User Manual
Nexto Series CPUs
NX3004, NX3005, NX3010, NX3020, NX3030

Rev. J 07/2017

Doc. Code.: MU214605

www.altus.com.br

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

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. This can even put human lives at risk.

The user should consider the possible consequences of the defects and should provide additional external installations for safety reasons. This concern is higher when in initial commissioning and testing.

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

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use.

The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user trainings and improve experience with the products and their features.

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

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

Altus does not acknowledge any other guarantee, directly or implied, mainly when end customers are dealing with third-party suppliers.

The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing form. Altus is not responsible for supplying information about its equipment without formal request.

These products use EtherCAT® technology (www.ethercat.org).

COPYRIGHTS

OPEN SOURCE SOFTWARE NOTICE

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

Table of Contents

1. Introduction

Nexto Series CPUs were designed to fulfill several customers' demands in a variety of applications present on industrial automation and process control. Due to its compact and rugged body, excellent performance and fast I/Os update time provided by a unique high-speed communication bus, Nexto Series CPUs are the best choice for the most demanding control applications. In complex applications, where reliability, availability and remote I/O operation is required, Nexto Series CPUs are also a great choice due to its different redundant topologies and bus expansion possibilities.

Nexto Series CPUs provide innovative and unique enhanced diagnostics services. They take the user to a whole new diagnostics experience. By using a switch located on top of the module and a built-in compact graphical LCD display, the user has direct access to extensive information regarding I/Os, fieldbus interfaces and many other modules on the application. In addition, it has user register system services and ease at debbuging and tasks management, reducing the application cost and installation time.

Finally, Nexto Series CPUs feature several communication interfaces such as serial and Ethernet ports, a memory card interface and full IEC 61131-3 programming languages.

Figure 1-1. CPU NX3030

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 32 bits 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.

Figure 1-2. Nexto Series –Overview

Innovative Features

Nexto Series brings to the user several innovations in utilization, supervision and system maintenance. These features were developed focusing a new experience in industrial automation. The list below shows some new features that users will find in the Nexto Series.

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.

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, such as 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. For example: Nexto CPUs NX3010, NX3020 and NX3030 can execute a sum, multiplication and subtraction instruction in less than 15 ns for integer type values and in less than 23 ns for real type values. Nexto CPUs are able to perform 1,000 PID loops in less than 5 ms.

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.

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 http://www.altus.com.br/site_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 NX1001 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
CS114000	Serie Nexto – Especificaciones y Configuraciones	Spanish
CE114700	Nexto Series Backplane Racks Technical Characteristics	English
CT114700	Características Técnicas dos Bastidores da Série Nexto	Portuguese
CS114700	Características Técnicas de los Bastidores de la Serie Nexto	Spanish
CE114900	NX4010 Redundancy Link Module Technical Characteristics	English
CT114900	Características Técnicas do Módulo de Redundância NX4010	Portuguese
CS114900	Características Técnicas del Módulo de Redundancia NX4010	Spanish
CE114902	NX5001 PROFIBUS-DP Master Technical Characteristics	English
CT114902	Características Técnicas do Mestre PROFIBUS DP NX5001	Portuguese
CS114902	Especificaciones y Configuraciones Maestro PROFIBUS-DP NX5001	Spanish
CE114903	Ethernet Module NX5000 Technical Characteristics	English
CT114903	Características Técnicas do Módulo Ethernet NX5000	Portuguese
CS114903	Especificaciones y Configuraciones Modulo Ethernet NX5000	Spanish
CT112500	Características Técnicas do Painel de Controle de Redundância PX2612	Portuguese
MU214600	Nexto Series User Manual	English
MU214000	Manual de Utilização Série Nexto	Portuguese
MU214300	Manual del Usuario Serie Nexto	Spanish
MU214605	Nexto Series CPUs User Manual	English
MU214100	Manual de Utilização CPUs Série Nexto	Portuguese
MU214305	Manual del Usuario CPUs Serie Nexto	Spanish
MU299609	MasterTool IEC XE User Manual	English
MU299048	Manual de Utilização MasterTool IEC XE	Portuguese
MU299800	Manual del Usuario MasterTool IEC XE	Spanish
MP399609	MasterTool IEC XE Programming Manual	English
MP399048	Manual de Programação MasterTool IEC XE	Portuguese
MP399800	Manual de Programación MasterTool IEC XE	Spanish
MU214601	NX5001 PROFIBUS DP Master User Manual	English
MU214001	Manual de Utilização Mestre PROFIBUS DP NX5001	Portuguese
MU214301	Manual del Usuario Maestro PROFIBUS DP NX5001	Spanish
MU219000	Ponto Series Utilization Manual	English
MU209000	Manual de Utilização da Série Ponto	Portuguese
MU209508	Manual de Utilização Cabeça PROFIBUS PO5063V1 e Cabeça Redundante PROFIBUS PO5063V5	Portuguese
MU219511	PO5064 PROFIBUS Head and PO5065 Redundant PROFIBUS Head Utilization Manual	English

MU209511	Manual de Utilização Cabeça PROFIBUS PO5064 e Cabeça Redundante PROFIBUS PO5065	Portuguese
MU209020	Manual de Utilização Rede HART sobre PROFIBUS	Portuguese

Table 1-1. Related Documents

Visual Inspection

Before resuming the installation process, it is advised to carefully visually inspect the equipments, 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.

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 http://www.altus.com.br/site_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 of the used equipments and the installed system configuration
- The CPU serial number
- The equipment revision and the executive software version, written on the tag fixed on the product side
- CPU operation mode information, acquired through MasterTool IEC XE
- The application software content, acquired through MasterTool IEC XE
- Used program version

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 NX3004, NX3005, NX3010, NX3020 and NX3030.

Panels and Connections

The following figure shows the CPU NX3030 front panel.

Figure 2-1. CPU NX3030

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 2-1 shows the LEDs description. For further information regarding the LEDs status and meaning, see Diagnostics via LED chapter.

LED	Description
DG	Diagnostics LED
WD	Watchdog LED

Table 2-1. LEDs Description

Nexto Series CPUs has two switches available to the user. Table 2-2 shows the description of these switches. For further information regarding the diagnostics switch, see chapters One Touch Diag and

CPU's Informative and Configuration Menu. For further information regarding the MS switch, see Configuration - 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 2-2. Keys Description

On the frontal panel the connection interfaces of Nexto Series CPUs are available. These interfaces are: Ethernet communication, serial communication and memory card interface. Table 2-3 presents a brief description of these interfaces.

Interfaces	Available on Models	Description
NET 1	NX3004 NX3005 NX3010 NX3020 NX3030	RJ45 communication connector standard 10/100Base-TX. Allows the point to point or network communication through open protocols MODBUS TCP client and server. MODBUS RTU via TCP client and server. For further utilization information, see Ethernet Interfaces Configuration section.
NET 2	NX3020 NX3030	RJ45 communication connector pattern 10/100Base-TX. Allows the point to point or network communication through open protocols MODBUS TCP client and server. MODBUS RTU via TCP client and server. For further utilization information, see Ethernet Interfaces Configuration section.
COM 1	NX3010 NX3020 NX3030	DB9 female connector for RS-232 pattern communication. Allows the point to point or network communication through open protocols MODBUS RTU slave or MODBUS RTU master. For further utilization information, see Serial Interfaces Configuration section.
COM1	NX3004 NX3005	DB9 female connector for RS-485 and RS-422 standard. Allows point-to-point or network communication through open protocols, MODBUS RTU slave or MODBUS RTU master. For further utilization information, see Serial Interfaces Configuration section.
COM 2	NX3010 NX3020 NX3030	DB9 female connector for RS-485 and RS-422 standard. Allows point-to-point or network communication through open protocols, MODBUS RTU slave or MODBUS RTU master. For further utilization information, see Serial Interfaces Configuration section.
Power Supply	NX3004 NX3005	6-terminal connector with fixing. It powers Nexto series modules connected at the same bus, providing 15 W of power.
MEMORY SLOT	NX3010 NX3020 NX3030	Memory card slot. Allows the use of a memory card for different types of data storage such as: user logs, Web pages, project documentation and files. For further utilization information, see Configuration-Memory Card section.

Table 2-3. Connection Interfaces

General Features

Common General Features

	NX3004, NX3005, NX3010, NX3020, NX3030
Backplane rack occupation	2 sequential slots
Programming languages	Instruction List (IL) Structured Text (ST) Ladder Diagram (LD) Sequential Function Chart (SFC) Function Block Diagram (FBD) Continuous Function Chart (CFC)
Tasks	Cyclic (periodic) Event (software event) External (hardware event) Freewheeling (continuous) Status (software event)
Online changes	Yes
Hot swap support	Yes
Bus expansion redundancy support	Yes
Serial interfaces	NX3004/NX3005 – COM 1: 1 x RS-485 / RS-422 NX3010/NX3020/NX3030 – COM 1: 1 x RS-232C NX3010/NX3020/NX3030 – COM 2: 1 x RS-485 / RS-422
MODBUS Protocol	RTU (COM 1 and COM 2) master and slave TCP (NET 1 and NET 2) client and server RTU via TCP (NET1 and NET2) client and server
OPC protocol	Yes
EtherCAT protocol	Yes, NX3020 and NX3030
SNMP Protocol	Yes, v1, v2c and v3 versions
Real time clock (RTC)	Yes Resolution of 1 ms and maximum variance of 2 s per day
Watchdog	Yes
Status and diagnostic indication	Graphic display, LEDs, web pages and CPU's internal memory
One Touch Diag (OTD)	Yes
Electronic Tag on Display (ETD)	Yes
Isolation	
Logic to protective earth ⏚	1250 Vac / 1 minute
Logic to Ethernet interfaces	1500 Vac / 1 minute
Logic to serial port (COM 2)	1000 Vac / 1 minute
Logic to serial port NX3004/NX3005 (COM 1)	1000 Vac / 1 minute
Ethernet interfaces to protective earth ⏚	1250 Vac / 1 minute
Ethernet interfaces to serial port (COM 2)	1500 Vac / 1 minute
Ethernet interfaces to serial port on the NX3004/NX3005 (COM 1)	1500 Vac/ 1 minute
Ethernet interface to Ethernet interface	1500 Vac / 1 minute
Serial port (COM 2) to protective earth ⏚	1250 Vac / 1 minute
Serial port N3004/NX3005 (COM 1) to protective earth ⏚	1000 Vac / 1 minute
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

Standards	IEC 61131-2 IEC 61131-3 CE, Electromagnetic Compatibility (EMC) and Low-Voltage Directive (LVD).
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 2-4. Common Features

Notes:

Tasks: Task is an object used to call POUs. A Task can be triggered by period, events or can run in freewheeling mode. Each task can call one or more POUs.

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 ºC. At the maximum product temperature, the retention time is reduced to 10 days.

Isolation: The term logic is used to refer to the internal circuits such as processor, memory and bus interfaces.

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

Specific Features

	NX3004	NX3005	NX3010	NX3020	NX3030
Addressable input variables memory (%I)	32 Kbytes	32 Kbytes	32 Kbytes	64 Kbytes	96 Kbytes
Addressable output variables memory (%Q)	32 Kbytes	32 Kbytes	32 Kbytes	64 Kbytes	96 Kbytes
Addressable variables memory (%M)	16 Kbytes	16 Kbytes	16 Kbytes	32 Kbytes	64 Kbytes
Symbolic variables memory	2 Mbytes	3 Mbytes	4 Mbytes	5 Mbytes	6 Mbytes
Maximum amount of memory configurable as retentive or persistent	7.5 Kbytes	7,5 Kbytes	64 Kbytes	112 Kbytes	112 Kbytes
Total redundant data memory	-	-	-	-	736 Kbytes
Addressable input variables memory (%I)	-	-	-	-	80 Kbytes
Addressable output variables memory (%Q)	-	-	-	-	80 Kbytes
Addressable variables memory (%M)	-	-	-	-	64 Kbytes
Symbolic variables memory	-	-	-	-	512 Kbytes
Program memory	3 Mbytes	3 Mbytes	4 Mbytes	6 Mbytes	8 Mbytes
Source code memory (backup)	32 Mbytes	40 Mbytes	40 Mbytes	80 Mbytes	120 Mbytes
User files memory	16 Mbytes	16 Mbytes	16 Mbytes	32 Mbytes	32 Mbytes
Maximum number of tasks	16	16	16	24	32
Maximum number of expansion bus	1	4	8	24	24
Maximum number of I/O modules on the bus	32	64	128	128	128
Ethernet TCP/IP local interface	1	1	1	2	2
Maximum number of additional Ethernet TCP/IP interface modules	0	1	0	2	6
Ethernet TCP/IP interface redundancy support	No	No	No	Yes	Yes
Maximum number of PROFIBUS-DP network (using master modules PROFIBUS-DP)	1	1	1	4	4
PROFIBUS-DP network redundancy support	No	No	No	Yes	Yes

Redundancy support (half-clusters)	No	No	No	No	Yes
Event oriented data reporting (SOE) **Protocol** **Maximum event queue size**	No - -	No - -	No - -	Yes DNP3 1000	Yes DNP3 1000
Clock synchronization (SNTP)	Yes	Yes	Yes	Yes	Yes
Web pages development (available trough the HTTP protocol)	No	Yes	No	No	No
Integrated power supply	Yes	Yes	No	No	No
Current consumption from backplane rack power supply		-	800 mA	1000 mA	1000 mA
Power dissipation	4 W	4 W	4 W	5 W	5 W

Table 2-5. Specific Features

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.

The Nexto Series NX3030 CPU allows defining an area of redundant variables inserted inside of the addressable output variables %Q. The subset of addressable output variables memories are part of the total size of available memory.

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 stored. Symbolic variables are IEC variables created in POUs and GVLs during application development, not addressed directly in memory. Symbolic variables can be defined as retain or persistent. In these cases, it will be used the memory area of retain symbolic variables memory or persistent symbolic variables memory respectively.

Persistent and 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 2-6. Besides the persistent area size declared in the Table 2-5, are reserved these 44 bytes to store information about the persistent variables (not available for use).

The Table 2-6 shows the behavior of retentive and persistent variables for different situations in which "-" means the value is lost and "X" means the value is kept.

Command	Symbolic variable	Retain variable	Persistent variable
Reset warm/Power-on/off cycle	-	X	X
Reset cold	-	-	X
Reset origin	-	-	-
Remove CPU or Power Supply from the rack while energized.	-	-	-
Download	-	-	X
Online change	X	X	X
Reboot PLC	-	X	X
Clean All	-	-	X

Table 2-6. Post-command Variable Behavior

In versions 1.5.0.21 and lower for NX3004 and 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 2-7, 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.

	NX3004	NX3010	NX3020	NX3030
Retentive addressable output variables memory (%Q)	2 Kbytes	8 Kbytes	16 Kbytes	16 Kbytes
Persistent addressable output variables memory (%Q)	2 Kbytes	8 Kbytes	16 Kbytes	48 Kbytes
Retentive symbolic variables memory	2 Kbytes	32 Kbytes	48 Kbytes	32 Kbytes
Persistent symbolic variables memory	1,5 Kbytes	16 Kbytes	32 Kbytes	16 Kbytes

Table 2-7. 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.

Total redundant data memory: Redundant data memory is the maximum memory area that can be used as redundant memory between two redundant CPUs. This value is not a different memory, note that the sum of all redundant variables (addressable input variable, addressable output variable, addressable variable, symbolic variable, retain symbolic variable, persistent symbolic variable) must be less than or equal to the available redundant data memory.

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, being the total area the sum of "program memory" and "source code 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 MemoryUser Files Memory.

Maximum number of tasks: The maximum number of tasks defined for each CPU model, and among different project profiles, is better detailed in the chapter Maximum Number of Tasks.

Redundancy support (half-clusters): The software version 1.1.0.0 or onwards/product revision AB or onwards supports redundancy of NX3030 CPUs.

Event oriented data reporting (SOE): The data types are found in the DNP3 Device Profile.

Maximum number of PROFIBUS-DP network: From MasterTool IEC XE version 1.22 and onwards, 4 PROFIBUS-DP networks are supported for NX3020 and NX3030 CPUs. Previous versions support 2 PROFIBUS-DP networks. The limit of PROFIBUS-DP masters is 4, which means that only 2 redundant networks can be used.

Serial Interfaces

COM 1(NX3010/NX3020/NX3030)

	NX3010, NX3020, NX3030
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
Protocols	Master/ Slave MODBUS RTU Open protocol

Table 2-8. COM 1 Serial Interface Features

COM 1 (NX3004/NX3005) and COM 2 (NX3010/NX3020/NX3030)

	NX3004, NX3005, NX3010, NX3020, NX3030
Connector	Shielded female DB9
Physical interface	RS-422 or RS-485 (depends on the cable choice)
Communication direction	RS-422: full duplex RS-485: half duplex
RS-422 maximum transceivers	11 (1 transmitter and 10 receivers)
RS-485 maximum transceivers	32
Termination	Yes (optional via cable selection)
Baud rate	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200
Protocols	Master/ Slave MODBUS RTU Open protocol

Table 2-9. COM 2 Serial Interface Features

Notes:

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

RS-422 Maximum Transceivers: It is the maximum number of transceivers that can be used on a same bus.

RS-485 Maximum Transceivers: It is the maximum number of transceivers that can be used on a same bus.

Ethernet Interfaces

NET 1

	NX3004, NX3005, NX3010, NX3020, NX3030
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	10/100BASE-TX
Data link layer	LLC (logical link control)
Network layer	IP (internet protocol)
Transport layer	TCP (Transmission Control Protocol) and UDP (User Datagram Protocol)
Application layer	Client/ Server MODBUS TCP Client/ Server MODBUS RTU via TCP HTTP (Web server) MasterTool IEC XE programming protocol DNP3 (event oriented data reporting) SNTP (Clock synchronism) EtherCAT OPC SNMP (Ethernet Network Management) Ethernet/IP Scanner/Adapter
Diagnostics	LEDs - green (speed), yellow (link/activity)

Table 2-10. Ethernet NET 1 Interface Features

Note:

Application Layer: The DNP3 and EtherCAT protocols are not available for CPUs NX3004, NX3005 and NX3010.

