting with the short touch, with the conditions and the CPU times presented in smaller rectangles. It is important to stress the diagnostics may have more than one screen, in other words, the specified time in the block diagram below is valid for one of them.

LEVEL 1

```
                    Short
                    Press
   STATUS SCREEN  ───────▶  TAG UCP  ◀──────────────┐
                                │                    │
                            Delay 2s                 │
                                │                    │
                          DIAGNOSTIC 1               │
                                │                    │
                            Delay 2s                 │
                                │                    │
                          DIAGNOSTIC 2               │
                                │                    │
                            Delay 2s                 │
                                │                    │
                          DIAGNOSTIC 3               │
                                │                    │
                            Delay 2s                 │
                                │                    │
                        DIAGNOSTIC "N"               │
                                │                    │
                            Delay 2s                 │
                                │                    │
                              ◆ Twice? ◆ ──No──▶─────┘
                                │
                               Yes
                                ▼
```
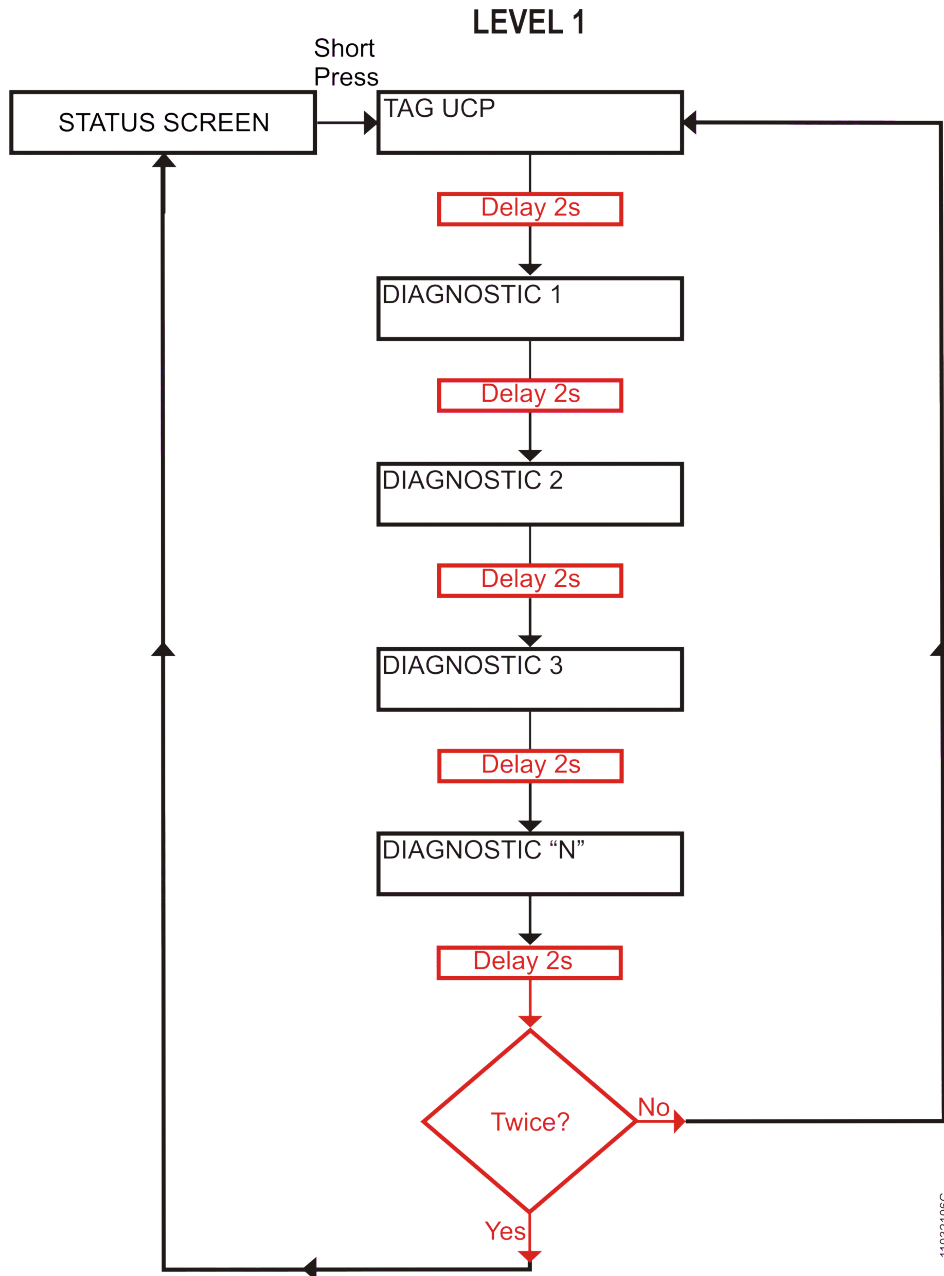
Figure 131: CPU Diagnostics Visualization

Before all visualization process be concluded, it is just to give a short touch on the diagnostic switch, at any moment, or press the diagnostic switch from any I/O module connected to the bus. Also, it is important to observe that the One Touch Diag could be available when the module could be in Operational Mode.

In case a long touch is executed, the CPU goes to navigation menu, which is described in the Configuration – CPU's Informative and Configuration Menu section.

The table below shows the difference between the short touch time, the long touch time and stuck button.

Touch type	Minimum time	Maximum time	Indication condition
No touch	-	59.99 ms	-
Short touch	60 ms	0.99 s	Release
Long touch	1 s	20 s	More than 1 s till 20 s
Locked Switch	20.01 s	(∞)	Diagnostics indication, see on Table 162

Table 157: One Touch Time

The messages presented on the Nexto CPU graphic display, correspondent to the diagnostics, are described in the Diagnostics via Variables section, on Table 162.

If any situation of stuck button occur in one of the I/O modules, the diagnostic button of this module will stop of indicate the diagnostics on CPU graphic display when is pressed. In this case, the CPU will indicate that there is a module with active diagnostics. To remove this diagnostic from the CPU, a hot swap must be done in the module where the diagnostic is active.

For further details on the procedure for viewing the diagnostics of the CPU or other bus modules, see description in the User Manual Nexto Series – MU214600.

6.1.2. Diagnostics via LED

This product have a LED for diagnostic indication (LED DG) and a LED for watchdog event indication (LED WD). The Tables 158 and 159 show the meaning of each state and its respective descriptions.

6.1.2.1. DG (Diagnostic)

Green	Red	Description	Causes	Priority
Off	Off	Not used	No power supply. Hardware problem	-
On	Off	All applications in execution mode (Run)	-	3 (Low)
Off	On	All applications in stopping mode (Stop)	-	3 (Low)
Blinking 2x	Off	Bus modules with diagnostic	At least, a bus module, including the CPU, is with an active diagnostic	1
Blinking 3x	Off	Data forcing	Some memory area is being forced by the user through MasterTool IEC XE	2
Off	Blinking 4x	Configuration or hardware error in the bus	The bus is damaged or is not properly configured	0 (High)

Table 158: Description of the Diagnostic LEDs States

altus

6.1.2.2. WD (Watchdog)

Red LED	Description	Causes	Priority
Off	No watchdog indication	Normal operation	3 (Low)
Blinking 1x	Software watchdog	User application watchdog	2
On	Hardware watchdog	Damaged module and/or corrupted operational system	1 (High)

Table 159: Description of the Watchdog LED States

Notes:

Software Watchdog: In order to remove the watchdog indication, make an application reset or turn off and turn on the CPU again. This watchdog occurs when the user application execution time is higher than the configured watchdog time.