NET 2

	NX3020, NX3030
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	10/100BASE-TX
Data link layer	LLC (logical link control)
Network layer	IP (internet protocol)
Transport layer	TCP (Transmission Control Protocol) and UDP (User Datagram Protocol)
Application layer	Client/ Server MODBUS TCP Client/ Server MODBUS RTU via TCP HTTP (Web server) MasterTool IEC XE programming protocol DNP3 (event oriented data reporting) SNTP (clock synchronism) EtherCAT OPC SNMP (Ethernet Network Management) Ethernet/IP Scanner/Adapter
Diagnostics	LEDs - green (speed), yellow (link/activity)

Table 2-11. Ethernet NET 2 Interface Features

Power Supply

	NX3004, NX3005
Nominal input voltage	24 Vdc
Maximum output power	15 W
Maximum output current	3 A
Input voltage	19.2 to 30 Vdc
Maximum input current (inrush)	30 A
Maximum input current	1.4 A
Maximum input voltage interrupt time	10 ms @ 24 Vdc
Isolation	
Input to output	1000 Vac / 1 minute
Input to protective earth ⏚	1500 Vac / 1 minute
Input do functional earth ⏚	1000 Vac / 1 minute
Wire size	0.5 mm²
Polarity inversion protection	Yes
Internal auto recovery fuse	Yes
Output short-circuit protection	Yes
Overcurrent protection	Yes

Table 2-12. Power Supply Features

Note:

Maximum output power: Using modules I/O NextoJet, it is possible to extend and come using 20 W of power output. See Application Note NAP152 to meet the restrictions to use this limit.Memory Card Interface

The memory card can be used for different data to be stored such as user logs, web pages, project documentation and source files. More information about how to use the memory card interface can be found Memory Card section.

	NX3010, NX3020, NX3030
Maximum Capacity	8 Gbytes
Minimum Capacity	2 Gbytes
Type	miniSD
File System	FAT32
Remove card safely	Yes, by pressing MS switch

Table 2-13. 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.

Compatibility with Other Products

There are some incompatibilities between the Nexto Series CPUs and the MasterTool IEC XE. See the following table to find out which version of MasterTool IEC XE should be used.

Nexto Series CPU Software Version			MasterTool IEC XE compatible version
NX3004	NX3005	NX3010, NX3020, NX3030	
-	-	1.2.0.9 or lower	1.00 to 1.28
-	-	1.2.1.0 to 1.2.4.0	1.29 to 1.40
-	-	1.3.0.20	1.40 to 1.41
-	-	1.4.0.33	2.00
1.5.0.18	-	1.5.0.10 to 1.5.0.16	2.01 to 2.02
1.5.1.2	-	1.5.1.3	2.03 to 2.06
1.6.0.0	1.6.0.0	1.6.0.0	2.07 or above

Table 2-14. Compatibility with other products

Note:

Compatibility between versions: some features are available only from a particular version. To further information, see the Manual MasterTool IEC XE – MU299609 to verify the availability of some characteristic in a specific product version.

Performance

The Nexto Series CPU performance relies on:

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

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 “Main Task” 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 “Main Task” and, when using the communication modules, as the Ethernet NX5000 module, for instance, the time addition to the “Main Task” may be up to 25% of the execution average time.

Time for Instructions Execution

Table 2-15 presents the necessary execution time for different instructions in Nexto Series CPUs:

Instruction	Language	Variables	Instruction Times (us)
1000 Contacts	LD	BOOL	6
1000 Divisions	ST	INT	43
		REAL	81
	IL	INT	43
		REAL	81
	LD	INT	43
		REAL	81
1000 Multiplications	ST	INT	15
		REAL	23
	IL	INT	15
		REAL	23
	LD	INT	15
		REAL	23
1000 Sums	ST	INT	15
		REAL	23
	IL	INT	15
		REAL	23
	LD	INT	15
		REAL	23

Table 2-15. Instruction Times

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.

Interval Time

The CPU interval time of every task depends on the application software which can be set from 5 to 750ms.

Physical Dimensions

NX3004/NX3005

Dimensions in mm.

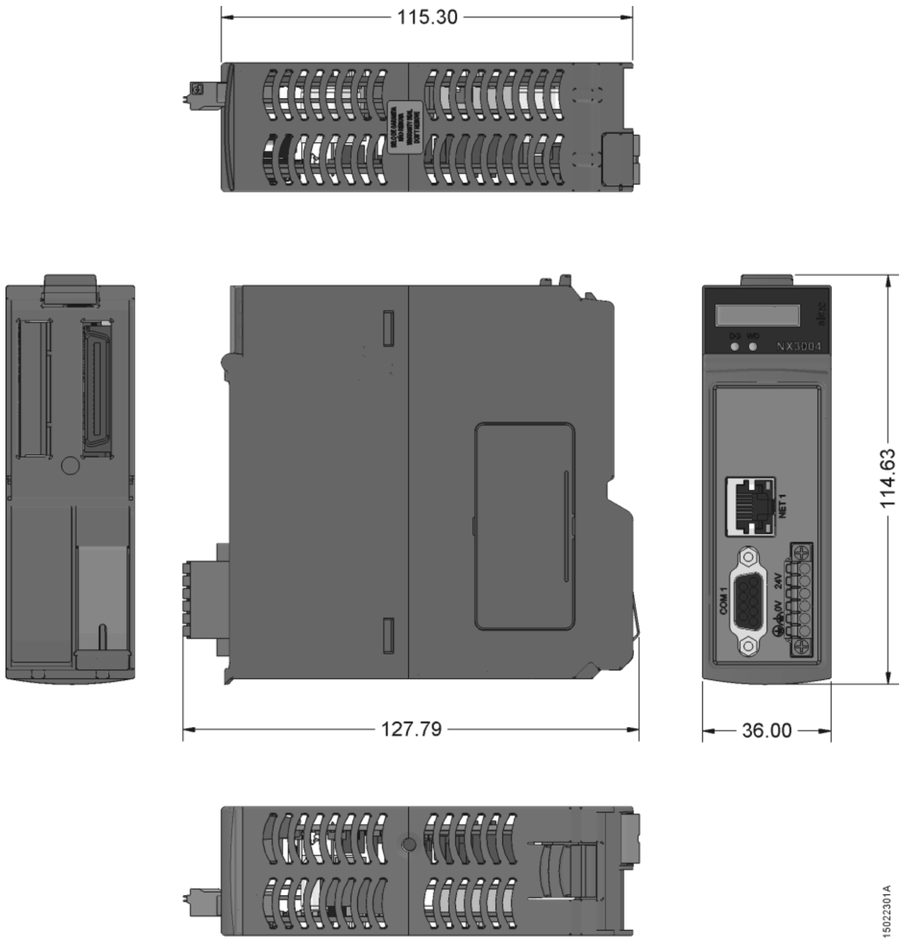


Figure 2-2. NX3004 and NX3005 CPU Physical Dimensions

NX3010/NX3020/NX3030

Dimensions in mm.

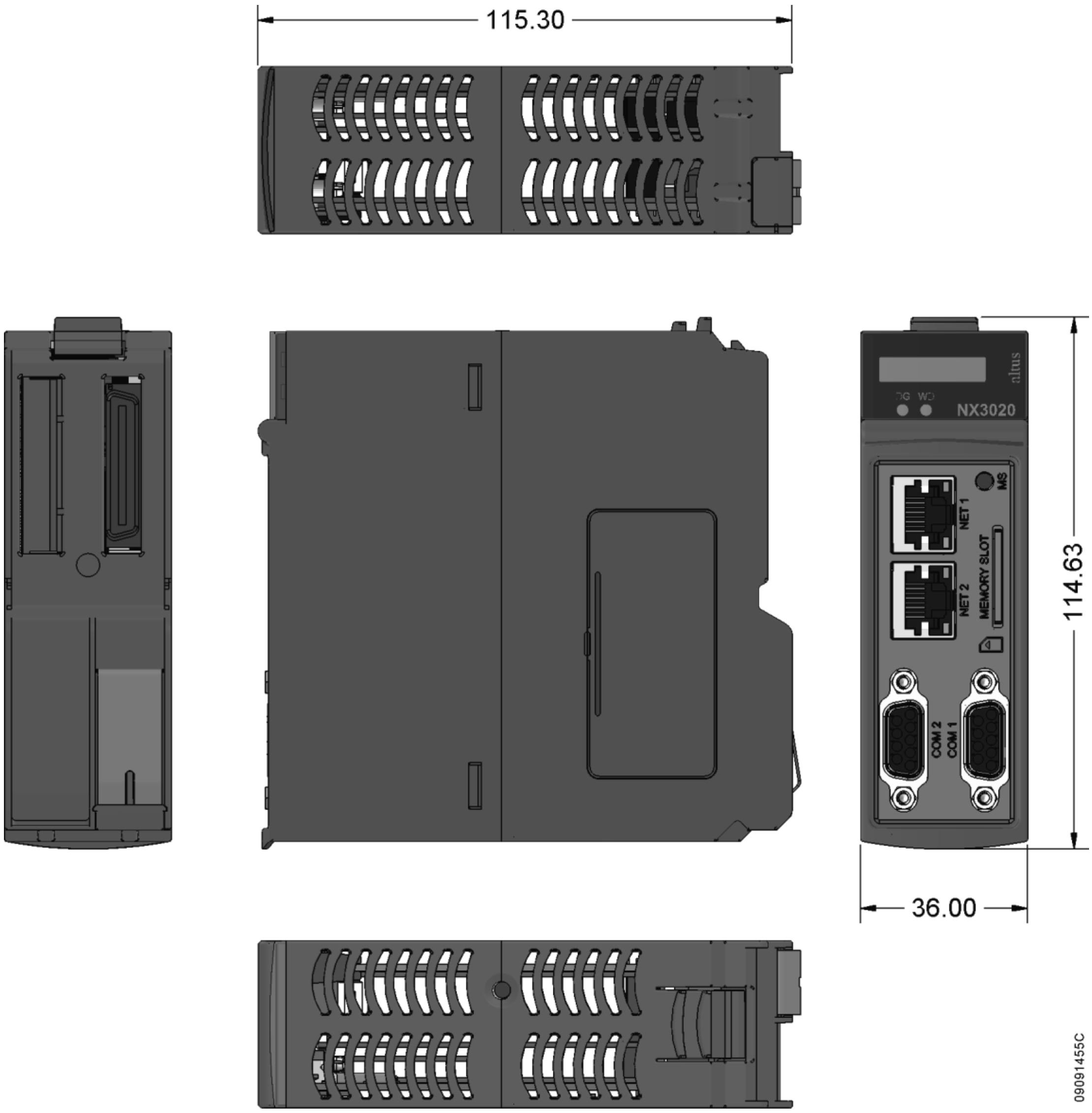


Figure 2-3. NX3010, NX3020 and NX3030 CPU Physical Dimensions

Purchase Data

Integrant Items

The product package has the following items:

- NX3004, NX3005, NX3010, NX3020 or NX3030 module
- 6-terminal connector with fixing (only NX3004 and NX3005)
- Installation guide

Product Code

The following code should be used to purchase the product:

Code	Description
NX3004	CPU, 1 Ethernet port, 1 serial channel, remote rack expansion support and power supply integrated
NX3005	CPU, 1 Ethernet port, 1 serial channel, remote rack expansion support, power supply integrated and user web pages support
NX3010	High-speed CPU, 1 Ethernet port, 2 serial channels, memory card interface and remote rack expansion support
NX3020	High-speed CPU, 2 Ethernet ports, 2 serial channels, memory card interface and remote rack expansion support
NX3030	High-speed CPU, 2 Ethernet ports, 2 serial channels, memory card interface, remote rack expansion and redundancy support

Table 2-16. Nexto Series CPUs Models

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-2301	RS-485 network cable (up to 1000 meters)
AL-2306	RS-485 network cable (up to 500 meters)
AL-2319	RJ45-RJ45 Cable
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
NX9101	8 Gb Memory Card, MicroSD with MiniSD Adapter
NX9202	RJ45-RJ45 2 m Cable
NX9205	RJ45-RJ45 5 m Cable
NX9210	RJ45-RJ45 10 m Cable

Table 2-17. Related Products

Notes:

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

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-2301: Two shielded twisted pairs cable without connectors, used for networks based on RS-485 interface, with 1000 meters of maximum length.

AL-2306: Two shielded twisted pairs cable without connectors, used for networks based on RS-485 interface, with 500 meters of maximum length.

AL-2319: Two RJ45 connectors for programming the CPUs of the Nexto Series and Ethernet point-to-point with another device with Ethernet interface communication.

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 1 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 o 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 Serie CPUs and Altus products of the AL Series.

AL-1762: RS-232C standard cable with two DB9 male connections 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.

NX9202/NX9205/NX9210: Cables used 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.

Mechanical Installation

NX3004 and NX3005

The CPUs NX3004 and NX3005 must be inserted in the backplane rack position 0. It requires two sequential positions, this means that it uses postions 0 and 1 of the rack.

NX3010, NX3020 and NX3030

Nexto Series CPUs (NX3010, NX3020 and NX3030) 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 Nexto Series User Manual – MU214605.

Electrical Installation

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

NX3004 and NX3005

The Figure 3-1 shows the Nexto Series CPUs electric diagram installed in a Nexto Series backplane rack.

The connectors placement depicted are merely illustrative.

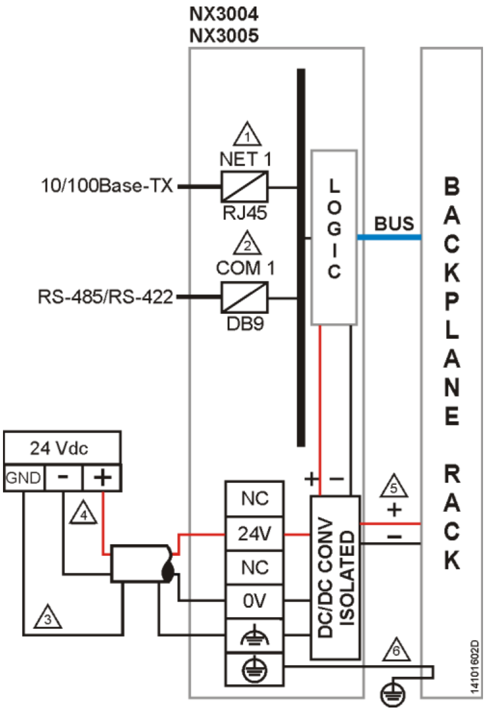


Figure 3-1. NX3004 and NX3005 CPU Electric Diagram

Diagram Notes:

1. Ethernet interface pattern 10/100Base-TX for programming, debugging and MODBUS TCP network connection or other protocols.
2. Serial interface pattern RS-485/RS-422 for MODBUS RTU network connection or other protocols. The physical interface choice depends on the cable used.
3. The grounding from the external power source is connected to the terminal ⏚. Use 0,5 mm² cables
4. The power supply is connected to terminal 0 V. Use 0.5 mm ² cables
5. The power supply is connected to terminal 24 V. Use 0.5 mm ² cables
6. The power supply feeds the internal circuit directly
7. Local data bus
8. The module feeds the other modules of the Nexto Series through rack connection
9. The module is grounded through Nexto Series backplane rack

NX3010, NX3020 and NX3030

The NX3010, NX3020 and NX3030 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 3-3 shows the Nexto Series CPUs electric diagram installed in a Nexto Series backplane rack.

The connectors placement depicted are merely illustrative.

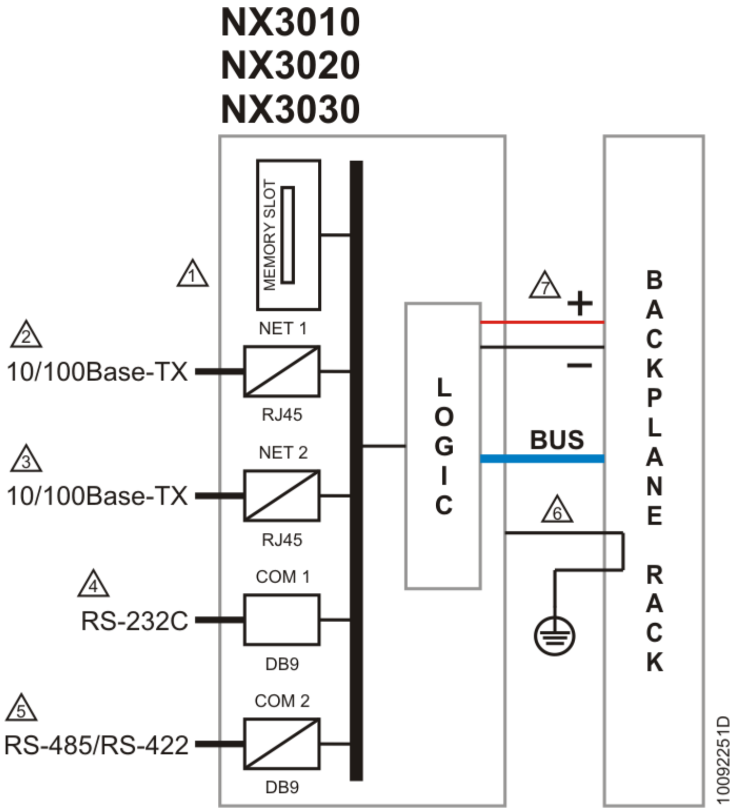


Figure 3-2. NX3010, NX3020, NX3030 CPUs Electric Diagram

Diagram Notes:

10. Memory card interface.
11. Ethernet interface pattern 10/100Base-TX for programming, debugging and MODBUS TCP network connection or other protocols.

12. Ethernet interface pattern 10/100Base-TX for MODBUS TCP network connection or other protocols (only for NX3020 and NX3030).
13. Serial interface pattern RS-232C for MODBUS RTU network connection or other protocols.
14. Serial interface pattern RS-485/RS-422 for MODBUS RTU network connection or other protocols. The physical interface choice depends on the cable used.
15. The module is grounded through Nexto Series backplane rack.
16. The power supply comes from the backplane rack connection. There is no need for external connections.