The diagnostics can be checked in the Exception.wExceptionCode variable, see on Table 166.

Hardware Watchdog: In order to reset any watchdog indication, as in the WD LED or in the Reset.bWatchdogReset operand, the module must be disconnected from the power supply.

In order to verify the application conditions in the module restart, see configurations on Table 41.

6.1.2.3. RJ45 Connector LEDs

Both LEDs placed in the RJ45 connectors, help the user in the installed physical network problem detection, indicating the network Link speed and the existence of interface communication traffic. The LEDs meaning is presented on table below.

Yellow	Green	Meaning
○	○	Network LINK absent
●	○	10 Mbytes/s network LINK
●	●	100 Mbytes/s network LINK
X	-	Ethernet network transmission or reception occurrence, for or to this IP address. Blinks on Nexto CPU demand and not every transmission or reception, in other words, it may blink on a lower frequency than the real transmission or reception frequency

Table 160: Ethernet LEDs Meaning

6.1.3. Diagnostics via System Web Page

Besides the previously presented features, the Nexto Series brings to the user an innovating access tool to the system diagnostics and operation states, through a System Web Page.

The utilization, besides being dynamic, 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 desired CPU's System Web Page, it is just to use a standard browser (Internet Explorer 7 or superior, Mozilla Firefox 3.0 or superior and Google Chrome 8 or superior) and type, on the address bar, the CPU IP address (Ex.: http://192.168.1.1). First, the CPU information is presented, according to Figure 132:

altus
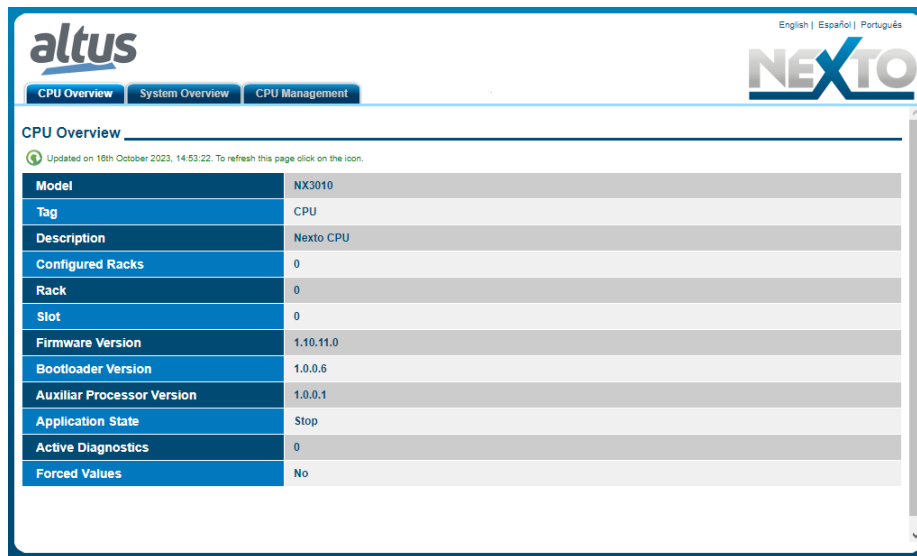
Figure 132: Initial Screen

There is also the *Bus Information* tab, which can be visualized through the Rack or the present module list (option on the screen right side). While there is no application on the CPU, this page will display a configuration with the largest available rack and a standard power supply, connected with the CPU. When the Rack visualization is used, the modules that have diagnostics blink and assume the red color, as shown on Figure 133. Otherwise a list with the system connected modules, Tags and active diagnostics number is presented:
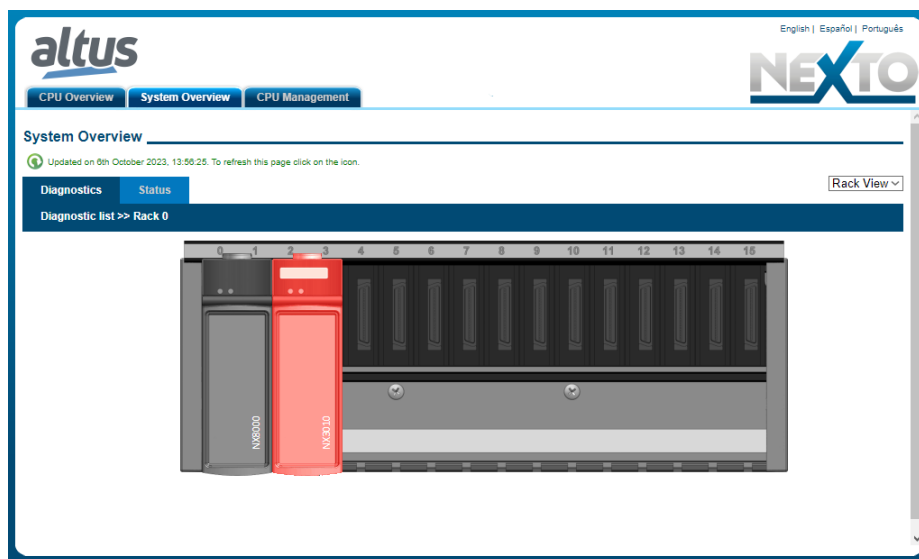
Figure 133: System Information

When the module with diagnostics is pressed, the module active(s) diagnostic(s) are shown, as illustrated on Figure 134:

> **ATTENTION**
>
> When a CPU is restarted and the application goes to exception in the system's startup, the diagnostics will not be valid. It is necessary to fix the problem which generates the application's exception so that the diagnostics are updated.
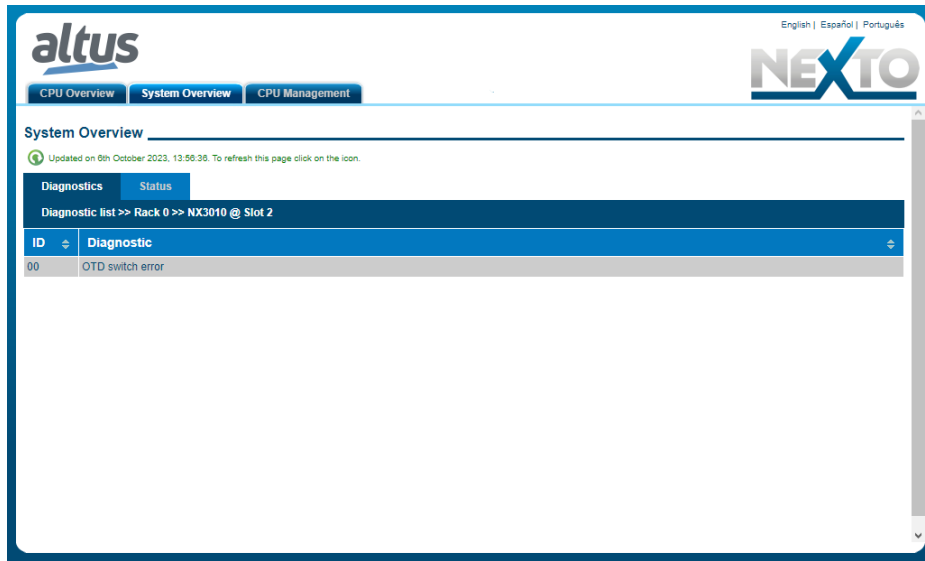
Figure 134: System Diagnostics

In case the Status tab is selected, the state of all detailed diagnostics is shown on the screen, as illustrated on Figure 135:
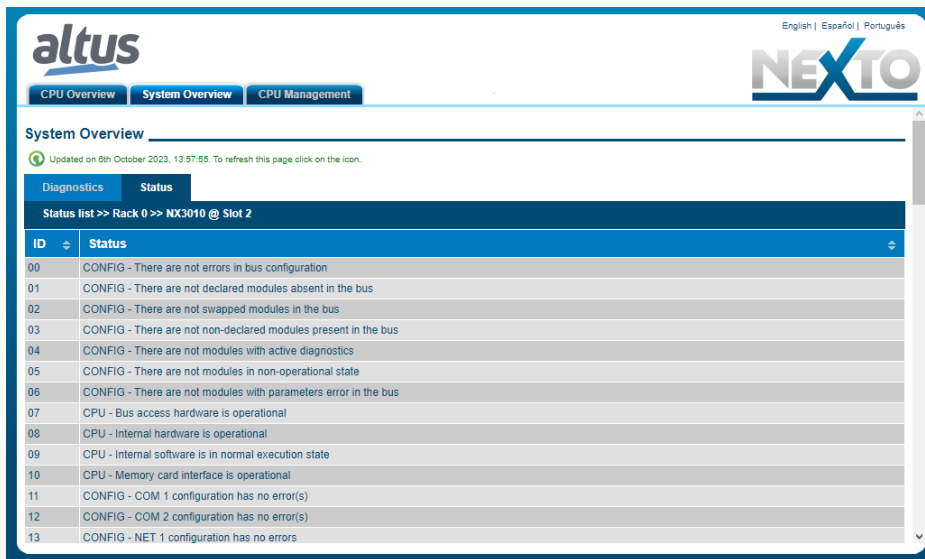
Figure 135: System Status

The user can choose to visualize two language options: Portuguese and English. Simply change in the upper right part of the screen to the desired language.

6.1.4. Diagnostics via Variables

The Nexto Series CPUs have many variables for diagnostic indication. There are data structures with the diagnostics of all modules declared on the bus, mapped on the variables of direct representation %Q, and defined symbolically through the AT directive, in the GVL System_Diagnostics created automatically by the MasterTool IEC XE.

The table below summarizes the diagnostic byte/words division:

Byte	Description
0 to 3	CPU summarized diagnostics.
4 to 558	CPU detailed diagnostics.

Table 161: CPU Diagnostics Division

6.1.4.1. Summarized Diagnostics

The table below shows the meaning of each CPU summarized diagnostic bit:

Direct Variable		Diagnostics Message	AT Variable DG_Module.tSummarized.*	Description
Variable	**Bit**			
-	-	NO DIAG	-	There is no active diagnostic.
%QB(n)	0	CONFIG. MISMATCH	bConfigMismatch	TRUE – There is a configuration problem in the bus, as the module inserted in the wrong position.
				FALSE – The bus is configured correctly.
	1	ABSENT MODULES	bAbsentModules	TRUE – One or more declared modules are absent.
				FALSE – All declared modules were detected in the bus.
	2	SWAPPED MODULES	bSwappedModules	TRUE – There are changed modules in the bus.
				FALSE – There are no changed modules in the bus.
	3	NON-DECLARED MODULES	bNonDeclaredModules	TRUE – One or more modules in the bus were not declared in the configuration.
				FALSE – All modules were declared.
	4	MODULES W/ DIAGNOSTICS	bModulesWithDiagnostic	TRUE – One or more modules in the bus are with active diagnostic.
				FALSE – There is no active diagnostic in the modules in the bus.
	5	MODULES W/ FATAL ERROR	bModuleFatalError	TRUE – One or more modules in the bus are in fatal error.
				FALSE – All modules are working properly.
	6	MODULES W/ PARAM. ERROR	bModuleParameterError	TRUE – One or more modules in the bus have parameterization error.
				FALSE – All modules are parameterized.
	7	BUS ERROR	bWHSBBusError	TRUE – Master indication there is failure in the WHSB bus.
				FALSE – The WHSB bus is working properly.
	0	HARDWARE FAILURE	bHardwareFailure	TRUE – CPU hardware failure.
				FALSE – The hardware is working properly.
	1	SOFTWARE EXCEPTION	bSoftwareException	TRUE – One or more exceptions generated by the software.

altus

Direct Variable		Diagnostics Message	AT Variable DG_Module.tSummarized.*	Description
Variable	**Bit**			
%QB(n+1)				FALSE – No exceptions generated in the software.
	2	HARDWARE WATCHDOG	bHwWatchdogReset	TRUE - The CPU restarted by hardware watchdog at least once.
				FALSE - The CPU is operating normally.
	3	ERROR IN MEMORY CARD	bMemoryCardError	TRUE – The memory card is inserted in the CPU, but is not working properly.
				FALSE – The memory card is working properly.
	4	COM1 CONF. ERROR	bCOM1ConfigError	TRUE – Some error occurred during, or after, the COM 1 serial interface configuration.
				FALSE – The COM 1 serial interface configuration is correct.
	5	COM2 CONF. ERROR	bCOM2ConfigError	TRUE – Some error occurred during, or after, the COM 2 serial interface configuration.
				FALSE – The COM 2 serial interface configuration is correct.
	6	NET1 CONF. ERROR	bNET1ConfigError	TRUE – Some error occurred during, or after, the NET 1 Ethernet interface configuration.
				FALSE – The NET 1 Ethernet interface configuration is correct.
%QB(n+2)	0	INVALID DATE/TIME	bInvalidDateTime	TRUE – The date or hour are invalid.
				FALSE – The date and hour are correct.
	1	RUNTIME RESET	bRTSReset	TRUE – The RTS (Runtime System) has been restarted at least once. This diagnostics is only cleared in the system restart.
				FALSE – The RTS (Runtime System) is operating normally.
	2	OTD SWITCH ERROR	bOTDSwitchError	TRUE – True in case the OTD key has been locked for more than 20 s at least once while the CPU was energized. This diagnostic is only cleared in the system restart.
				FALSE – The key is not currently locked or was locked while the CPU was energized.
%QB(n+3)	0	ABSENT RACK	bAbsentRacks	TRUE – One or more declared racks are absent.
				FALSE – There are no absent racks.
	1	DUPLICATED RACK	bDuplicatedRacks	TRUE – There are racks with a duplicated identification number.
				FALSE – There are no racks with a duplicated identification number.
	2	INVALID RACK	bInvalidRacks	TRUE – There are racks with an invalid identification number.
				FALSE – There are no racks with an invalid identification number.