Ethernet Network Connection

The NET 1 and NET 2 (only for NX3020 and NX3030) 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.

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
Subnet Mask	255.255.255.0
Gateway Address	192.168.15.253

Table 3-1. 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 Configuration – CPU's Informative and Configuration Menu chapter.

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

The NET 2 Ethernet interface, available only in NX3020 and NX3030 CPUs, is used only for Ethernet communication and comes with the following default parameters configuration:

	NET 2
IP Address	192.168.16.1
Subnet Mask	255.255.255.0
Gateway Address	192.168.16.253

Table 3-2. Default Parameters Configuration for Ethernet NET 2 Interface

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

The NET 2 interface parameters can be modified via MasterTool IEC XE. For further information regarding configuration and parameters modifications, see Configuration – Ethernet Interfaces Configuration chapter.

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.

Network Cable Installation

Nexto Series CPUs Ethernet ports, identified on the panel by NET 1 and NET 2 (NX3020 and NX3030), have pattern pin outs 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 Technical Description –Ethernet Interfaces Configuration Figure 3-3 and Table 3-3 present the RJ-45 Nexto CPU female connector, with the identification and description of the valid pin out for 10Base-T and 100Base-TX physical levels.

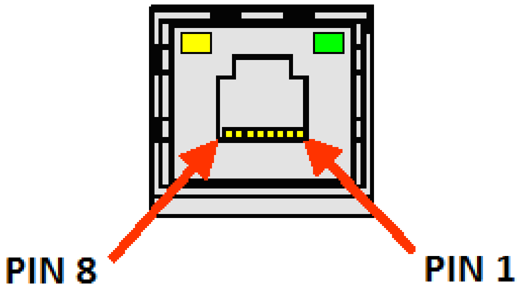


Figure 3-3. RJ45 Nexto CPU Female Connector

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 3-3. RJ45 Nexto CPU Female Connector Pin out

The interface can be connected in a communication network through a hub or switch, or straight from the communication equipment. In this last case, due to 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.

Serial Network Connection RS-232

The NX3010, NX3020 and NX3030 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.

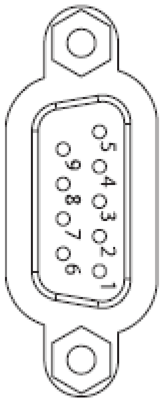


Figure 3-4. DB9 Female Connector, NX3010, NX3020 and NX3030 CPUs (COM 1)

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 3-4. DB9 Female Connector Pin Layout, NX3010, NX3020 and NX3030 CPUs (COM 1)

RS-232C Communication

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

Serial Network Connection RS-485/422

The NX3004/NX3005-COM 1 and NX3010/NX3020/NX3030-COM 2 isolated communication interfaces allows 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.

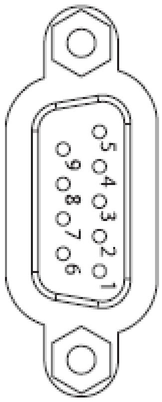


Figure 3-5. DB9 Female Connector, NX3004/NX3005 CPU (COM 1) and NX3010/NX3020/NX3030 CPU (COM 2)

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 3-5. DB9 Female Connector Pin Layout, COM 1 (NX3004/NX3005) and COM 2 (NX3010/NX3020/NX3030)

RS-485 Communication without termination

In order to connect in a RS-485 network with no termination in COM 1 (NX3004 or NX3005) or COM 2 (NX3010, NX3020 or NX3030) interface, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on Table 3-6.

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 3-6. RS-485 Connections with no Termination

The Figure 3-6 diagram indicates how the AL-1763 connection terminals should be connected in the device terminals.

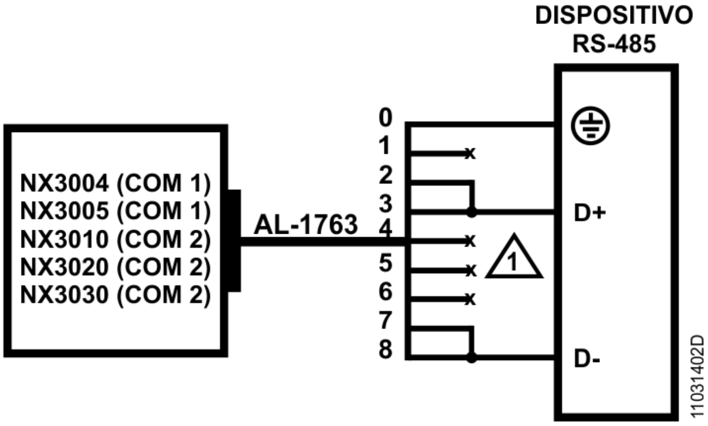


Figure 3-6. RS-485 Connections with no Termination Diagram

Diagram Note:

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

RS-485 Communication with Internal Termination

In order to connect in a RS-485 network using the internal termination in COM 1 (NX3004 or NX3005) or COM 2 (NX3010, NX3020 or NX3030) interface, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on Table 3-7.

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 3-7. RS-485 Connections with Internal Termination

PS.: The internal termination available in COM 1 (NX3004 and NX3005) or COM 2 (NX3010, NX3020 and NX3030) is safe state type in open mode.

The Figure 3-7 diagram indicates how the AL-1763 connection terminals should be connected in the device terminals.

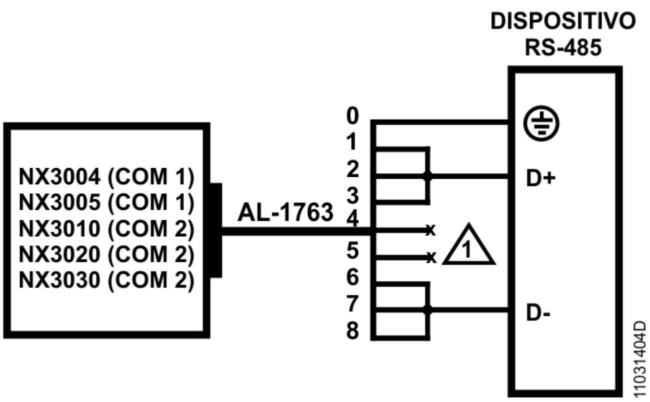


Figure 3-7. RS-485 Connections with Internal Termination Diagram

Diagram Note:

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

RS-485 Communication with External Termination

In order to connect to a RS-485 network using a COM 1 (NX3004 or NX3005) or COM 2 (NX3010, NX3020 or NX3030) interface external termination, the AL-1763 cable identified terminals must be connected in the respective device terminals according to the Table 3-8.

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 3-8.RS-485 Connections with External Termination

The Figure 3-8 diagram indicates how the AL-1763 connection terminals should be connected in the device terminals.

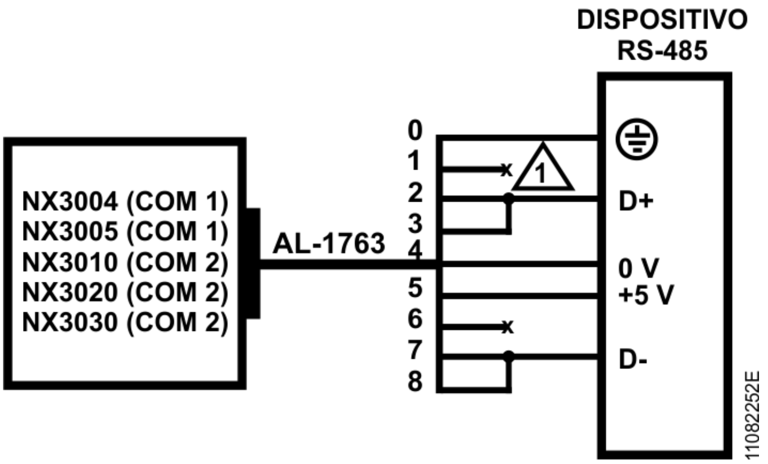


Figure 3-8. COM 2 with RS-485 Connections Diagram with External Termination

Diagram Note:

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

Example of Connection of a RS-485 Network with External Termination and Master Redundancy

Figure 3-9 below shows an example of RS-485 network connection with external termination, using two Nexto NX3030 CPUs with half-cluster redundancy as master.

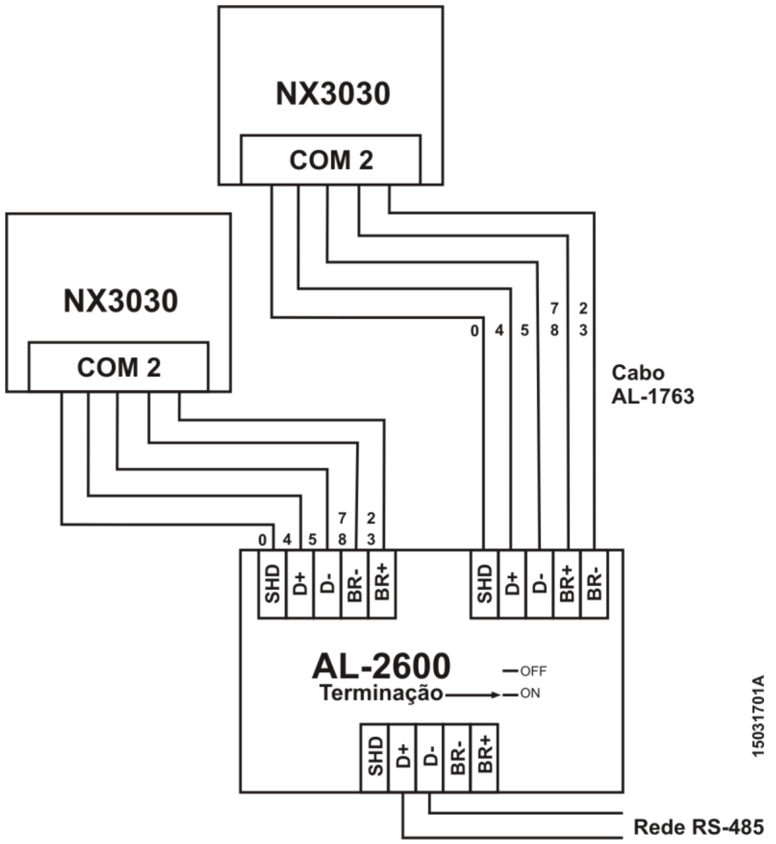


Figure 3-9. Connection Diagram of a RS-485 Network with External Termination and Master Redundancy

RS-422 Communication without Termination

In order to connect in a RS-422 network with no termination in COM 1 (NX3004 and NX3005) or COM 2 (NX3010, NX3020 and NX3030) interface, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on Table 3-9.

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 3-9. RS-422 Connections with no Termination

The Figure 3-10 diagram indicates how the AL-1763 connection terminals should be connected in the device terminals.

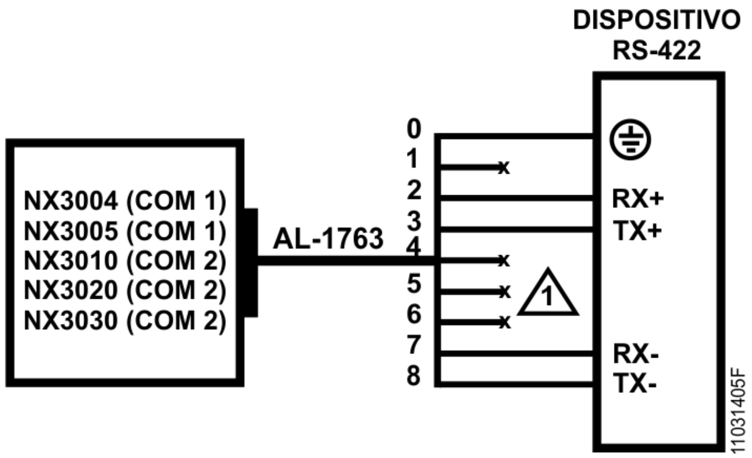


Figure 3-10. RS-422 Connections with no Termination Diagram

Diagram Note:

The not connected terminal must be insulated so they don’t make contact with each other.

RS-422 Communication with Internal Termination

In order to connect in a RS-422 network using the internal termination in COM 1 (NX3004 and NX3005) or COM 2 (NX3010, NX3020 and NX3030) interface, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on Table 3-10.

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 3-10. RS- 422 Connections with Internal Termination

PS.: The internal termination available in COM 2 is secure state in open mode.

The Figure 3-11 diagram indicates how the AL-1763 connection terminals should be connected in the device terminals.

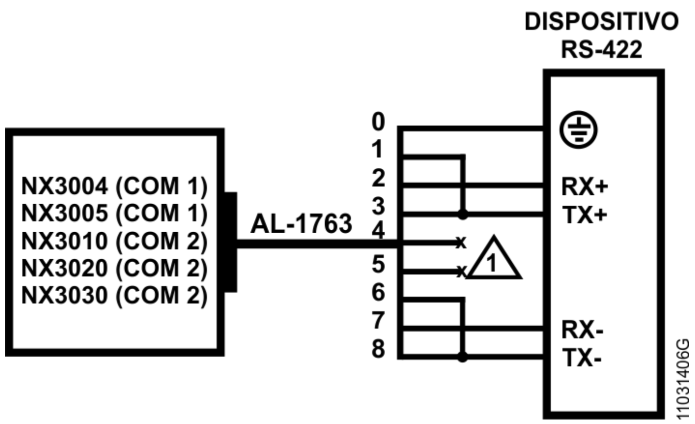


Figure 3-11. RS-422 Connections with Termination Diagram

Diagram Note:

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

RS-422 Communication with External Termination

In order to connect in a RS-422 network using the COM 1 (NX3004 or NX3005) or COM 2 (NX3010, NX3020 or NX3030) interface external termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on Table 3-11.

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 3-11. RS-422 Connections with External Termination

The Figure 3-12 diagram indicates how the AL-1763 connection terminals should be connected in the device terminals.

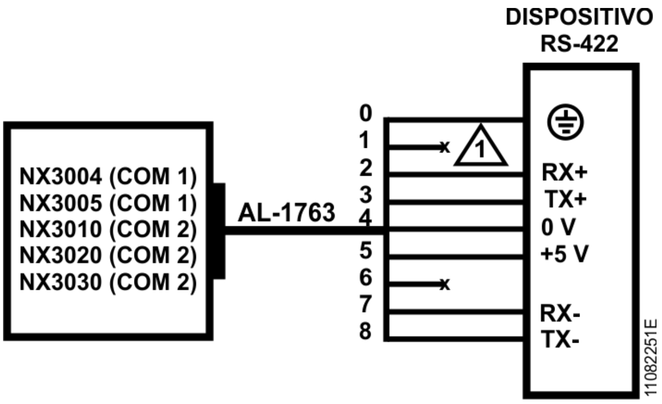


Figure 3-12. RS-422 Connections with External Termination Diagram

Diagram Note:

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

RS-422 Network Example

Figure 3-13 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.

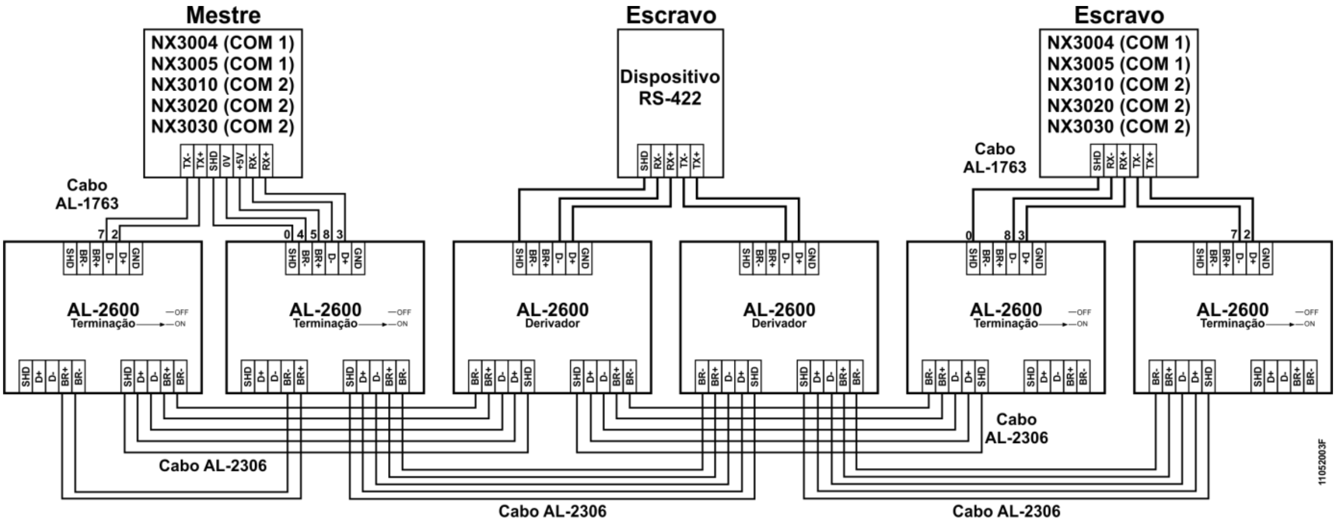


Figure 3-13. RS-422 Network Example

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.

Memory Card Installation

This section presents how to insert the memory card into the models NX3010, NX3020 and NX3030 Nexto Series CPUs . For further information see Configuration – Memory Card chapter.

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 3-14.

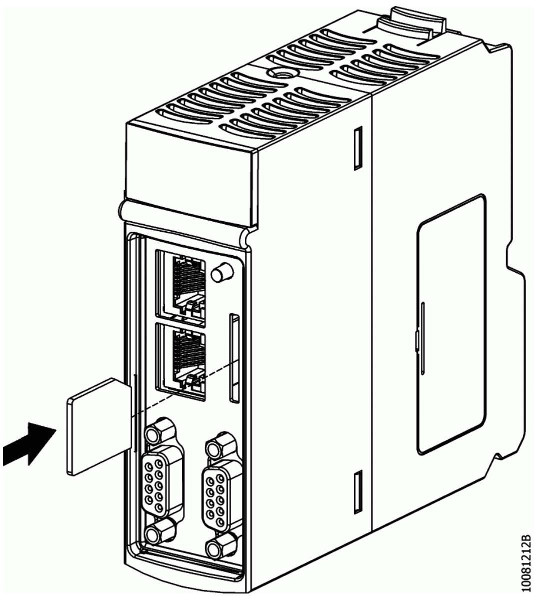


Figure 3-14. Memory Card Insertion in the CPU

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 3-15. At this moment the card will be loose.