Direct Variable		Diagnostics Message	AT Variable DG_Module.tSummarized.*	Description
Variable	Bit			
	3	NON DECLARED RACK	bNonDeclaredRacks	TRUE – There are racks with a non-declared identification number.
				FALSE – There are no racks with a non-declared identification number.
	4	DUPLICATED SLOT	bDuplicatedSlots	TRUE – There are some duplicated slot address.
				FALSE – There are no duplicated slot address.

Table 162: CPU Summarized Diagnostics

Notes:

Direct Representation Variable: "n" represents the value set in the CPU through the MasterTool IEC XE software, such as initial address diagnostics.

AT Directive: In the description of the symbolic variables which use the AT directive to make the mapping in direct addressing variables, the syntax that must be put before the desired summarized diagnostic is DG_Module.tSummarized, when the Module word is replaced by the used CPU. E.g. for the incompatible configuration diagnostic it must be used the variable: DG_NX3010.tSummarized.bConfigMismatch. The AT directive is a word reserved in the programming software, used only for diagnostic indication.

Configuration Mismatch: The incompatible configuration diagnostic is generated if one or more modules of the rack does not correspond to the declared one, so, in the absence or different modules conditions. The modules inserted in the bus that were not declared in the project are not considered.

Swapped Modules: If only two modules are changed between themselves in the bus, then changed diagnostic can be identified. Otherwise, the problem is treated as "Incompatible Configuration".

Modules with Fatal Error: In case the modules with fatal error diagnostic is true, it must be verified which is the problematic module in the bus and send it to Altus Technical Assistance, as it has hardware failure.

Module with Parameterization Error: In case the parameterization error diagnostic is true, it must be verified the module in the bus are correctly configured and if the firmware and MasterTool IEC XE software version are correct. If the problem occurred when inserting a module on the bus, make sure the module supports hot swapping.

Bus Error: Considered a fatal error, interrupting the access to the modules in the bus. In case the bus error diagnostic is true, an abnormal situation due to the hot exchange configuration selected might have occurred or a hardware problem in the bus communication lines, then, contact Altus Technical Assistance.

Hardware Failure: In case the Hardware Failure diagnostic is true, the CPU must be sent to Altus Technical Assistance, as it has problems in the RTC, auxiliary processor, 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 CPU detailed diagnostics table.

Diagnostic Message: The diagnostics messages can be visualized by the CPU graphic display using the OTD key or using the CPU's System Web Page.

6.1.4.2. Detailed Diagnostics

The tables below contain Nexto Series' CPUs detailed diagnostics. It is important to have in mind the observations below before consulting them:

- **Visualization of the Diagnostics Structures:** The Diagnostics Structures added to the Project can be seen at the item "*Library Manager*" of MasterTool IEC XE tree view. There, it is possible to see all data types defined in the structure.
- **Counters:** All CPU diagnostics counters return to zero when their limit value is exceeded.
- **Direct representation variable:** "n" represents the value configured at the CPU through MasterTool IEC XE as the initial diagnostics address.
- **AT Directive:** At the description of symbolic variables that use the AT directive to map it in direct mapping variables, the syntax to be used before the desired summarized diagnostic is *DG_Module.tDetailed.*, where the word Module must be replaced by the CPU being used. The AT directive is a reserved word of the programmer, and some symbolic variables that use this directive indicate diagnostics.

Direct representation	Size	AT Variable DG_Modulo.tDetailed.*	Description
%QD(n+4)	DWORD	Target. dwCPUModel	NX3003 = 0x3003 NX3004 = 0x3004 NX3005 = 0x3005 NX3010 = 0x3010 NX3020 = 0x3020 NX3030 = 0x3030
%QB(n+8)	BYTE ARRAY(4)	Target. abyCPUVersion	Firmware version.
%QB(n+12)	BYTE ARRAY(4)	Target. abyBootloaderVersion	Bootloader version.
%QB(n+16)	BYTE ARRAY(4)	Target. abyAuxprocVersion	Auxiliary processor version.

Table 163: Target Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QX(n+20).0	BIT	Hardware. bAuxprocFailure	Failure in the communication between the auxiliary processor and the principal processor.
%QX(n+20).1	BIT	Hardware. bRTCFailure	The main processor is not enabled to communicate with the RTC (CPU's clock).
%QX(n+20).2	BIT	Hardware. bThermometerFailure [1]	Failure in the communication between the thermometer and the main processor.
%QX(n+20).3	BIT	Hardware. bLCDFailure	Failure in the communication between the LCD screen and the main processor.

Table 164: Hardware Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QW(n+21)	WORD	Exception. wExceptionCode	Exception code generated by the RTS. See Table 166.
%QB(n+23)	BYTE	Exception. byProcessorLoad	Level, in percentage (%), of charge in the processor.

Table 165: Exception Detailed Diagnostics Group Description

Note:

Exception Code: the code of the exception generated by the RTS (Runtime System) can be consulted below:

Code	Description	Code	Description
0x0000	There is no exception code.	**0x0051**	Access violation.
0x0010	Watchdog time of the IEC task expired (Software Watchdog).	**0x0052**	Privileged instruction.
0x0012	I/O configuration error.	**0x0053**	Page failure.
0x0013	Checksum error after the program download.	**0x0054**	Stack overflow.
0x0014	Fieldbus error.	**0x0055**	Invalid disposition.
0x0015	I/O updating error.	**0x0056**	Invalid maneuver.
0x0016	Cycle time (execution) exceeded.	**0x0057**	Protected page.
0x0017	Program online updating too long.	**0x0058**	Double failure.
0x0018	External references not resolved.	**0x0059**	Invalid OpCode.

altus

Code	Description	Code	Description
0x0019	Download rejected.	**0x0100**	Data type misalignment.
0x001A	Project not loaded, as the retentive variables cannot be reallocated.	**0x0101**	Arrays limit exceeded.
0x001B	Project not loaded and deleted.	**0x0102**	Division by zero.
0x001C	Out of memory stack.	**0x0103**	Overflow.
0x001D	Retentive memory is corrupted and cannot be mapped.	**0x0104**	Cannot be continued.
0x001E	Project can be loaded but causes a drop later on.	**0x0105**	Watchdog in the processor load of all IEC task detected.
0x0021	Target of startup application does not match to the current target.	**0x0150**	FPU: Not specified error.
0x0022	Scheduled tasks error.	**0x0151**	FPU: Operand is not normal.
		0x0152	FPU: Division by zero.
0x0023	Downloaded file Checksum with error.	**0x0153**	FPU: Inexact result.
0x0024	Retentive identity is not correspondent to the current identity of the boot project program	**0x0154**	FPU: Invalid operation.
0x0025	IEC task configuration failure.	**0x0155**	FPU: Overflow.
0x0026	Application working with wrong target.	**0x0156**	FPU: Stack verification.
0x0050	Illegal instruction.	**0x0157**	FPU: Underflow.