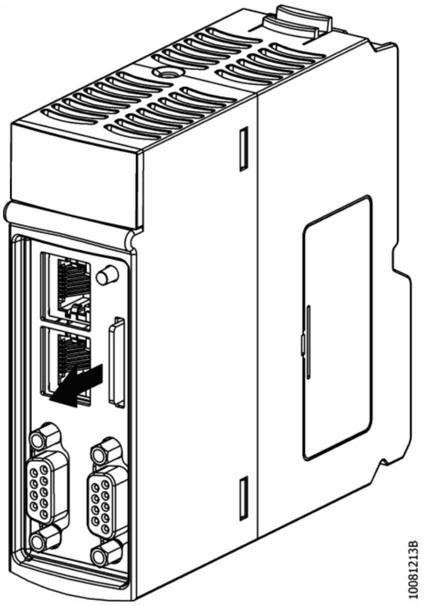


Figure 3-15. Memory Card Withdrawal

Architecture Installation

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.

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 http://www.altus.com.br/site_en/. For further information about the step by step to installation, consult MasterTool IEC XE User Manual MT8500 – MU299609.

4. 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, also known as applicative.

CPU Configuration

General Parameters

The parameters related below are part of the CPU configuration inserted in the application. Each item must be correctly revised for the project perfect 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.

Configuration	Description	Default	Options
	Diagnostics Area (%Q)		
%Q Initial Address	CPU diagnostics initial address (%Q)	Automatically allocated in the project creation	NX3004: 0 to 32210 NX3005: 0 to 32210 NX3010: 0 to 32210 NX3020: 0 to 64843 NX3030: 0 to 97611
Size	Diagnostics area size in bytes	NX3004: 558 NX3004: 558 NX3010: 558 NX3020: 693 NX3030: 693	It is not possible to change the size of the CPU diagnostics area
	Retain Area (%Q)		
%Q Initial Address	Retentive data memory initial address(%Q)	NX3004: 4096 NX3005: 4096 NX3010: 4096 NX3020: 4096 NX3030: 4096	NX3004: 0 to 32767 less the memory size of retentive data NX3005: 0 to 32767 less the memory size of retentive data NX3010: 0 to 32767 – less the memory size of retentive data NX3020: 0 to 65535 – less the memory size of retentive data NX3030: 0 to 98303 less the memory size of retentive data
Size	Retentive data memory size in bytes	NX3004: 7680 NX3005: 7680 NX3010: 32768 NX3020: 65536 NX3030: 98304	NX3004: 0 to 7680 NX3005: 0 to 7680 NX3010: 0 to 32768 NX3020: 0 to 65536 NX3030: 0 to 98304
	Persistent Area (%Q)		
%Q Initial Address	Persistent data memory initial address (%Q)	NX3004: 12288 NX3005: 12288 NX3010: 12288 NX3020: 20480 NX3030: 20480 NX3010: 8192 NX3020: 16384	NX3004: 0 to 32767 less the memory size of retentive data NX3005: 0 to 32767 less the memory size of retentive data NX3010: 0 to 32767 – less the memory size of retentive data NX3020: 0 to 65534 – less the memory size of retentive data NX3030: 0 to 98302 – less the memory size of retentive data
Size	Persistent data memory size in bytes	NX3004: 7680 NX3005: 7680	NX3004: 0 to 7680 NX3005: 0 to 7680

		NX3010: 32768 NX3020: 65536 NX3030: 98304	NX3010: 0 to 32768 NX3020: 0 to 65536 NX3030: 0 to 98304
	CPU Parameters		
Start User Application After a Watchdog Reset	When enabled starts the user application after the hardware watchdog reset or through the Runtime restart, but keeps the diagnostics indication via LED WD and via variables	Disable	Enable Disable
Hot Swap Mode	Hot module change	Enable, without consistency in the start	- Disable, for declared modules only - Disabled - Enable, with consistency in the start only for declared modules - Enable, with consistency in the start - Enabled, without consistency in the start
	TCP/IP Parameters		
Initial Time-out (x100 ms)	Indicates how long after the first transmission of a message, it has to be retransmitted, assuming it was not received by the destination device. Every retransmission the time-out is doubled.	2	1 to 75
ACK Delay (x10 ms)	Delay time for a confirmation command sending	10	0 to 100
	Project Parameters		
Consist retain and persistent area in %Q	Configuration to consist the addressable persistent and retentive memories	Marked	- Marked: It consists the addressable persistent and retentive memories - Unmarked: It doesn't consist the addressable persistent and retentive memories
Enable I/O update per task	Configuration to update the inputs and outputs within the tasks that they are used.	Unmarked	- Marked: The inputs and outputs are updated within the tasks in which they are used. - Unmarked: The inputs and outputs are updated only by the MainTask.
Enable retain and persistent variables in Function Blocks	Configuration to allow the use of retain and persistent variables on Function Blocks	Unmarked	- Marked: allows the use of retain and persistent variables on Function Blocks. - Unmarked: If this is done with this option unmarked, it may occur an exception error on startup.

Table 4-1. CPU Configuration

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.

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.

On the hot swap, the related system behavior modifies itself following the configuration table defined by the user which represents the options below, as described on Table 4-1:

- Disable, for declared modules only
- Disabled
- Enabled, with consistency in the start only for declared modules
- Enabled, with startup consistency
- Enabled, without consistency in the start

Therefore, the user can choose the behavior the system must assume in abnormal bus situations and when the CPU is in Run Mode. Table 4-2 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 mal functioning module is not responding to CPU - Some bus position is malfunctioning.

Table 4-2. 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 Consistency in the Start Only for Declared Modules do not take into consideration the modules that are in this condition.

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 4-2) happens. The LED DG starts to blink 4x (according to Table 4-3). 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 Warm Reset or a Cold Reset (it can be done through the Communication menu of the MasterTool IEC XE). 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 commands Warm, Cold and Origin can be done by MasterTool IEC XE in the Communication menu.

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

Hot Swap Disabled

This setting does not allow any abnormal situation in the bus (as shown in Table 4-2) 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 4-3). 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 commands Warm, Cold and Reset Origin can be done by MasterTool IEC XE in the Communication menu.

Hot Swap Enabled with Consistency in the Start Only for Declared Modules

"Start" 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 4-2) during the start. In affirmative case, the CPU gets in Stop Mode and the LED DG starts to blink 4x (according to Table 4-3).

Afterwards, in order to set the CPU in Run mode, further to fix what caused the abnormal situation, it is necessary to execute a Warm or Cold Reset command, which can be done by the MasterTool IEC XE (Communication 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 Table 4-2, 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 4-3). This is considered a start situation. - This is the most advised option because guarantee the system integrity on its initialization and allows the modules change with a working system.

Hot Swap Enabled with Startup Consistency

This setting checks whether there has been any abnormal situation in the bus (as shown in Table 4-2) during the start, 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 4-3). 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 commands Warm, Cold and Reset Origin can be done by MasterTool IEC XE in the Communication menu.

Hot Swap Enabled without Consistency in the Start

Allows the system to start working even if a module is in an abnormal bus situation (as described on Table 4-2). The abnormal situations are reported via diagnostics during and after the start.

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.

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 LED DG starts to blink 4x (according to Table 4-3), due to any abnormal bus situation (as described on Table 4-2), 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 4-3 presents the bus conditions and the Nexto CPU LED DG operation state. For further information regarding the diagnostics LEDs states, see Diagnostics via LED chapter.

Condition	Enabled with Startup Consistency	Enabled, with Consistency in the Start Only for Declared Modules	Enabled without Consistency in the Start	Disabled	Disabled for declared modules only
Non declared module	LED DG: 2x Application: Run	LED DG: 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run
Non declare module (start 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
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
Absent module (start 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
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
Incompatible configuration (start 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
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
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

Table 4-3. Hot Swap and Conditions Relations

Note:

Enabled, without startup consistency: When this hot-swap mode is configured, in normal situations when there's a 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.

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 Configuration-CPU Configuration-General Parameters. 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 MainPrg
VAR RETAIN

   byVariavelRetentiva_01 AT %QB4096 : BYTE;
END_VAR
VAR PERSISTENT

   byVariavelPersistente_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 (as described at Configuration-CPU Configuration-General Parameters, configuration *Consist retain and persistent area in %Q*), 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:

```
PROGRAM MainPrg
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 1.5.0.22 for NX3004 and 1.5.1.1 for NX3010, NX3020 and NX3030, 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 Table 2-5 in Specific Features.

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.

If, for example, an NX3004 CPU that has 7680 Bytes of retentive and persistent memory is used and it's configured 1000 retentive Bytes and 1000 directly addressable (%Q) output Bytes and, still, the variables in the code below are declared, the total retentive and persistent memory used is going to be 2004 Bytes, leaving 5676 Bytes for free usage.

```
VAR_GLOBAL PERSISTENT RETAIN
   wVariavelSimbolicaPersistenteGlobal_01 : WORD;
END_VAR

VAR_GLOBAL RETAIN
   wVariavelSimbolicaRetentivaGlobal_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.

TCP Configurations

Some of the advanced configurations affect the Nexto Series CPUs supported protocols, as they are connected to the TCP network layer, as following:

- Initial Time-out
- ACK Delay

The CPU Nexto, before it responds to any request and as any other Ethernet equipment which uses the TCP transport layer, demands a communication door opening, in other words, the connection establishment.

The Interface connections quantity is limited and simply does not establish any other connection after its limit is reached. This can cause problems for the established connections in the server mode, as the connections closing depends on the other equipment, the client.

The TCP transport layer, responsible for the messages deliver from the origin to destiny, has some parameters with time-outs, very common in general communication protocols. Such parameters are intended to recover the communication after the failures are determined. The user must be aware with the time-outs configuration, as it some conflicts might occur with the values configured inside the application layer. As the TCP configuration is a reference for every instances configured, the time will be valid if it is smaller than the configured inside a protocol:

- Initial Time-out: indicates how long, after the first message transmission, the message must be retransmitted, assuming it has not been received by the destiny device. At each retransmission the time-out is doubled. The number of transmission tries is connected to the communication time-out configured inside the protocol. It will be the maximum time before it gives up the message delivering, when the transmission failure is concretized. In addition, it is important to stress that there's a maximum quantity of attempts for the Nexto Series CPUs. This number is set in five attempts before the connection is set up and in three attempts after that. Every time it reaches the maximum number of retries, the communication attempt process restarted. See Protocols Configuration section further details regarding time-out parameters utilization as they can be different depending on the situation. It is important to stress this parameter is only used in the connection setting up, after that it is used statistics from the last communications to calculate the new time-out.

Example of the initial time-out and the communication time-out parameters inside the MODBUS TCP Server, considering a not received acknowledge, for the following values: 300 for initial time-out (300 ms) and 3000 for communication time-out (3000 ms):

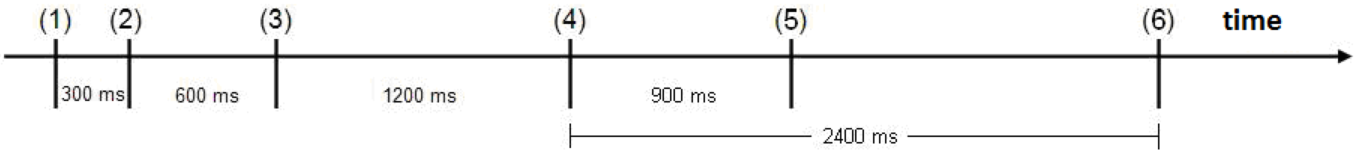


Figure 4-1. Initial Time-out and Communication Time-out

Legend:

17. Message transmission instant.
18. First attempt for message transmitting, after initial time-out.
19. Second attempt for message transmission, after two times the initial time-out.
20. Third try for message transmission, after two times the latter time-out.
21. Quit of message transmission and failure indication, after the communication time-out exceeds (total time until the given up: 300 + 600 + 1200 + 900 = 3000 ms).
22. It would be the fourth message transmission attempt, after two times the latter time-out but the communication time-out configured inside the protocol was exceeded and the failure was indicated.

- ACK sending delay: defines the maximum time waited by the interface for the TCP ACK transmitting. This ACK is responsible for the message receiving conformation, in case of MODBUS, by the destiny device. The set of this field decreases the amount of messages circling through the network. This mechanism is explained below:
  - All request messages, sent by a client to a server, need to be confirmed by the server through an ACK message transmitted to the client.
  - All response messages, sent by a client to a server, need to be confirmed by the server through an ACK message transmitted to the client.

- If one of the parts does not receive the ACK message, within the time defined by the TCP time-out, the message will be retransmitted by the origin address (see TCP number of tries parameter)
- The ACK message does not need to be exclusive. The ACK needed to be sent by the server to the client, at the moment it receives a request, can be included in the same answer message, and the ACK needed to be sent by the client to the server, at the request moment, can be included in the same message including the next request.

The following pictures depict the difference between the sending of an immediate and a calibrated ACK:

ATTENTION;
The NX3004 and NX3005 CPUs have a slightly different behavior. They don't consider the value configured on the parameter Initial Time out, they consider only the communication Time out. Even so, the value configured for communication Time out is used when it's inferior to 3 seconds and it's never going to be doubled on retries. But when the communication time out parameter is superior to 3 seconds, they ignore this value and consist the initial value of 3 seconds and double each retry.

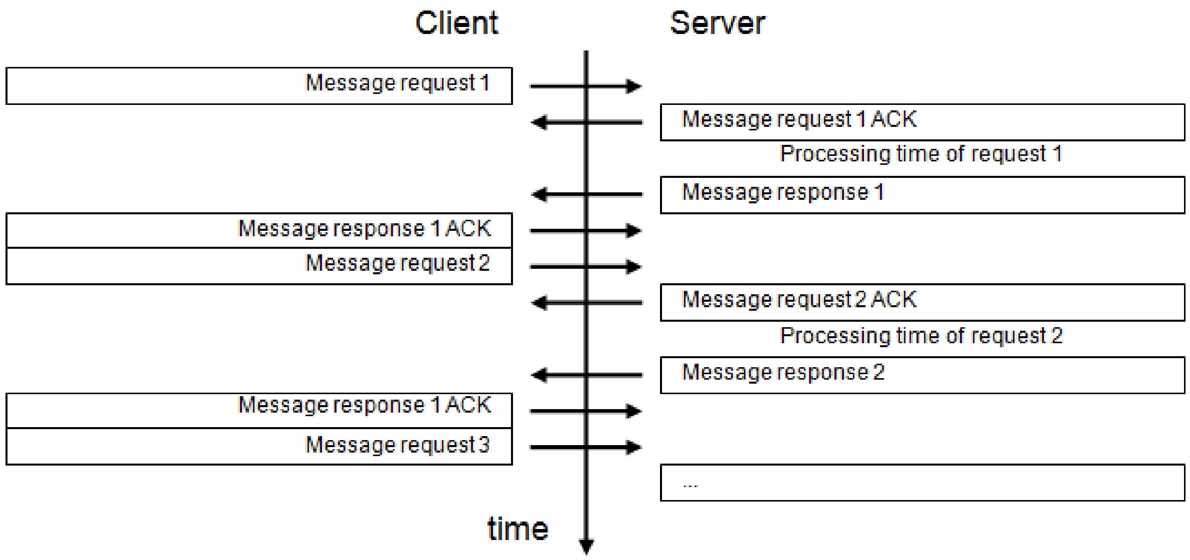


Figure 4-2. Example of an Immediate ACK Sending (=0)

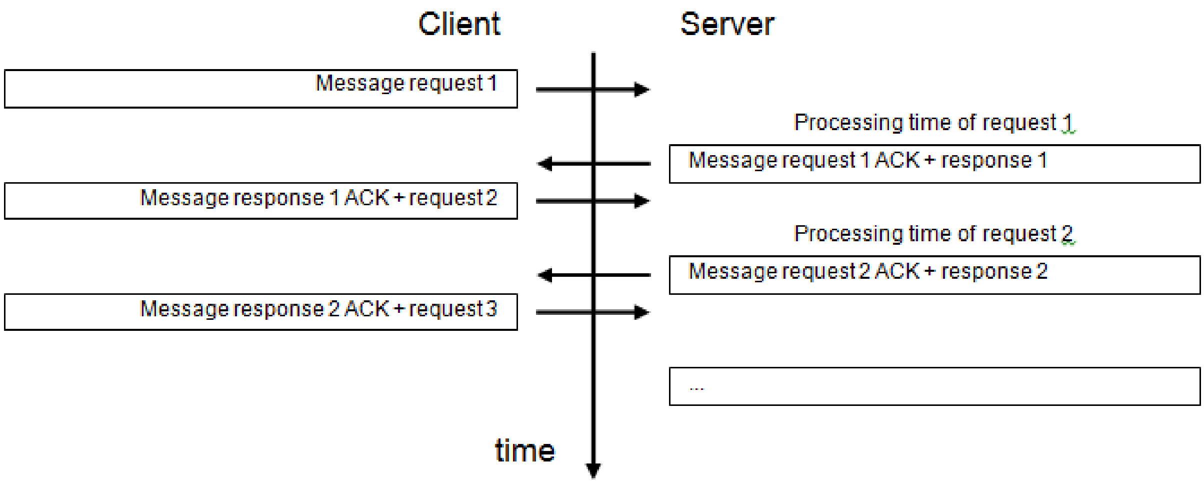


Figure 4-3. Example of a Calibrated ACK Sending

ATTENTION:
All operational systems with support to TCP/IP protocol network interface have equivalent parameters to the discussed in this chapter for the Nexto Series CPUs Ethernet interface. In the Windows ® operational system, these parameters are defined in the system registers, under several different identifications, and must be modified only by network administrators, thus affect all programs and applicative installed under the operational system.