Table 166: RTS Exception codes

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QB(n+25)	BYTE	RetainInfo. byCPUInitStatus	CPU Startup Status: 01: Hot start 02: Warm Start 03: Cold Start Note: These variables are reset in every powerup.
%QW(n+26)	WORD	RetainInfo. wCPUColdStartCounter	Increments when the CPU starts with loss of retentivity. (0 to 65535)
%QW(n+28)	WORD	RetainInfo. wCPUWarmStartCounter	Increments when the CPU starts normally with valid retain data. (0 to 65535)
%QW(n+30)	WORD	RetainInfo. wCPUHotStartCounter	Disturbance counter less than CPU power failure support time. (0 to 65535).
%QW(n+32)	WORD	RetainInfo. wRTSResetCounter	Counter of resets performed by RTS (Runtime System). (0 to 65535).

Table 167: RetainInfo Group Detailed Diagnostics

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QX(n+36).0	BIT	Reset. bBrownOutReset	The CPU was restarted due a failure in the power supply in the last startup.
%QX(n+36).1	BIT	Reset. bWatchdogReset	The CPU was restarted due the active watchdog in the last startup.

Table 168: Reset Detailed Diagnostics Group Description

Note:

Brownout Reset: The brownout reset diagnostic is only true when the power supply exceed the minimum limit required in its technical characteristics, remaining in low-voltage, i.e. without undergoing any interrupt. The CPU will identify the drop in supply and will indicate the power failure diagnostic. When the voltage is reestablished, the CPU will automatically reset and will indicate the brownout reset diagnostic.

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QX(n+37).0	BIT	Thermometer. bOverTemperatureAlarm [1]	Alarm generated due internal temperature at 85 °C or above it.
%QX(n+37).1	BIT	Thermometer. bUnderTemperatureAlarm [1]	Alarm generated due internal temperature at 0 °C or under it.
%QD(n+38)	DINT	Thermometer. diTemperature [1]	Temperature read in the internal sensor of the CPU.

Table 169: Thermometer Detailed Diagnostics Group Description

Note:

Temperature: In order to see the temperature directly in the memory address, a conversion must be made, since the data size is DINT and monitoring is done in 4 bytes. Therefore, it's recommended to use the associated symbolic variable, because it already provides the final temperature value.

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+42)	BYTE	Serial.COM1. byProtocol	Protocol selected in the COM 1: 00: Without protocol 01: MODBUS RTU Master 02: MODBUS RTU Slave 03: Other protocol
%QD(n+43)	DWORD	Serial.COM1. dwRXBytes	Counter of characters received from COM 1 (0 to 4294967295).
%QD(n+47)	DWORD	Serial.COM1. dwTXBytes	Counter of characters transmitted from COM 1 (0 to 4294967295).
%QW(n+51)	WORD	Serial.COM1. wRXPendingBytes	Number of characters left in the reading buffer in COM 1 (0 to 1024).
%QW(n+53)	WORD	Serial.COM1. wTXPendingBytes	Number of characters left in the transmission buffer in COM 1 (0 to 1024).
%QW(n+55)	WORD	Serial.COM1. wBreakErrorCounter	The transmitter is holding the data line at zero for too long, according to the databit configured.
%QW(n+57)	WORD	Serial.COM1. wParityErrorCounter	The received frame has the mismatched parity bit.
%QW(n+59)	WORD	Serial.COM1. wFrameErrorCounter	The received frame has the wrong start point, usually caused by a noise or baud rate mismatch.
%QW(n+61)	WORD	Serial.COM1. wRXOverrunCounter	When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device).

Table 170: Serial COM 1 Detailed Diagnostics Group Description

Note:

Parity Error Counter: When the serial COM 1 is configured Without Parity, this error counter won't be incremented when it receives a message with a different parity. In this case, a frame error will be indicated.

altus

Direct Variable	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+67)	BYTE	Serial.COM2. byProtocol	Protocol selected in the COM 2: 00: Without protocol 01: MODBUS RTU Master 02: MODBUS RTU Slave 03: Other protocol
%QD(n+68)	DWORD	Serial.COM2. dwRXBytes	Counter of characters received from COM 2 (0 to 4294967295).
%QD(n+72)	DWORD	Serial.COM2. dwTXBytes	Counter of characters transmitted through COM 2 (0 to 4294967295).
%QW(n+76)	WORD	Serial.COM2. wRXPendingBytes	Number of characters left in the reading buffer in COM 2 (0 to 1024).
%QW(n+78)	WORD	Serial.COM2. wTXPendingBytes	Number of characters left in the transmission buffer in COM 2 (0 to 1024).
%QW(n+80)	WORD	Serial.COM2. wBreakErrorCounter	The transmitter is holding the data line at zero for too long, according to the databit configured.
%QW(n+82)	WORD	Serial.COM2. wParityErrorCounter	The received frame has the mismatched parity bit.
%QW(n+84)	WORD	Serial.COM2. wFrameErrorCounter	The received frame has the wrong start point, usually caused by a noise or baud rate mismatch.
%QW(n+86)	WORD	Serial.COM2. wRXOverrunCounter	When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device).

Table 171: Serial COM 2 Detailed Diagnostics Group Description

Note:

Parity Error Counter: When the serial COM 2 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.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QX(n+92).0	BIT	Ethernet.NET1. bLinkDown	Indicates link state on NET 1.
%QW(n+93)	WORD	Ethernet.NET1. wProtocol	Protocol selected in NET 1: 00: No protocol
%QX(n+93).0	BIT	Ethernet.NET1. wProtocol. bMODBUS_RTU_ETH_Client	MODBUS RTU client via TCP.
%QX(n+93).1	BIT	Ethernet.NET1. wProtocol. bMODBUS_ETH_Client	MODBUS TCP Client.
%QX(n+93).2	BIT	Ethernet.NET1. wProtocol. bMODBUS_RTU_ETH_Server	MODBUS RTU Server via TCP.
%QX(n+93).3	BIT	Ethernet.NET1. wProtocol. bMODBUS_ETH_Server	MODBUS TCP Server.
%QB(n+95)	STRING (15)	Ethernet.NET1. szIP	NET IP address 1.
%QB(n+111)	STRING (15)	Ethernet.NET1. szMask	NET 1 Subnet Mask.
%QB(n+127)	STRING (15)	Ethernet.NET1. szGateway	NET 1 Gateway Address.
%QB(n+143)	STRING (17)	Ethernet.NET1. szMAC	MAC NET 1 Address.
%QB(n+161)	BYTE ARRAY(4)	Ethernet.NET1. abyIP	NET IP address 1.
%QB(n+165)	BYTE ARRAY(4)	Ethernet.NET1. abyMask	NET 1 Subnet Mask.