ATTENTION:
The delay parameter in case of ACK sending only applies to communication between the CPU and the MasterTool IEC XE software. To communicate with other devices and/or other protocols (MODBUS, for example) the standard used shall be "no delay".

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 consistency of the retentive and persistent area in %Q, and in the cases of the NX3010, NX3020 and NX3030 CPUs, the options for reading and writing on the memory card.

Configuration	Description	Default	Options
Consist retain and persistent area in %Q	Performs the consistency of retentive and persistent areas in %Q.	Unmarked	- Marked - Unmarked
Enable I/O update per task	Updates the input and output in the tasks where they are used.	Unmarked	- Marked - Unmarked
Enable retain and persistente variables in Function Blocks	Setting to allow the use of retentive and persistent variables in function blocks	Unmarked	- Marked - Unmarked
Copy Project from CPU to Memory Card	**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 4-4. CPU Project Parameters

ATTENTION:
After setting the project copy possibilities and having created the startup 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 Application.crc through the "Locate File..." key, as can be seen on Figure 4-66.

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.

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 software programming tool MT8500.

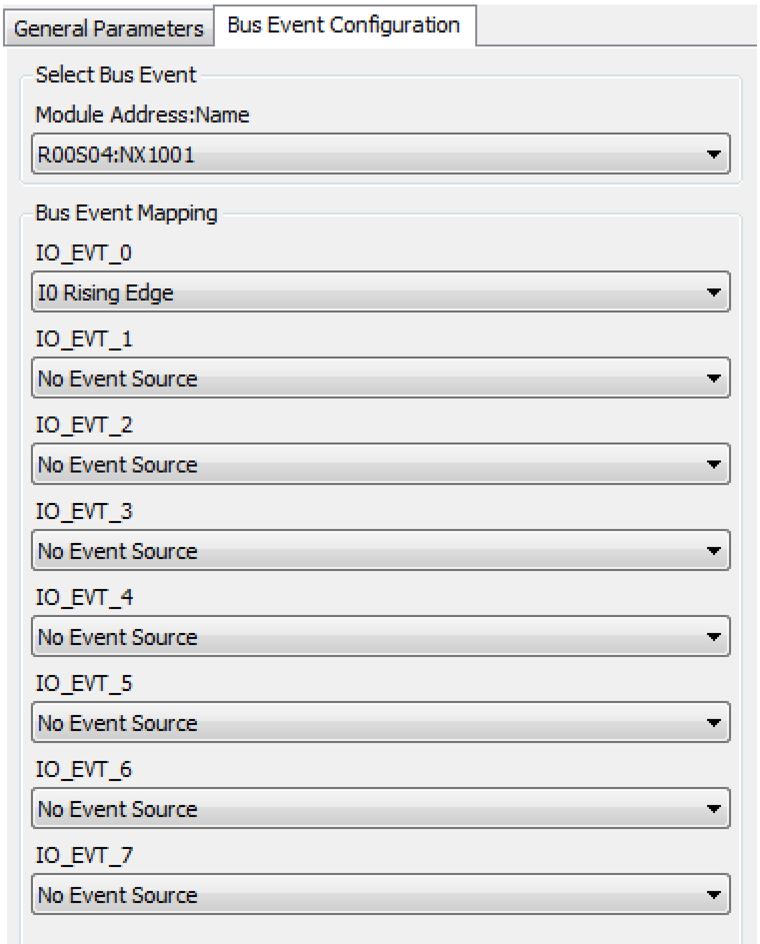


Figure 4-4. Configuration Screen for External Event in CPU

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 4-5 can be chosen.

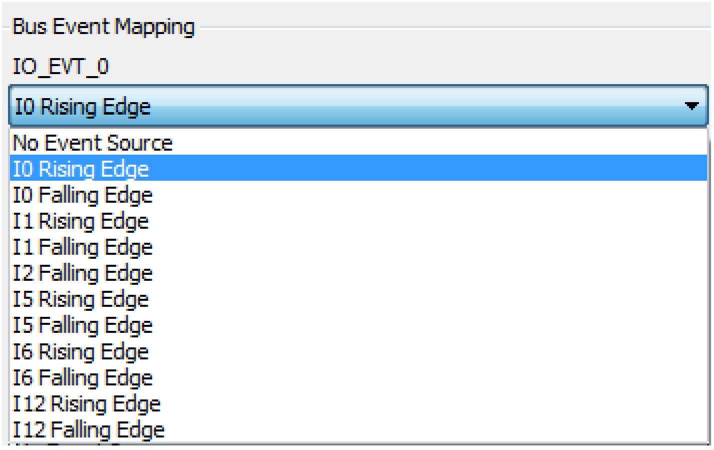


Figure 4-5. NX1001 Module External Event Source Options

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 chapter Project Profiles. In the configuration screen of the ExternInterruptTask00 task (Figure 4-6), 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 POUS 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.

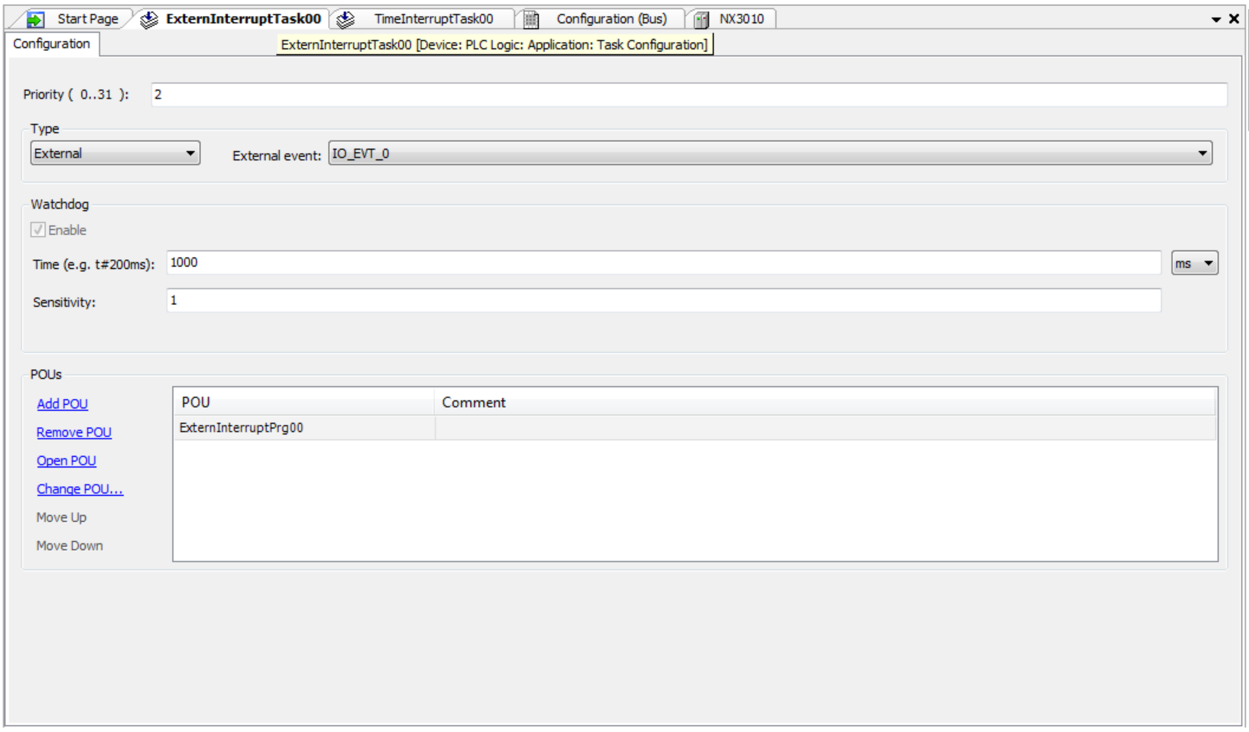


Figure 4-6. ExternInterruptTask00 Configuration Screen

SOE Configuration

The SOE (Sequence of Events) is responsible for the generation of a sequence of digital events. Through the SOE it is possible to analyze the historic behavior of the system variables mapped in its monitoring area. The SOE is an exclusive service available for the NX3020 and NX3030 models.

Once the SOE service has been enabled, the CPU starts to behave as a DNP3 server, thus it is necessary the support to the DNP3 protocol by the client for the use of this resource. The supported object types as well as the function codes and the qualifiers can be found at DNP3 Device Profile

The SOE service uses the %Q addresses in order to form its base of static data. For it, it has to be set a continuous area of %Q memory where the user will inform its beginning and size divided by two. For redundant projects the %Q area also has to be redundant so that in the switchover moment the DNP3 server data base is kept.

The DNP 3 initial address will always be 0, corresponding to %QBxxxx.0 and the last address will be: ((Area Size of %Q * 8)*2) -1.

Thus, once defined the static data base, the user must copy each digital point which should generate events within the %Q continuous area. The maximum number of points which can be copied is 8000.

For the events configuration, it is necessary to inform only the size of the events queue. This queue is persistent and redundant, so the events will not be lost in the switchover moment neither in case of a power supply failure. In case an overflow occurs in the events queue, the oldest events will be overwritten. In case in one single cycle are generated more events than what is supported by the queue, its generation is interrupted and the overflow diagnostic is turned on (SOE[x].bOverflowStatus). For example, if 100+n bits vary in a 100 events configuration, causing a dispose of n events.

The SOE will run in the MainTask context, starting already at the first cycle. The SOE will run at the end of each MainTask cycle, comparing the mapped bits in order to detect transitions occurred in the cycle. In this way, every cycle in which the events are generated, an increase of time in this cycle of the MainTask will occur. In the worst case (1000 events, being generated only 1000 and discarded the remaining ones), this influence will be approximately of 5 ms. Therefore, for an application with the SOE enabled, the user will have to take into account this time when setting the parameters of watchdog time and interval of the MainTask.

For the use of it the user must set the following parameters in the SOE Configuration tab:

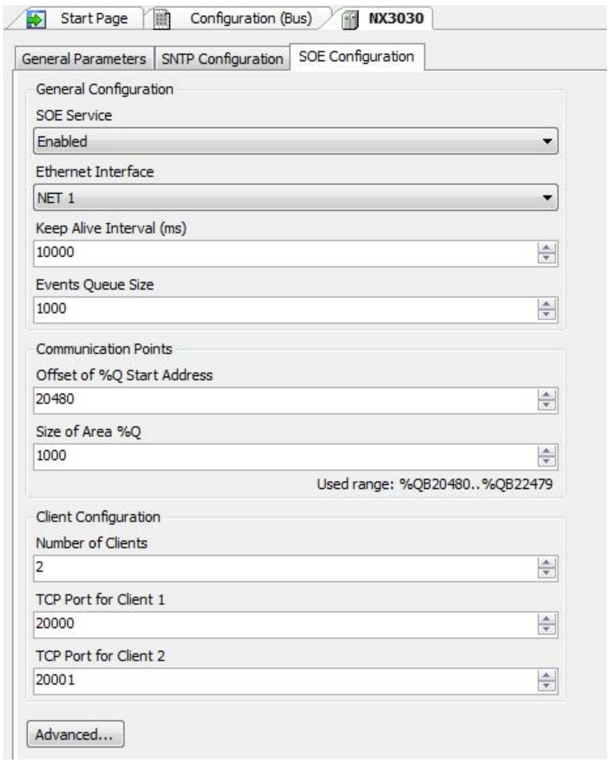


Figure 4-7. Events Sequence Configuration

Configuration	Description	Default Value	Options
General Configurations			
SOE Service	Enables the SOE.	Disabled	Enabled Disabled
Ethernet Interface	Selects the used interface.	NET1	NET1 NET2
Keep Alive Interval (ms)	Keep alive (ms) interval messages.	10000	0 to 4294967295
Events Queue Size	Events queue size	1000	100 to 1000
Communication Points			
Offset of %Q Start Address	Initial address for static data	20480	Any %Q area address can be used
Size of Area %Q	Memory size to be used by the static data (%Q).	1000	1 to 1000
Client Configuration			
Configuration	**Description**	**Default Value**	**Options**
Number of Clients	Defines the number of clients.	2	1, 2
TCP Port for Client 1	Selects the communication port for the first client.	20000	1 to 65535
TCP Port for Client 2	Selects the communication port for the second client.	20001	1 to 65535

Table 4-5. SOE Configuration

Notes:

Data Memory Size: The data memory size reserved to be used by the static data will always be twice the value set as the second half of the memory area is used to store the previous variables values of the first half.

Keep Alive: While it is connected to a client, keep alive messages will be sent in intervals according to what has been set. If the client does not respond to these messages, the connection is closed. That is, a connection between client and server may take a time equal to the interval set to be closed in case of error.

In the advanced options (Advanced... key) it is possible to set the communication addresses regarding to the DNP3 protocol.

Configuration	Description	Default Value	Options
DNP3 Source Address	Origin Address (PLC)	4	0 to 65519
DNP3 Destination Address of Client 1	Address of the first client	3	0 to 65519
DNP3 Destination Address of Client 2	Address of the first client	3	0 to 65519

Table 4-6 SOE. Advanced Configurations

Note:

DNP3 Address: The DNP3 addresses from the range 65520 to 65535 cannot be set at the origin or at a destiny as they are used for messages in broadcast.

ATTENTION:
The DNP3 DataLink messages are not used by the Nexto series CPUs as the standard does not recommend its use them in TCP/IP communications.

Time Synchronization

For the time synchronization, Nexto Series CPUs use the SNTP (Simple Network Time Protocol) protocol. For that, 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. The SNTP client works with a 1 ms resolution, 100 ms precision, which means that when synchronization is performed, the updated time in the client may be up to 100 ms early or late in relation to the server.

The CPU sends the cyclic synchronization requests according to the time set in the SNTP Synchronization Period field. In the first synchronization attempt, just after the service start up, the request is for the first server set in the first server IP address. In case it does not respond, the requests are directed to the second server set in the second server IP address providing a redundancy of SNTP servers. In case the second server does not respond either, the same process of synchronization attempt is performed again but only after the Period of Synchronization having been passed. In other words, at every synchronization period the 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:

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

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

When NX3020 and NX3030 CPUs are used, depending on the SNTP server subnet, the client will use an Ethernet interface which is in the corresponding subnet in order to do the synchronism requests. In case it doesn’t have a configured interface at the same subnetwork of the server the request may be made by any interfaces which can find a route for the server. That is also true for the models NX3004, NX3005 and NX3010.

For the SNTP client, the user must set the following parameters in the SNTP Configuration tab, accessed by the CPU in the devices tree:

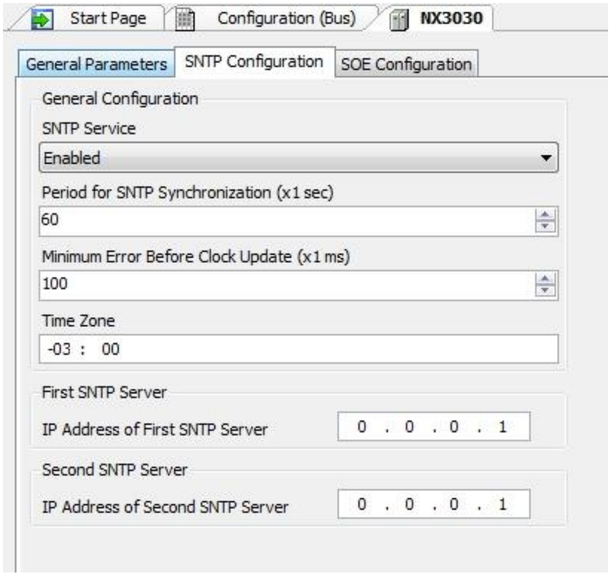


Figure 4-8. SNTP Configuration

Configuration	Description	Standard Default	Options
SNTP Service	Enables the SNTP service	Disabled	Disabled Enabled
Period for SNTP Synchronization (x1 sec)	Time interval of the synchronization requests (seconds)	60	1 to 255
Minimum Error Before Clock Update (x1 ms)	Offset value acceptable between the server and client (milliseconds)	100	100 to 65519
Time zone (hh:mm)	Time zone of the user location. Hours and minutes can be inserted.	-3:00	-12:59 to +13:59
IP Address of the First SNTP Server	IP Address of the primary SNTP server	192.168.15.10	1.0.0.0 to 223.255.255.255
IP Address of the 2º Second SNTP Server	IP Address of the secondary SNTP server	192.168.15.11	1.0.0.0 to 223.255.255.255

Table 4-7. 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.

Factory default: from MasterTool IEC XE version 1.40 and later the factory default value for the IP addresses of SNTP Servers have been changed.

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

> **CAUTION:**
> **It is vital a configuration of at least one SNTP server. It is recommended to set two SNTP servers (primary and secondary). The SNTP synchronism is necessary to generate events with timestamp coherent between CPA and CPB and with world time. Another purpose is to avoid discontinuity during a switchover in applications which reference date and hour, considering that there is no synchronism of date and hour between the CPs through NETA and NETB synchronism channels.**

Daylight Saving Time (DST)

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

For further information, see the section RTC Clock of this manual.

Serial Interfaces Configuration

COM 1 (NX3010/NX3020/NX3030)

The COM 1communicationinterface, present in the NX3010, NX3020 and NX3030 CPUs, is composed by a DB9 female connector for RS-232C pattern. It allows the point to point communication (or in network by using a converter) in MODBUS RTU slave or MODBUS RTU master the 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 Parity Always One Parity Always Zero No Parity
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 4-8. RS-232 Standard Serial Configuration

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 (parity always zero) for a time greater than a character (start bit + data bit + parity bit + stop bit)
- Bit 4: UART frame error. The logic 0 (space) was read when the first stop bit was expected and it should be logic 1 (parity always one)
- 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 10us as base. It saturates in 655.35ms 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. Table 4-9 shows the allowed configurations for COM 1 interfaces of the NX3010, NX3020 and NX3030 CPUs:

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 4-9. Specific Configurations

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

Table 4-10. RS-232 Standard Serial Advanced Configurations

Notes:

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

Event RX DCD: 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.