altus

Direct representation	Size	AT DG vari-able_Modulo.tDetailed.*	Description
%QB(n+169)	BYTE ARRAY(4)	Ethernet.NET1. abyGateway	NET 1 Gateway Address.
%QB(n+173)	BYTE ARRAY(6)	Ethernet.NET1. abyMAC	MAC NET 1 Address.
%QD(n+179)	DWORD	Ethernet.NET1. dwPacketsSent	Counter of packets sent via NET 1 port (0 to 4294967295).
%QD(n+183)	DWORD	Ethernet.NET1. dwPacketsReceived	Counter of packets received through NET 1 port (0 to 4294967295).
%QD(n+187)	DWORD	Ethernet.NET1. dwBytesSent	Count of bytes sent over NET 1 port (0 to 4294967295).
%QD(n+191)	DWORD	Ethernet.NET1. dwBytesReceived	Count of bytes received through NET 1 port (0 to 4294967295).
%QW(n+195)	WORD	Ethernet.NET1. wTXErrors	Counter of transmission errors via NET 1 port (0 to 65535).
%QW(n+197)	WORD	Ethernet.NET1. wTXFIFOErrors	Error counter in transmit buffer through NET 1 port (0 to 65535).
%QW(n+199)	WORD	Ethernet.NET1. wTXDropErrors	Connection loss counter when transmitting through the NET 1 port (0 to 65535).
%QW(n+201)	WORD	Ethernet.NET1. wTXCollisionErrors	Collision error counter when transmitting via NET 1 port (0 to 65535).
%QW(n+203)	WORD	Ethernet.NET1. wTXCarrierErrors	Transmission error counter on NET 1 port transmission (0 to 65535).
%QW(n+205)	WORD	Ethernet.NET1. wRXErrors	Counter of errors received via NET 1 port (0 to 65535).
%QW(n+207)	WORD	Ethernet.NET1. wRXFIFOErrors	Error counter in the receive buffer via NET 1 port (0 to 65535).
%QW(n+209)	WORD	Ethernet.NET1. wRXDropErrors	Connection loss counter when receiving via NET 1 port (0 to 65535).
%QW(n+211)	WORD	Ethernet.NET1. wRXFrameErrors	Frame error counter on reception via NET 1 port (0 to 65535).
%QW(n+213)	WORD	Ethernet.NET1. wMulticast	Counter of multicast packets through NET 1 port (0 to 65535).

Table 172: NET1 Ethernet Group Detailed Diagnostics

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+219)	BYTE	UserFiles. byMounted	Indicates if the memory used for recording user files is able to receive data.
%QD(n+220)	DWORD	UserFiles. dwFreeSpacekB	Free memory space for user files in Kbytes.
%QD(n+224)	DWORD	UserFiles. dwTotalSizekB	Storage capacity of the memory of user files in Kbytes.

Table 173: UserFiles Detailed Diagnostics Group Description

Note:

User Partition: The user partition is a memory area reserved for the storage of data in the CPU. For example: files with PDF extension, files with DOC extension and other data.

altus

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+229)	BYTE	UserLogs. byMounted	Status of memory in which are inserted the user logs.
%QW(n+230)	WORD	UserLogs. wFreeSpacekB	Free space in the memory of user logs in Kbytes.
%QW(n+232)	WORD	UserLogs. wTotalSizekB	Storage capacity of the memory of user logs in Kbytes.

Table 174: UserLogs Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+235)	BYTE	MemoryCard. byMounted	Status of the Memory Card: 00: Memory card not mounted 01: Memory card inserted and mounted
%QX(n+236).0	BIT	MemoryCard. bMemcardtoCPUEnabled	Protection level of the Memory Card: Data reading of the memory card by the authorized CPU.
%QX(n+236).1	BIT	MemoryCard. bCPUtoMemcardEnabled	Data writing in the memory card by the authorized CPU.
%QD(n+237)	DWORD	MemoryCard. dwFreeSpacekB	Free space in the Memory Card in Kbytes.
%QD(n+241)	DWORD	MemoryCard. dwTotalSizekB	Storage capacity of the Memory Card in Kbytes.

Table 175: MemoryCard Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+245)	BYTE	WHSB. byHotSwapAndStartupStatus	Informs the abnormal situation in the bus which caused the application stop for each mode of hot swapping. See Table 177 for more information.
%QD(n+247)	DWORD ARRAY (32)	WHSB. adwRackIOErrorStatus	Identification of errors in I/O modules, individually. For more information about this diagnostic, see the notes below.
%QD(n+375)	DWORD ARRAY (32)	WHSB. adwModulePresenceStatus	Status of presence of declared I/O modules in buses, individually. For more information about this diagnostic, see the notes below.
%QB(n+503)	BYTE	WHSB. byWHSBBusErrors	Counter of failures in the WHSB bus. This counter is restarted in the energization (0 to 255).

Table 176: WHSB Detailed Diagnostics Group Description

Notes:

Bus modules error diagnostic: Each DWORD from this diagnostic array represents a rack, whose positions are represented by the bits of these DWORDS. So, Bit-0 of the DWORD-0 is equivalent to position zero of the rack with address zero. Each one of these Bits is the result of an OR logic operation between the Incompatible Configuration (bConfigMismatch), absent modules (bAbsentModules), swapped modules (bSwappedModules), module with fatal error (bModuleFatalError) diagnostics and the operational state of the module in a certain position.

Module presence status: Each DWORD from this diagnostic array represents a rack, whose positions are represented by the bits of these DWORDS. So, Bit-0 from DWORD-0 is equivalent to position zero of the rack with address zero. So, if a module is present, this bit will be true. It's important to notice that this diagnostic is valid for all modules, except power supplies, CPUs and non-declared modules, e.g. those that are not in the rack on the respective position (bit remains in false).

Situations in which the Application Stops: The codes for the possible situations in which the application stops can be consulted below:

altus

Code	Enumerable	Description
00	INITIALIZING	This state is presented while other states are not ready.
01	RESET_WATCHDOG	Application in Stop Mode due to hardware watchdog reset or runtime reset, when the option *"Start User Application After a Watchdog Reset"* is unmarked.
02	ABSENT_MODULES_HOT_SWAP_ DISABLED	Application in Stop Mode due to Absent Modules diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
03	CFG_MISMATCH_HOT_SWAP_ DISABLED	Application in Stop Mode due to Configuration Mismatch diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
04	ABSENT_MODULES_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Absent Modules diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
05	CFG_MISMATCH_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Incompatible Configuration diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
06	APPL_STOP_ALLOWED_TO_RUN	Application in Stop Mode and all consistencies executed successfully. The application can be set to Run Mode.
07	APPL_STOP_MODULES_NOT_READY	Application in Stop Mode and all consistencies executed successfully, but the I/O modules are not able to start the system. It is not possible to set the application to Run Mode.
08	APPL_STOP_MODULES_GETTING_ READY_TO_RUN	Application in Stop Mode and all consistencies executed successfully. The I/O modules are being prepared to start the system. It is not possible to set the application to Run Mode.
09	NORMAL_OPERATING_STATE	Application in Run Mode.
10	MODULE_CONSISTENCY_OK	Internal usage.
11	APPL_STOP_DUE_TO_EXCEPTION	Application in Stop Mode due to an exception in the CPU.
12	DUPLICATED_SLOT_HOT_SWAP_ DISABLED	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
13	DUPLICATED_SLOT_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
14	DUPLICATED_SLOT_HOT_SWAP_ ENABLED	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Enabled, without startup consistency"*.
15	NON_DECLARED_MODULE_HOT_ SWAP_STARTUP_CONSISTENCY	Application in Stop Mode due to Non Declared Modules diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"*.
16	NON_DECLARED_MODULE_HOT_ SWAP_DISABLED	Application in Stop Mode due to Non Declared Modules diagnostic being set when the Hot Swap Mode is *"Disabled"*.

Table 177: Codes of the Situations in which the Application Stops

altus

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+504)	BYTE	Application. byCPUState	Informs the operation state of the CPU: 01: All user applications are in Run Mode 03: All user applications is in Stop Mode
%QX(n+505).0	BIT	Application. bForcedIOs	There is one or more forced I/O points.
%QX(n+505).1	BIT	Application. bNetDefinedByWeb	The NX30x0 CPU does not support IP exchange via the System Web Page. Therefore, the diagnosis will always remain FALSE.

Table 178: Application Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QD(n+506)	DWORD	Rack. dwAbsentRacks	Each bit presents a rack identification number, if the bit is TRUE, it means the rack is absent.
%QD(n+510)	DWORD	Rack. dwDuplicatedRacks	Each bit presents a rack identification number, if the bit is TRUE, it means that more than one rack is with the same identification number.
%QD(n+514)	DWORD	Rack. dwNonDeclaredRacks	Each bit presents a rack identification number, if the bit is TRUE, it means there is a rack configured with a non-declared identification number.

Table 179: Rack Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QD(n+520)	DWORD	ApplicationInfo. dwApplicationCRC	32 bits CRC of Application. When the application is modified and sent to the CPU, a new CRC is generated.

Table 180: ApplicationInfo Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Modulo.tDetailed.*	Description
%QX(n+532).0	BIT	SNTP. bServiceEnabled	SNTP Service enabled.
%QB(n+533)	BYTE	SNTP. byActiveTimeServer	Indicates which server is active: 00: None active server. 01: Active Primary Server. 02: Active Secondary Server.
%QW(n+534)	WORD	SNTP. wPrimaryServerDownCount	Counter of times in which the primary server is unavailable (0 to 65535).
%QW(n+536)	WORD	SNTP. wSecondaryServerDownCount	Counter of times in which the secondary server is unavailable (0 to 65535).
%QD(n+538)	DWORD	SNTP. dwRTCTimeUpdatedCount	Counter of times the RTC was updated by the SNTP service (0 to 4294967295).
%QB(n+542)	BYTE	SNTP. byLastUpdateSuccessful	Indicates status of the last update: 00: It was not updated. 01: Last update failed. 02: Last update was successfully.
%QB(n+543)	BYTE	SNTP. byLastUpdateTimeServer	Indicates which server was used in the last update: 00: None update. 01: Primary Server. 02: Secondary Server.

altus

Direct representation	Size	AT Variable DG_Modulo.tDetailed.*	Description
%QB(n+544)	BYTE	SNTP.sLastUpdateTime. byDayOfMonth	Day of the last update of the RTC.
%QB(n+545)	BYTE	SNTP.sLastUpdateTime. byMonth	Month of the last update of the RTC.
%QW(n+546)	WORD	SNTP.sLastUpdateTime. wYear	Year of the last update of the RTC.
%QB(n+548)	BYTE	SNTP.sLastUpdateTime. byHours	Hour of the last update of the RTC.
%QB(n+549)	BYTE	SNTP.sLastUpdateTime. byMinutes	Minute of the last update of the RTC.
%QB(n+550)	BYTE	SNTP.sLastUpdateTime. bReservedAlign	Reserved for alignment.
%QB(n+551)	BYTE	SNTP.sLastUpdateTime. bySeconds	Second of the last update of RTC.
%QW(n+552)	WORD	SNTP.sLastUpdateTime. wMilliseconds	Millisecond of the last update of RTC.

Table 181: SNTP Detailed Diagnostics Group Description

6.1.5. 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.

6.1.5.1. GetTaskInfo

This function returns the task information of a specific application.

Figure 136: 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 182: GetTaskInfo Input Parameters

altus

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 183: 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 stTaskInfo;
TaskInfo : stTaskInfo;
Info : 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. Graphic Display

The graphic display available in this product has an important tool for the process control, as through it is possible to recognize possible error conditions, active components or diagnostics presence. Besides, all diagnostics including the I/O modules are presented to the user through the graphic display. For further information regarding the diagnostic key utilization and its visualization see One Touch Diag section.

On figure below, it is possible to observe the available characters in this product graphic display and, next, its respective meanings.
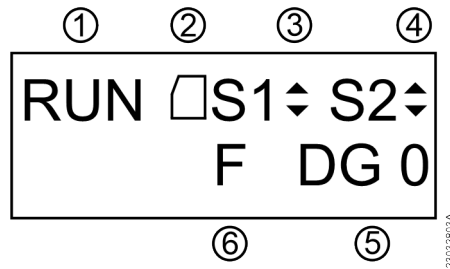
altus

Figure 137: CPU Status Screen

Legend:

- 1. Indication of the CPU status operation. In case the CPU application is running, the state is RUN. In case the CPU application is stopped, the state is STOP and, when is stopped in an application depuration mark, the state is BRKP. For further details, see CPU Operating States section.
- 2. Memory Card presence indication. Further details regarding its installation see Memory Card Installation section.
- 3. COM 1 traffic indication. The up arrow (▲) indicates data transmission and the down arrow (▼) indicates data reception. For further information regarding the COM 1 interface see Serial Interfaces section.
- 4. COM 2 traffic indication. The up arrow (▲) indicates data transmission and the down arrow (▼) indicates data reception. For further information regarding the COM 2 interface see Serial Interfaces section.
- 5. Indication of the CPU active diagnostics quantity. In case the number shown is different than 0 (zero), there are active diagnostics in the CPU. For further details regarding their visualization on the CPU graphic display, through diagnostic key, see One Touch Diag section.
- 6. Forced variables in the CPU indication. In case the "F" character is shown in the graphic display, a variable is being forced by the user, whether symbolic, direct representation or AT. For further information regarding variable forcing see Writing and Forcing Variables section.

Besides the characters described above, Nexto CPUs can present some messages on the graphic display, correspondent to a process which is being executed at the moment.

The table below present the messages and their respective descriptions:

Message	Description
FORMATTING...	Indicates the CPU is formatting the memory card.
FORMATTING ERROR	Indicates that an error occurred while formatting the memory card by the CPU.
WRONG FORMAT	Indicates that the memory card format is incorrect.
INCORRECT PASSWORD	Indicates the typed password is different from the configured password.
TRANSFERRING...	Indicates the project is being transferred.
TRANSFERRING ERROR	Indicates there is been an error in the project transference caused by some problem in the memory card or its removal during transference.
TRANSFERRING COMPLETE	Indicates the transference has been executed successfully.
TRANSFERRING TIMEOUT	Indicates a time-out has been occurred (communication time expired) during the project transference.
CPU TYPE MISMATCH	Indicates the CPU model is different from the one configured in the project within the memory card.
VERSION MISMATCH	Indicates the CPU version is different from the one configured in the project within the memory card.
APPLICATION CORRUPTED	Indicates the application within the memory card is corrupted.
APPLICATION NOT FOUND	Indicates there is no application in the memory card to be transferred to the CPU.
CRC NOT FOUND	Indicates that the CRC application does not exist.
MCF FILE NOT FOUND	Indicates there is no MCF file in the memory card.