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

COM1 (NX3004/NX3005) and COM 2 (NX3010/NX3020/NX3030)

The COM 1 (NX3004 and NX3005) and COM 2 (NX3010, NX3020 and NX3030) communication interface is composed by a DB9 female connector for the RS-422 and RS-485 patterns. It allows the point to point or network communication in the open protocols MODBUS RTU slave or MODBUS RTU master.

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

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

Configuration	Description	Default	Options
Serial Type	Serial channel configuration	RS-485	RS-422 and RS-485
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 Parity Always One Parity Always Zero No parity
Data Bits	Sets the data bits quantity in each serial communication character	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	Normal Mode	- Extended Mode: Extended operation mode which delivers information regarding the received data frame (see note on COM 1 section) - Normal Mode: Serial communication normal operation mode

Table 4-11. RS-485/RS-422 Standard Serial Configurations

The serial interfaces Data Bits configuration limits the Stop Bits and Communication Parity fields. Therefore, the number of stop bits and the parity method will vary according to the data bits number. Table 4-12 shows the allowed configurations for COM 1(NX3004 and NX3005) and COM 2 (NX3010, NX3020 and NX3030) interfaces.

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

Table 4-12. Specific Configurations

Advanced Configurations

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

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

Table 4-13. RS-485/RS-422 Standard Serial Advanced Configurations

Ethernet Interfaces Configuration

The Nexto CPUs can provide up to two Ethernet interfaces locations, NET 1 and NET 2. The NX3004, NX3005 and NX3010 CPUs has only the NET1 interface and the CPUs NX3020 and NX3030 have NET 1 and NET 2. In addition to the local Ethernet interfaces, the Nexto Series also provides remote Ethernet interfaces by including the NX5000 module. The NX5000 modules have only the NET 1 interface.

Local Ethernet Interfaces

NET 1

The NET 1 interface is composed by a RJ45 communication connector pattern 10/100Base-TX. It allows the point to point or network communication in the following open protocols, for example: MODBUS TCP Client, MODBUS RTU via TCP Client, MODBUS TCP Server and MODBUS RTU via TCP Server.

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

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

Table 4-14. NET 1 Configuration

NET 2

The NET 2 interface is composed by a RJ45 communication connector pattern 10/100Base-TX. 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
IP Address	IP address of the controller in the Ethernet bus.	192.168.16.1	1.0.0.1 to 223.255.255.254
Sub network Mask	Sub-net mask of the controller in the Ethernet bus.	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Gateway address of the controller in the Ethernet bus.	192.168.16.253	1.0.0.1 to 223.255.255.254

Table 4-15. NET 2 Configuration

ATTENTION:
It is not possible to configure the two local Ethernet interfaces in the same sub-net. This kind of configuration is blocked in MasterTool. This way, each Ethernet interface must be configured in a different Subnetwork.

Remote Ethernet Interfaces

NET 1

NET 1 is an interface composed by an RJ45 communication connector pattern 10/100Base-TX. It allows 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 Value	Options
IP Address	IP address of the Controller in the Ethernet bus	192.168.xx.68	1.0.0.1 to 223.255.255.254
Sub network Mask	Sub net mask of the Controller in the Ethernet bus	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Gateway address of the Controller in the Ethernet bus	192.168.xx.253	1.0.0.1 to 223.255.255.254

Table 4-16. Remote NET 1 Configurations

Reserved TCP Ports

The following TCP ports of the Ethernet interfaces, both local and remote, are used by CPUs services, so they are reserved and cannot be used by the user: 80, 161, 8080, 1217, 1740, 1741, 1742,1743 and 11740.

NX5000 Module Configuration

The NX5000 modules can be inserted in the project to increase the number of Ethernet interfaces if the local CPU interfaces are not enough. Only the NX3005, NX3020 and NX3030 CPUs support the NX5000 module.

The Ethernet channels of the NX5000 modules can be used individually, or arranged in NIC Teaming pairs. NIC Teaming pairs are used to provide redundant Ethernet channels.

An example of typical application for NX5000 module is the setting of a redundant HSDN (High Speed Deterministic Network) for communication between different PLCs. Through this network, several PLCs can exchange messages in an entirely segregated network to ensure determinism and fast communication. Moreover, setting up this network as redundant with NIC Teaming pairs, provides an excellent availability. To build a redundant HSDN, two NX5000 modules must be inserted. Figure 4-9 shows an example of redundant HSDN using two NX5000 modules.

The Figure 4-9 also shows an example with one NX5000 module used isolated (without NIC Teaming redundancy), inserted to the right of the other modules.

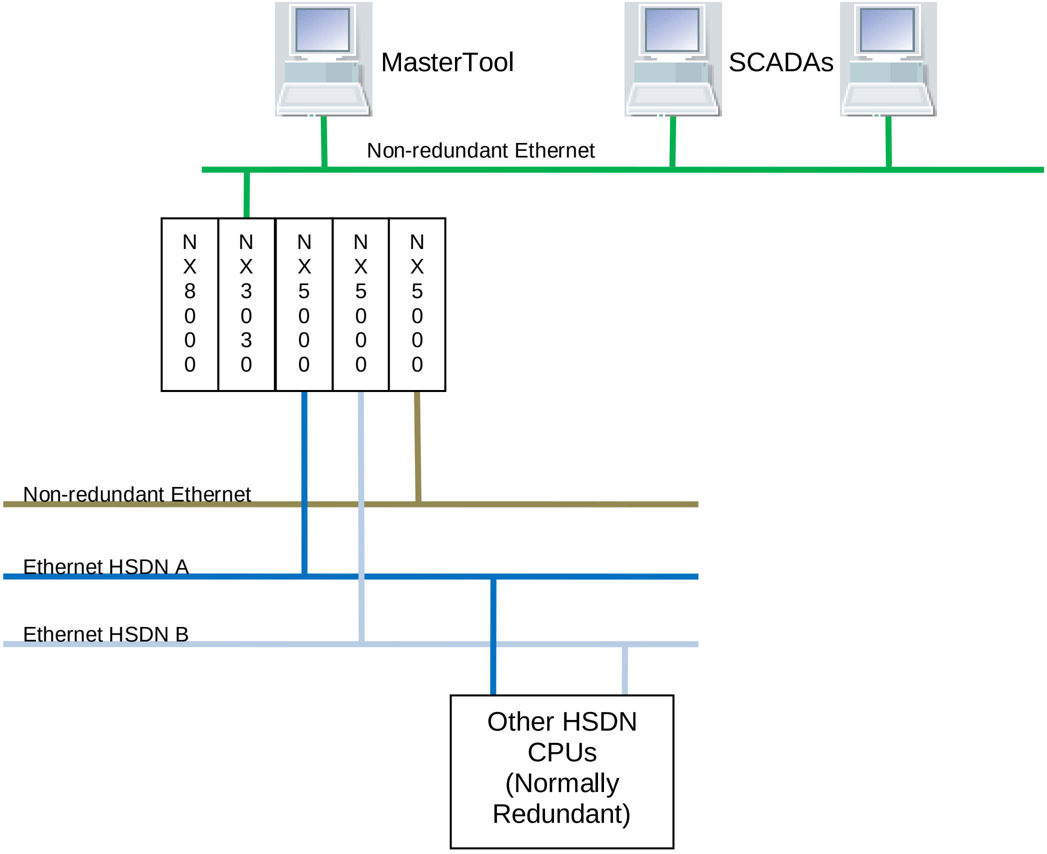


Figure 4-9. Simple and Redundant Ethernet Networks Using NX5000

The two first NX5000 modules from the backplane rack make up a redundant NIC Teaming pair interconnected in two different switches (Ethernet HSDN A and Ethernet HSDN B). At some point, these two switches must be interconnected so that there is connection between the two NIC Teaming ports and greater availability (against double failures).

Such Ethernet architectures enable excellent availability against failures on Ethernet ports, cables and switches.

A set of two Ethernet ports forming a NIC Teaming pair presents a unique IP address connected to the pair of ports. Thus, a client such as a SCADA or MasterTool connected to a server on a PLC does not need to concern about changing the IP address in case there is a failure in some of the NIC Teaming ports.

Diagnostics indicate eventual failures that may arise in any of the NIC Teaming pair ports.

ATTENTION: Both NX3020 and NX3030 CPUs support the NX5000 module and can put two NX5000 together as an NIC Teaming pair.

Using the NX3020 CPU it is possible to insert up to two NX5000 modules in the project. By using the NX3030 CPU it is possible to insert up to six. If it is used a CPU NX3020 or NX3030, it is possible to configure a NIC Teaming pair, using up to the maximum number of modules allowed for each CPU, such as the architecture shown in the Figure 4-9, where we have a NIC Teaming pair and one independent Ethernet interface, using three modules.

In order to put together two NX5000 modules as a redundant pair, these two modules must necessary occupy adjacent positions on the backplane rack and the checkbox “Redundancy of Communication” from the module on the left must be selected, as show in the figure above Figure 4-10.

By doing this, the parameters edition of the module on the right is blocked. The parameters edited in the module inserted on the left get common for the two modules.

On the other hand, clearing the “Redundant Communication” checkbox from the module on the left causes the separation of the modules, which return to behave as individual modules without redundant NIC Teaming.

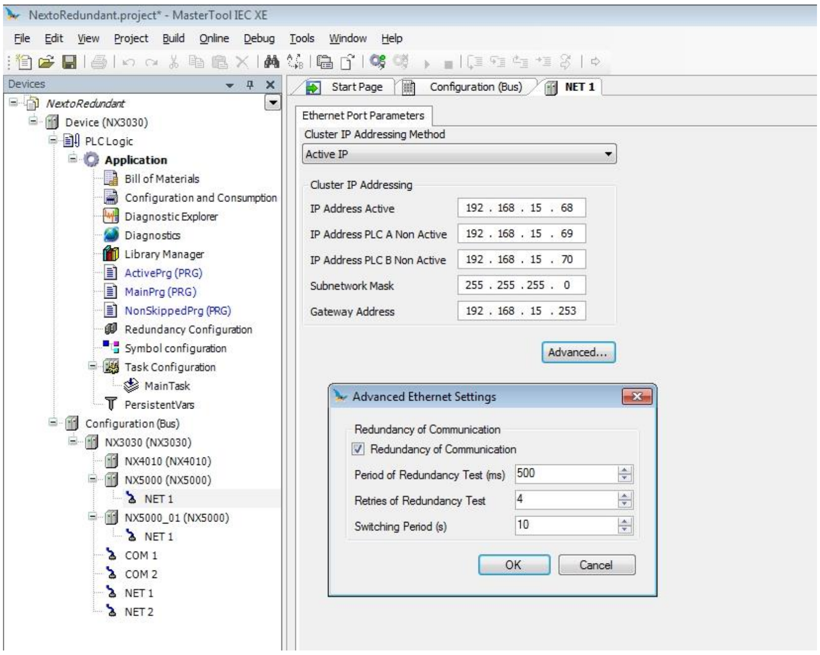


Figure 4-10. NX5000 Redundancy Parameter

Protocols Configuration

Independent of the protocols used in each application, the Nexto Series CPUs possess 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 4-17:

	NX3004	NX3005	NX3010	NX3020	NX3030
Mapped Points	20480	20480	20480	20480	20480
Symbolic MODBUS Mappings	5120	5120	5120	5120	5120
MODBUS Mapping/requests (direct representation/symbolic, respectively)	512	512	512	512	512
NETs – Clients or Servers instances	4	4	4	8	16
COM (n) – Master or Slave instances	1	1	1	1	1

Table 4-17. Limits of the protocols by CPU

Notes:

Mapped Points: Each variable or item of a given data type is assumed to be a mapping. The same is considered for each position of the ARRAY type. This means that if a simple variable is declared, it will be considered a mapping and if an ARRAY type is declared, the count will be equal to the size of the declared ARRAY. The amount of mappings increments by one when there is a simple type of variable being declared independent of the size of the given type. Then, mapping a variable of INT type (16-bit) in a Holding Register of symbolic Modbus drivers or a variable of type LINT (64-bit) in four Holding Register of symbolic Modbus drivers is accounted for as just a mapping.

Symbolic MODBUS Mappings: A mapping is a relationship between an application intern variable and an application protocol object. The limit value for the project mappings corresponds to the sum of all the mappings made within the instances of communication protocols and their respective devices.

MODBUS Mapping/requisitions (directly representation/symbolic, respectively): MODBUS mapping limits (lines) for direct representation and requisitions for MODBUS by symbolic mapping.

ATTENTION:
In cases where the two types of protocol are used together, as one type is used the capacity of the other decreases. For example, if 10240 symbolic mappings are used, it will only be possible to use 256 mappings by direct representation. The ratio between the two types of mapping is of 40 symbolic mappings to each mapping by direct representation.

NETs: Clients or Servers instances: The maximum value defined above is distributed between all Ethernet interfaces of the system, in other words, it includes the expansion modules when they are applied. Examples for this type of task are the MODBUS protocol instances.

COM (n): Master or Slave instances: The “n” represents the serial interfaces number, in other words, even with the expansion modules, the value in the table will be the maximum limit per interface. Examples for this task type are the MODBUS protocol instances.

The maximum number of instances competes between themselves, that is, between the MODBUS RTU Master and Slave there is only one instance that can be configured per interface on any CPU model. Between Ethernet MODBUS Client and server only four (NX3010) eight (NX3020) or sixteen (NX3030) instances can be configured per interface.

The limitations of the MODBUS protocol for Direct Representation and symbolic mapping for the CPUs can be seen in Table 4-18 and Table 4-19, respectively.

Limitations	MODBUS RTU Master	MODBUS RTU Slave	MODBUS Ethernet Client	MODBUS Ethernet Server
Maximum number of mappings per instance	128	32	128	32
Maximum number of devices	64	1	64	1
Maximum number of mappings per device	32	32	32	32
Maximum number of simultaneous requests per instance	-	-	128	64
Maximum number of simultaneous requests per device	-	-	8	64

Table 4-18. 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 stablish 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 4-17.

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 4-19. 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.
- **Slave MODBUS RTU Protocol:** the limit (1) informed relates to serial interfaces that do not allow a Slave to establish communication through the same serial interface, simultaneously, with more than one Master device. It's not necessary, nor is it possible to declare or configure the Master device in the instance of the MODBUS RTU slave protocol. The master device will have access to all the mappings made directly on the instance of MODBUS RTU slave protocol.
- **MODBUS RTU Server Protocol:** the limit (2) informed relates to the Ethernet interfaces, which limit the amount of connections that can be established with other devices through a single Ethernet interface. It is not necessary, nor is it possible to declare or configure Clients devices in the instance of the MODBUS Server protocol. All Clients devices will have access to all the mappings made directly in the instance of the MODBUS Server protocol.

Requests by device: Number of requests, such as reading or writing holding registers, that 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.

ATTENTION:
Communication drivers for symbolic mappings are available only from 1.3.0.20 version of Nexto Series CPUs. Similarly to use this feature is required MasterTool IEC XE version 1.40 or later.

Protocol Behavior x CPU State

Table 4-20 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	Slave/Server	×	×	×	✓	✓	×
	Master/Client	×	×	×	✓	✓	✓
SOE (DNP3)	Outstation	✓	✓	✓	✓	×	✓
EtherCAT	Master	×	×	×	✓	NA	×
OPC	Server	✓	✓	✓	✓	×	✓
SNTP	Client	✓	✓	✓	✓	✓	✓
HTTP	Server	✓	✓	✓	✓	✓	✓
SNMP	Agent	✓	✓	✓	✓	✓	✓

Table 4-20. Protocol Behavior x CPU State

Notes:

Symbol ✓: Protocol remains active and operating normally.

Symbol ×:Protocol is disabled.

EtherCAT: The tests were performed using MainTask as EtherCAT bus task. If another task is used, the protocol will remain active when there's a breakpoint in MainPrg. EtherCAT protocol is not available for CPUs NX3004, NX3005 and NX3010, neither for redundant NX3030. For further information on EtherCAT protocol, consult MasterTool IEC XE User Manual MT8500 - MU299609.

SOE: Sequence of Events protocol (SOE) is not available for NX3004, NX3005 and NX3010 CPU models.

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 chapter 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 chapter.
- 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 Configuration - Serial Interfaces Configuration chapter.

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 4-11.
- 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 4-12.
- Add and configure the MODBUS mappings on Mappings tab as Figure 4-13, 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 4-14, 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.

MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration

The general parameters, found on the MODBUS protocol initial screen (Figure 4-11), are defined as:

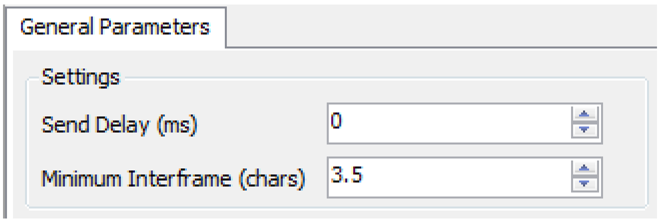


Figure 4-11. MODBUS RTU Master Configuration Screen

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 a 100.0

Table 4-21. 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 4-22 (where "n" is the configured value in the %Q Initial Address of Diagnostic Area field).