Message	Description
NO TAG	There is no configured tag for the CPU in the MasterTool IEC XE.
NO DESC	There is no configured description for the CPU in the MasterTool IEC XE.
MSG. ERROR	Indicates that there are error(s) on diagnostics message(s) of the requested module(s).
SIGNATURE MISSING	Indicates the product presented an unexpected problem. Get in contact with Altus Technical Support sector.
APP. ERROR RESTARTING	Indicates that occurred an error in the application and the Runtime is restarting the application.
APP. NOT LOADED	Indicates that the runtime will not load the application.
LOADING APP.	Indicates that the runtime will load the application.
WRONG SLOT	Indicates that the CPU is in an incorrect position in the rack.
FATAL ERROR	Indicates that there are serious problems in the CPU startup such as CPU partitions that were not properly mounted. Please, contact Altus customer support.
HW-SW MISMATCH	Indicates that the CPU hardware and software are not compatible because the product presented an unexpected problem. Please, contact Altus customer support.
UPDATING FIRMWARE	Indicates the firmware is being updated in the CPU.
RECEIVING FIRMWARE	Indicates the updating file is being transferred to the CPU.
UPDATED	Shows the firmware version updated in the CPU.
UPDATE ERROR	Indicates an error has occurred during the CPU firmware updating, caused by communication failure or configuration problems.
REBOOTING SYSTEM...	Indicates the CPU is being restarted for the updating to have effect.

Table 184: Other Messages of the Graphic Display

6.3. System Log

The *System Log* is an available feature in the MasterTool IEC XE programmer. It is an important tool for process control, as it makes it possible to find events on CPU that may indicate error conditions, presence of active components or active diagnostics. Such events can be viewed in chronological order with a resolution of milliseconds, with a storage capacity of up to one thousand log entries stored in the CPU internal memory, that can't be removed.

In order to access these Logs, just go to the *Device Tree* and double-click on *Device*, then go to the *Log* tab, where hundreds of operations can be seen, such as: task max cycles, user access, online change, application download and upload, application synchronization between CPUs, firmware update between another events and actions.

In order to view the *Logs*, just need to be connected to a CPU (selected Active Path) and click on ⟳ . When this button is pressed the Logs are displayed and updated instantly. When the button is not being pressed the Logs will be hold in the screen, it means, these button has two stages, one hold the logs state being updated and in the state the updating is disabled. To no longer show the Logs, press ✕ .

It is possible to filter the Logs in 4 different types: warning(s), error(s), exception(s) and information.

Another way to filter the messages displayed to the user is to select the component desired to view.

The Log tab's *Time Stamp* is shown by MasterTool after information provided by the device (CPU). MasterTool can display the Time Stamp in local time (computer's time) or UTC, if *UTC time* checkbox is marked.

> **ATTENTION**
>
> If the device's time or time zone parameter are incorrect, the Time Stamp shown in Master-Tool also won't be correct.

For further information about the System Logs please check the MasterTool IEC XE User Manual – MU299609 and the RTC Clock and Time Synchronization subsection of this manual.

ATTENTION

The system logs of the CPUs, starting in firmware version 1.4.0.33 (Nexto) and 1.14.36.0 (Xtorm), are reloaded in the cases of a restart of the CPU or a reboot of the Runtime System, that is, it will be possible to check the older logs when one of these situations occurs.

6.4.   Not Loading the Application at Startup

If necessary, the user can choose to not load an existing application on the CPU during its startup. Just power the CPU with the diagnostics button pressed and keep it pressed for until the message "APP. NOT LOADED" is shown in the screen. If a login attempt is made, MasterTool IEC XE software will indicate that there is no application on the CPU. For reloading the application, the CPU must be reset or a new application download must be done.

6.5.   Power Supply Failure

The Nexto Series Power Supply (NX8000) has a failure detection system according to the levels defined in its technical features (see Power Supply 30 W 24 Vdc Technical Characteristics - CE114200). There are two ways to diagnose a failure:

1. In case the NX8000 power supply is on with voltage lower than the required minimum limit, a power supply failure diagnostic is generated, which is recognized by the CPU and the message "POWER FAILURE" is shown on the display. When the supply is within the established limits, the CPU recognizes it and automatically is restarted with the user application. The diagnostic will still be active to show to the user that the last initialization suffered a power supply failure.
2. In case the NX8000 has a voltage drop to an inferior value than the minimum required limit and it returns to a higher value within 10 ms, the power supply failure is not recognized by the CPU and the diagnostic is not generated as the system remains intact during this time. But if the voltage drop takes longer than 10 ms, the "POWER FAILURE" message is shown on the CPU screen and the diagnostic is activated.
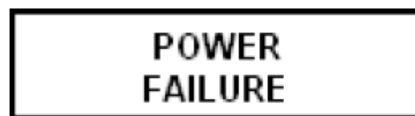

Figure 138: Power Supply Failure Message

The user can change the value of the variable attributed to the power supply failure to FALSE during the application execution, facilitating the verification and treatment of this diagnostic.

The POWER FAILURE diagnostic is already mapped in a specific memory region, defined as CPU Detailed Diagnostic. This way it is just to use it as global variable. The variable name is described in the detailed diagnostic list in the Diagnostics via Variables section.

6.6.   Common Problems

If, at power on the CPU, it does not work, the following items must be verified:

- Is the room temperature within the device supported range?
- Is the rack power supply being fed with the correct voltage?
- Is the power supply module inserted on the far left in the rack (observing the rack by the front view) followed by the Nexto Series CPU?
- Are there network devices, as hubs, switches or routers, powered, interconnected, configured and working properly?
- Is the Ethernet network cable properly connected to the Nexto CPU NET 1 or NET 2 port and to the network device?
- Is the Nexto Series CPU on, in execution mode (Run) and with no diagnostics related to hardware?

If the Nexto CPU indicates the execution mode (Run) but it does not respond to the requested communications, whether through MasterTool IEC XE or protocols, the following items must be verified:

- Is the CPU Ethernet parameters configuration correct?
- Is the respective communication protocol correctly configured in the CPU?
- Are the variables which enable the MODBUS relations properly enabled?

If no problem has been identified, consult the Altus Technical Support.

6.7. Troubleshooting

The table below shows the symptoms of some problems with their possible causes and solutions. If the problem persists, consult the Altus Technical Support.

Symptom	Possible Cause	Solution
Does not power on	Lack of power supply or incorrectly powered.	Verify if the CPU is connected properly in the rack. Power off and take off all modules from the bus, but the power supply and the CPU. Power on the bus and verify the power supply functioning, the external and the one in the rack. Verify if the supply voltage gets to the Nexto power supply contacts and if is correctly polarized.
CPU Screen shows the message WRONG SLOT	CPU in a wrong position.	The CPU must be placed in slots 2 and 3 of rack 0. Put it in the correct slots. CPUs must be placed in slots 2 and 3 of rack 0. Put it in the correct slots.
Does not communicate	Bad contact or bad configuration.	Verify every communication cable connection. Verify the serial and Ethernet interfaces configuration in the MasterTool IEC XE software.
Does not recognize the memory card	Bad connection or not mounted.	Verify if the memory card is properly connected in the compartment. Verify if the memory card was put in the right side, as indicated on the CPU frontal panel. Verify if the memory card wasn't unmounted through MS button, placed on the frontal panel, visualizing the indication on the CPU graphic display.

Table 185: Troubleshooting

6.8. 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.
- Bus tightness and cleanness every six months.
- For further information, see Nexto Series Manual - MU214600.