Direct Representation 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		bNotRunning	BIT	The master is not running (see bit: bInterruptedByCommand).
%QX(n).2		bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the master was interrupted by the user through command bits.
%QX(n).3		bConfigFailure	BIT	Discontinued diagnosis
%QX(n).4		bRXFailure	BIT	Discontinued diagnosis
%QX(n).5		bTXFailure	BIT	Discontinued diagnosis
%QX(n).6		bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present.
%QX(n).7		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 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		bRestart	BIT	Restart master
%QX(n+2).2		bResetCounter	BIT	Restart diagnostics statistics (counters)
%QX(n+2).3		bDiag_19_reserved	BIT	Reserved

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS_RTU_MASTER_1.*		Size	Description
%QX(n+2).4		bDiag_20_reserved	BIT	Reserved
%QX(n+2).5		bDiag_21_reserved	BIT	Reserved
%QX(n+2).6		bDiag_22_reserved	BIT	Reserved
%QX(n+2).7		bDiag_23_reserved	BIT	Reserved
%QB(n+3)	byDiag_03_reserved		BYTE	Reserved
Communication Statistics:				
%QW(n+4)	tStat.*	wTXRequests	WORD	Counter of request transmitted by the master (0 to 65535)
%QW(n+6)		wRXNormalResponses	WORD	Counter of normal responses received by the master (0 to 65535)
%QW(n+8)		wRXExceptionResponses	WORD	Counter of responses with exception codes received by the master (0 to 65535)
%QW(n+10)		wRXIllegalResponses	WORD	Counter of illegal responses received by master – invalid syntax, not enough received bytes, invalid CRC – (0 to 65535)
%QW(n+12)		wRXOverrunErrors	WORD	Counter of overrun errors during reception - UART FIFO or RX line – (0 to 65535)
%QW(n+14)		wRXIncompleteFrames	WORD	Counter of answers with construction errors, parity or failure during reception (0 to 65535)
%QW(n+16)		wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using RTS/CTS handshake, during transmission (0 to 65535)
%QW(n+18)			WORD	Reserved

Table 4-22. MODBUS RTU Master Diagnostics

Note:

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

Devices Configuration – Symbolic Mapping configuration

The slave devices configuration, shown on Figure 4-12, follows the parameters below:

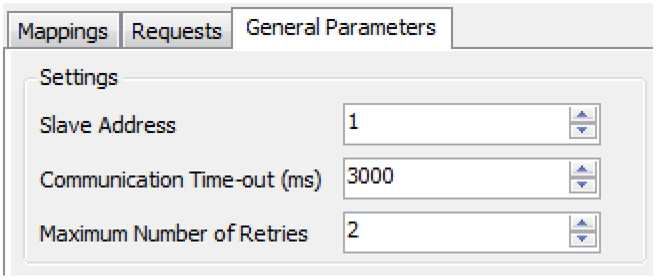


Figure 4-12. Device General Parameters Settings

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

Table 4-23. 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 Protocols Configuration-Communication Performance chapter.

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..

Mappings Configuration – Symbolic Mapping Settings

The MODBUS relations configuration, showed on Figure 4-13, follows the parameters described on Table 4-24:

Figure 4-13. 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	-	Write Coil (1 bit) Read Coil (1 bit) Write Holding Register (16 bits) Read Holding Register (16 bits) Holding Register – AND Mask (16 bits) Holding Register – OR Mask (16 bits) Input Register (16 bits) Input Status (1 bit)
Data Initial 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 4-24. 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
Writing Coil	1	Writing digital output
Reading Coil	1	Reading digital output
Writing Holding Register	16	Writing analog output
Reading Holding Register	16	Reading analog output
Holding Register with AND mask	16	Analog output which can be read or written with AND mask.
Holding Register with OR mask	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 4-25. Data Types Supported in MODBUS RTU Master

Data Initial 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 the user the memory address range used by the MODBUS interface.

Requests Configuration –Symbolic Mapping Settings

The configuration of the MODBUS requests, viewed in Figure 4-14, follow the parameters described in Table 4-26:

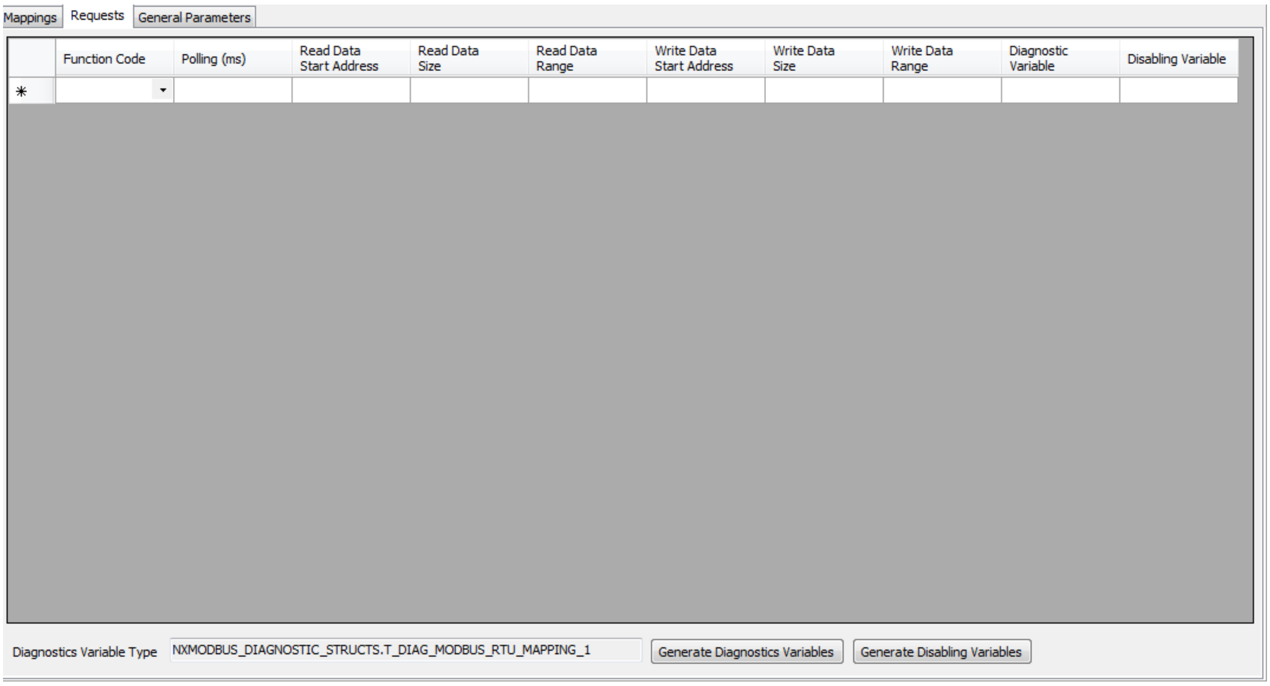


Figure 4-14. Data Requests Screen MODBUS Master

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 Coil 06 – Write Register 15 – Write Multiple Coils 16 – Write Multiple Registers 22 – Register Write Mask 23 – Read/Write Multiple Registers
Scan (ms)	Communication period (ms)	100	0 to 3600000
Initial Address of the Read Data	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
Initial Address of the Write Data	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 4-26. MODBUS Master 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:

Function Type	Code		Description
	DEC	HEX	
Access to Variables	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 coil (FC 05)
	6	0x06	Write holding register (FC 06)
	15	0x0F	Write multiple coils (FC 15)
	16	0x10	Write holding registers (FC 16)
	22	0x16	Register write mask (FC 22)
	23	0x17	Read/Write holding registers (FC 23)

Table 4-27. MODBUS Functions Supported by Nexto CPUs

Scan: 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 scan and after that, a new communication will be executed.

Initial Address of the Read Data: 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 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (FC 3): 125
- Read Input Registers (FC 4): 125
- Read/Write Holding 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.

Initial Address of the Write Data: 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 5): 1
- Write Single Holding Registers (FC 6): 1
- Write Multiple Coils (FC 15): 1968
- Write Holding Registers (FC 16): 123
- Register Write Mask (FC 22): 1
- Read/Write Holding 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 and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 4-28 (n is the value configured in the Diagnostic Initial Address field in %Q).

Direct Representation 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		bCommExecuting	BIT	Active communication
%QX(n).2		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		bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
%QX(n).4		bCommOk	BIT	Communication terminated previously was held successfully.
%QX(n).5		bCommError	BIT	Communication terminated previously had an error. Check error code.
%QX(n).6		bCommAborted	BIT	Not used in MODBUS RTU Master
%QX(n).7		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 4-29 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)*
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 4-28. MODBUS Relations Diagnostics

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 5 and 6).
8	ERR_REFERENCE_NUMBER	Invalid reference number (FC 15 and 16).
9	ERR_INVALID_FRAME_SIZE	Reply shorter than expected.
21	ERR_SEND	Error during transmission stage.
22	ERR_RECEIVE	Error during reception stage.
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	Timeout while waiting CTS = false in transmission.
44	ERR_CTS_ON_TIMEOUT	Timeout while waiting CTS = true in transmission.
128	NO_ERROR	No error since startup.

Table 4-29. MODBUS Master 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.

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 Master protocol, such as: communication times and direct representation variables (%Q) to receive diagnostics.
- The descriptions of each configuration are listed below in this chapter.
- Add and configure devices by setting address, direct representation variables (%Q) to disable the relations, time-outs and number of communication retries.
- 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 chapter.

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

The General parameters, found on the home screen of MODBUS protocol configuration (Figure 4-15), are defined as:

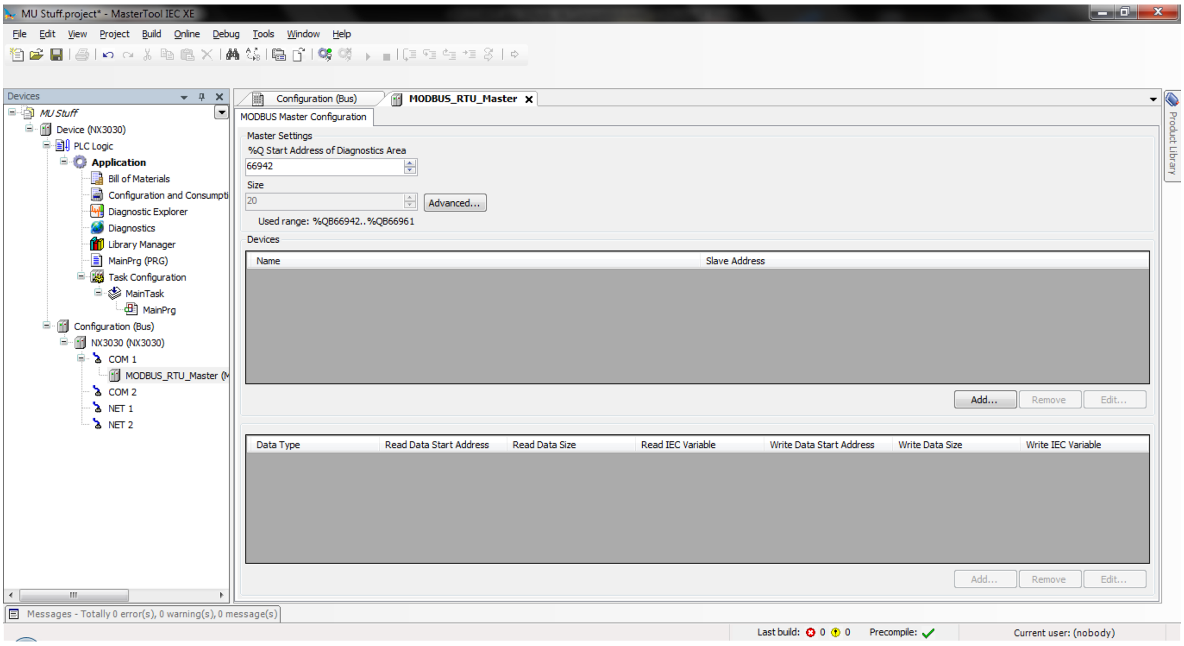


Figure 4-15 MODBUS RTU Master Setup Screen

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

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

Table 4-30. 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 chapter Technical Description.

Default Value: the factory default value cannot be set to the Initial Address of Diagnostics in %Q 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 4-22.

The communication times of the MODBUS Master protocol, found on the button "Advanced ..." in the configuration screen are divided into Transmission Delay and Minimum Interframe, further details are described in section Mappings Configuration – Symbolic Mapping Settings.

Devices Configuration – Configuration for Direct Representation (%Q)

The configuration of the slave devices, viewed in Figure 4-16, comprises the following parameters:

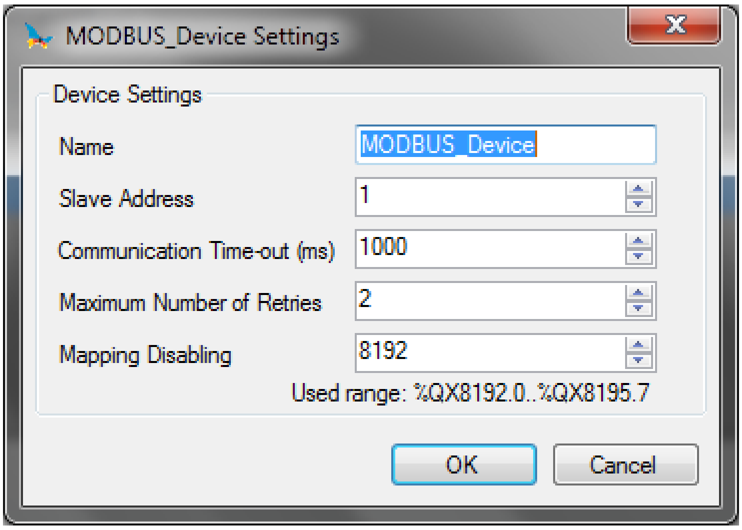


Figure 4-16. Device Configuration

Configuration	Description	Default Value	Option
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 4-31. 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 Mapping 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 chapter Technical Description.

Default Value: the factory default value cannot be set to the Disabling Area of Mappings 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, time-out and maximum number of retries parameters see notes in section Devices Configuration – Configuration for Direct Representation (%Q).

Mappings Configuration – Configuration for Direct Representation (%Q)

The MODBUS relations settings, viewed in Figure 4-17 and Figure 4-18, follow the parameters described in Table 4-32:

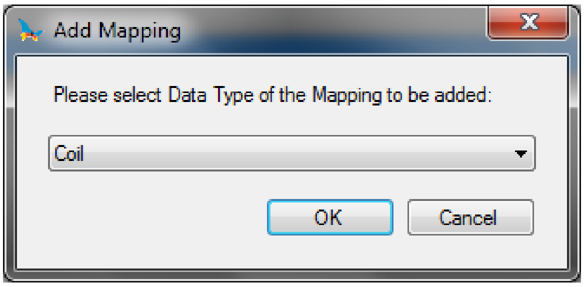


Figure 4-17. MODBUS Data Type

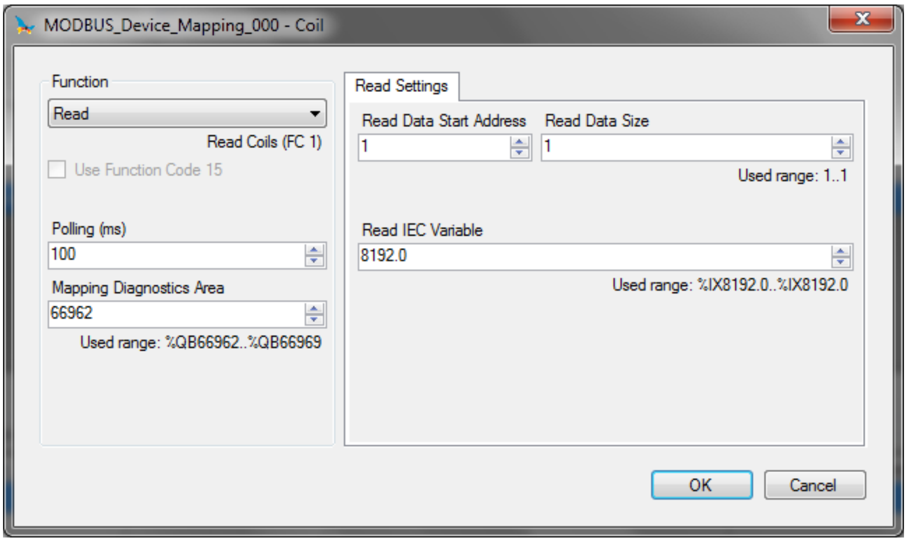


Figure 4-18. MODBUS Function

In Table 4-32, 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 Write Mask
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
Read IEC Variable	Initial address of the read variables (%I)	-	0 to 2147483646
Write Data Start Address	Initial address of the MODBUS 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

Write Mask of IEC Variables	Initial address of the variables for the write mask (%Q)	-	0 to 2147483644

Table 4-32. Device Mapping

Notes:

Function: the available data types are detailed in the Table 4-25 and MODBUS functions (CF) are available in the Table 4-27.

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 chapter Technical Description. The configured MODBUS relations diagnostics are described in Table 4-28.

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

Read IEC Variable: if the MODBUS data type is Coil or Input Status (1-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-bit), 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 chapter Technical Description.

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-bit), 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 chapter Technical Description.

Write Mask: the function Write Mask (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 chapter Technical Description.

Default Value: the factory default value cannot be set for the Mapping Diagnostic Area, IEC Reading Variable, IEC Writing Variable and Writing Mask 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 Reading/Writing 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.

MODBUS RTU SLAVE

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

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 -MU299605. 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 Initial Programming chapter.
- 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.

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.

MODBUS Slave Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol initial screen (Figure 4-19), are defined as:

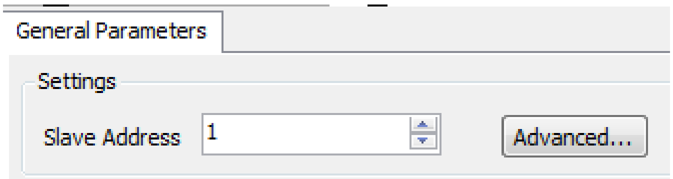


Figure 4-19. MODBUS RTU Slave Configuration Screen

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

Table 4-33. 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 4-20 and in Table 4-34.

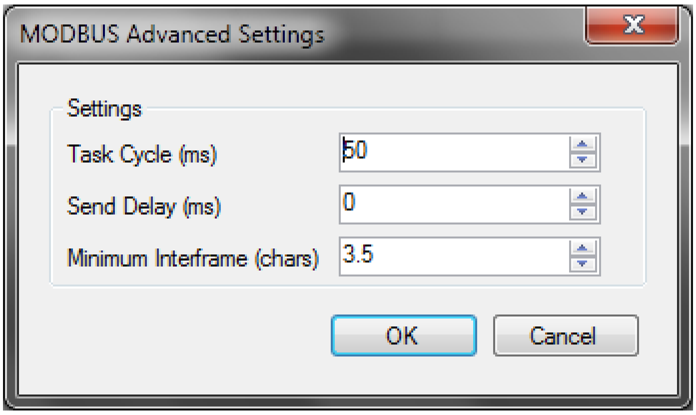


Figure 4-20. Modbus Slave Advanced Configurations

Configuration	Description	Default	Possibilities
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

Table 4-34. 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's 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. 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 4-35 (where "n" is the configured value in the %Q Initial Address of Diagnostic Area field).

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		bNotRunning	BIT	The slave is not in execution (see bit: bInterruptedByCommand)
%QX(n).2		bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the slave was interrupted by the user through command bits
%QX(n).3		bConfigFailure	BIT	Discontinued diagnosis
%QX(n).4		bRXFailure	BIT	Discontinued diagnosis
%QX(n).5		bTXFailure	BIT	Discontinued diagnosis
%QX(n).6		bModuleFailure	BIT	Discontinued diagnosis
%QX(n).7		bDiag_7_reserved	BIT	Reserved
Error codes:				
%QB(n+1)	eErrorCode		SERIAL_STATUS (BYTE)	0: there is no error 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

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS_RTU_SLAVE_1 *.		Size	Description
				parameter 9: invalid time-out parameter 10: serial port busy 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid modem signal method 22: time-out of CTS = true 23: time-out of CTS = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from the manual 33: invalid flow control for the configured serial port 34: data reception not allowed in the normal mode 35: data reception not allowed in the 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 slave
%QX(n+2).1		bRestart	BIT	Restart slave
%QX(n+2).2		bResetCounter	BIT	Restart diagnostics statistics (counters)
%QX(n+2).3		bDiag_19_reserved	BIT	Reserved
%QX(n+2).4		bDiag_20_reserved	BIT	Reserved
%QX(n+2).5		bDiag_21_reserved	BIT	Reserved
%QX(n+2).6		bDiag_22_reserved	BIT	Reserved
%QX(n+2).7		bDiag_23_reserved	BIT	Reserved
%QB(n+3)	byDiag_03_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)		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)		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)

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS_RTU_SLAVE_1 *.		Size	Description
%QW(n+10)		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)		wRXOverrunErrors	WORD	Counter of frames with overrun errors during reception – UART FIFO or RX line – (0 to 65535).
%QW(n+14)		wRXIncompleteFrames	WORD	Counter of frames with construction errors, parity or failure during reception (0 to 65535).
%QW(n+16)		wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using the RTS/CTS handshake, during the transmission (0 to 65535).
%QW(n+18)		wDiag_18_reserved	WORD	Reserved

Table 4-35. MODBUS RTU Slave Diagnostic

Note:

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

Configuration of the Relations – Symbolic Mapping Setting

The MODBUS mapping configuration, depicted on Figure 4-21 follow the parameters described on Table 4-36.

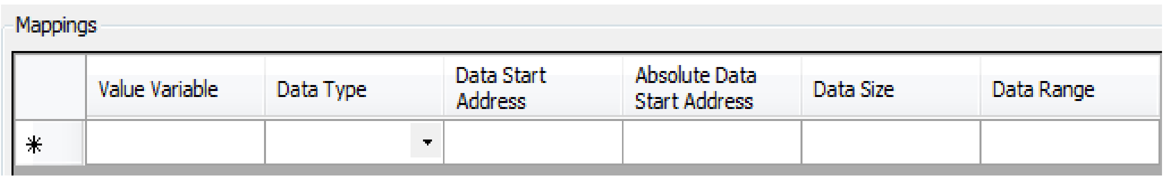


Figure 4-21. MODBUS Data Mappings Screen

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 (1-bit) Input Status (1-bit) Holding Register (16-bit) Input Register (16-bit)
Data Start Address	MODBUS data initial address	-	1 to 65536
Data Size	MODBUS data size	-	1 to 65536
Data Range	The data address range configured	-	-

Table 4-36. MODBUS Mappings Configurations

Notes:

Variable Value: 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 4-37. Data types supported in MODBUS RTU Slave

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 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.

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 chapter.

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

The general parameters, found on the home screen of MODBUS protocol configuration (Figure 4-22), are defined as:

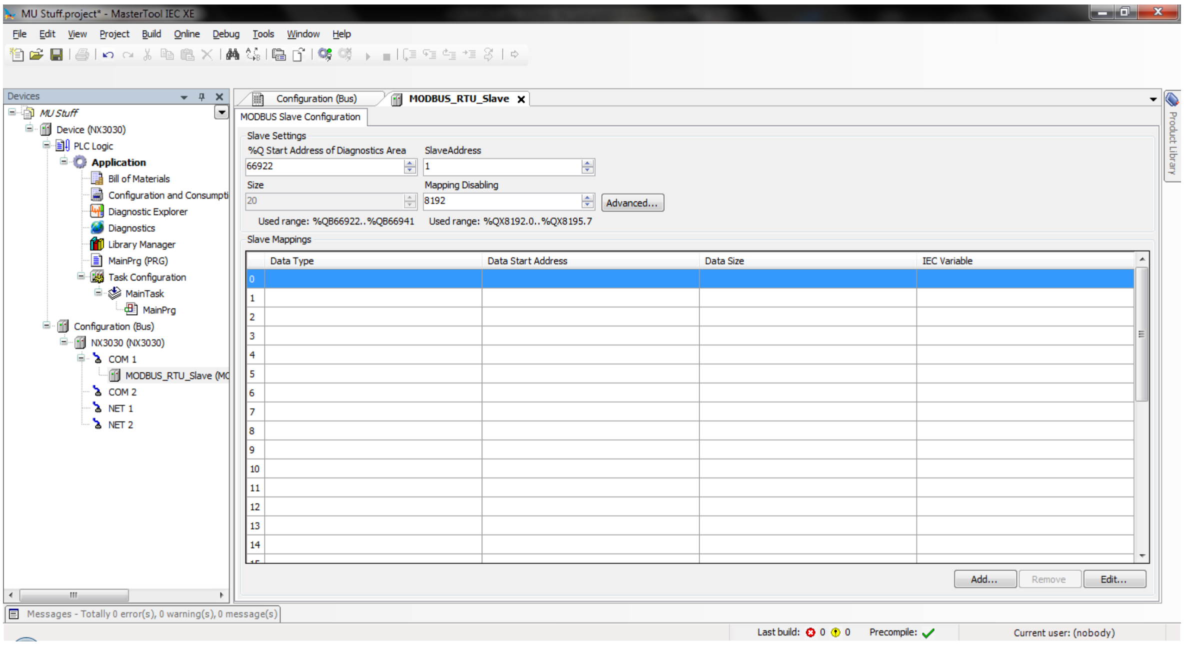


Figure 4-22. MODBUS RTU Slave Configuration Screen

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

Configuration	Description	Default Value	Options
%Q Start Address of DiagnosticsArea	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 4-38. Address and Direct Representation Variables Settings

Notes:

%Q Start Address of DiagnosticsArea: this field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found in chapter Technical Description.

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 Technical Description chapter.

Default Value: the factory default value cannot be set for the Initial Address of Diagnostics in %Q and Disabling of Mappings 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 protocol diagnostics and commands are described in the Table 4-35.

The communication times of the MODBUS Slave protocol, found on the button "Advanced ..." of the configuration screen, are described in Table 4-34.

Mappings Configuration – Configuration via Direct Representation (%Q)

The settings of the MODBUS relations, viewed in Figure 4-23 and Figure 4-24, follow the parameters described in Table 4-39:

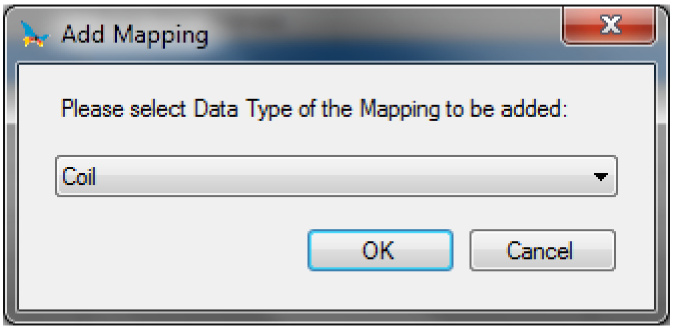


Figure 4-23. Adding MODBUS Relations

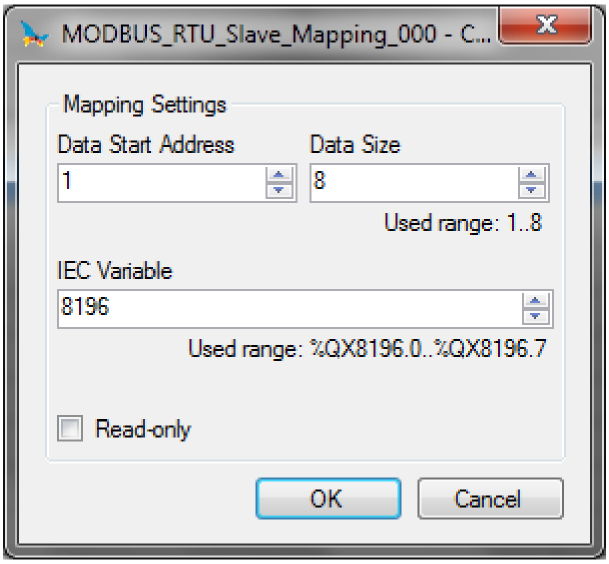


Figure 4-24. Configuring the MODBUS Relation

Configuration	Description	Default Value	Options
Data Type	MODBUS data type	Coil	Coil (1-bit) Holding Register (16-bit) Input Status (1-bit) Input Register (16-bit)
Data Start Address	Initial address of the MODBUS 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 4-39. 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 type, the data field size 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.

IEC Variable: in case the MODBUS data type is Coil or Input Status (1-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-bit), 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 Technical Description chapter.

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
- Write register mask (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.

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 4-17.

Figure 4-25 represents some of the communication possibilities using the MODBUS TCP protocol simultaneously with the MODBUS RTU via TCP protocol.

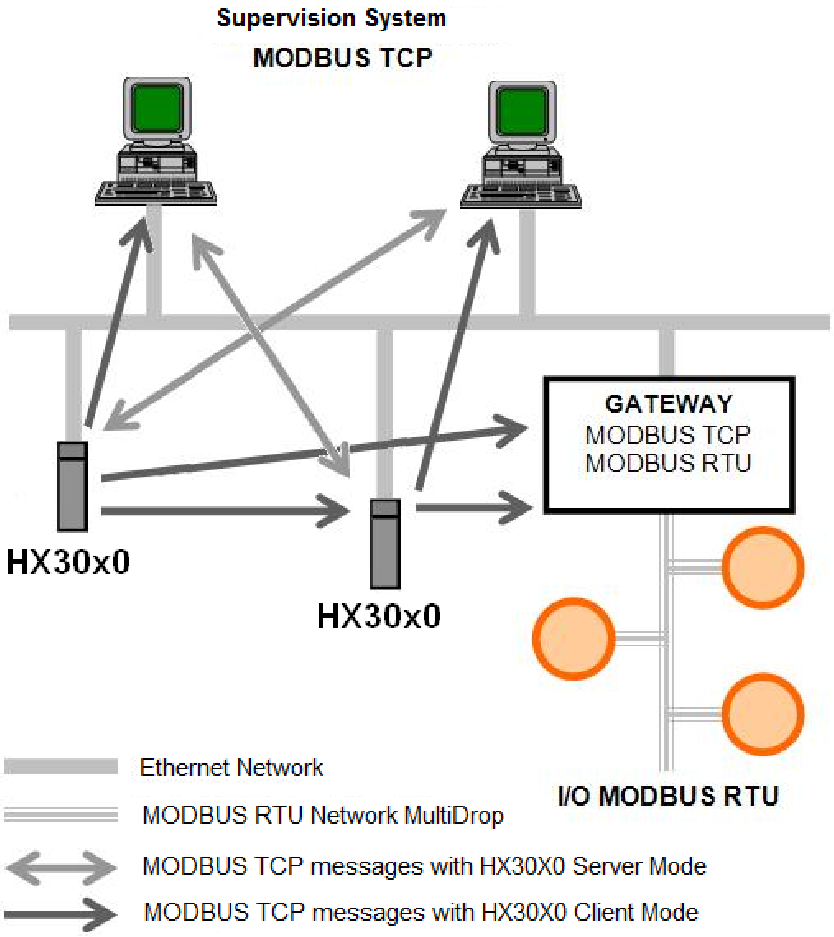


Figure 4-25. MODBUS TCP Communication Network

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 Mapping Disabling 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.

Table 4-40 and Table 4-41 bring, respectively, the complete list of data and MODBUS functions supported by the Nexto CPUs.

Data type	Size [bits]	Description
Coil	1	Digital output which can be read or written
Input Status	1	Digital input which can be only read
Holding Register	16	Analog output which can be read or written
Input Register	16	Analog input which can be only read

Table 4-40. MODBUS Data Types Supported by Nexto CPUs

Function Type	Code		Description
	DEC	HEX	
Variables Access	1	0x01	Coils reading (FC 1)
	2	0x02	Input status reading (FC 2)
	3	0x03	Holding registers reading (FC 3)
	4	0x04	Input registers reading (FC 4)
	5	0x05	Coil writing (FC 5)
	6	0x06	Holding register writing (FC 6)
	15	0x0F	Multiple coils writing (FC 15)
	16	0x10	Multiples holding registers writing (FC 16)
	22	0x16	Writing mask of a holding register (FC 22)
	23	0x17	Multiples holding registers reading/writing (FC 23)

Table 4-41. 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 NET 1 or NET 2 (or both, in the case of more than one communication network). To perform this procedure, refer to the section Initial Programming - Inserting a Protocol Instance.
- Configure the Ethernet interface. To perform this procedure, see section Ethernet Interfaces Configuration.

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 – MU299605 or on Initial Programming - Inserting a Protocol Instance section.

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, and 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.

MODBUS Client Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol configuration initial screen (Figure 4-26), are defined as:

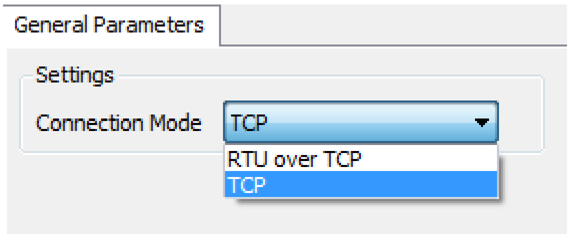


Figure 4-26. MODBUS Client General Parameters Configuration Screen

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

Table 4-42. 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 4-43 (where "n" is the configured value in the %Q Initial Address of Diagnostic Area field).

Direct Representation 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		bNotRunning	BIT	The client is not in execution mode (see bit bInterruptedByCommand)
%QX(n).2		bInterruptedByCommand	BIT	The bit bNotRunning was enabled, as the client was interrupted by the user through command bits
%QX(n).3		bConfigFailure	BIT	Discontinued diagnostics
%QX(n).4		bRXFailure	BIT	Discontinued diagnostics
%QX(n).5		bTXFailure	BIT	Discontinued diagnostics
%QX(n).6		bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present
%QX(n).7		bAllDevicesCommFailure	BIT	Indicates that all devices configured in the Client are in fail
%QB(n+1)	byDiag_1_reserved		BYTE	Reserved
Command bits, automatically initialized:				
%QX(n+2).0	tCommand.*	bStop	BIT	Stop client
%QX(n+2).1		bRestart	BIT	Restart client
%QX(n+2).2		bResetCounter	BIT	Restart the diagnostic statistics

				(counters)
%QX(n+2).3		bDiag_19_reserved	BIT	Reserved
%QX(n+2).4		bDiag_20_reserved	BIT	Reserved
%QX(n+2).5		bDiag_21_reserved	BIT	Reserved
%QX(n+2).6		bDiag_22_reserved	BIT	Reserved
%QX(n+2).7		bDiag_23_reserved	BIT	Reserved
%QB(n+3)	byDiag_03_reserved		BYTE	Reserved
Communication Statistics:				
%QW(n+4)		wTXRequests	WORD	Counter of number of requests transmitted by the client (0 to 65535)
%QW(n+6)		wRXNormalResponses	WORD	Counter of normal answers received by the client (0 to 65535)
%QW(n+8)		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)		wDiag_12_reserved	WORD	Reserved
%QW(n+14)		wDiag_14_reserved	WORD	Reserved
%QW(n+16)		wDiag_16_reserved	WORD	Reserved
%QW(n+18)		wDiag_18_reserved	WORD	Reserved

Table 4-43. MODBUS Client Protocol Diagnostics

Note:

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

Device Configuration – Configuration via Symbolic Mapping

The client devices configuration, depicted on Table 4-44, follow the parameters:

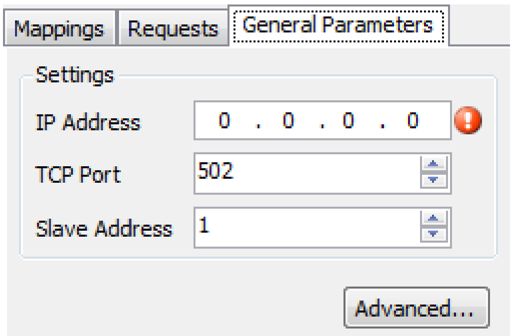


Figure 4-27. Device General Parameters Settings

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	-	0 to 255

Table 4-44. 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. They are: 80, 8080, 1217, 1740, 1741, 1742,1743 and 11740.

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 number of simultaneous requests, Time out, Time-out Mode of Connection 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 timeout. 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 4-45. MODBUS Client Advanced Configurations

Notes:

Maximum Simultaneous Requests: it is used with a high scan cycle. This parameter is fixed in 1 (not editable) when the configured protocol is MODBUS RTU via 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 Protocols Configuration-Communication Performance chapter.

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

- *Connection is closed after an 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 the 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 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.

Mappings Configuration – Configuration via Symbolic Mapping

The MODBUS Client mappings configuration, showed on Figure 4-28, follow the parameters described on Table 4-46:

Figure 4-28. MODBUS Data Type

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 Write (1-bit) Coil Read (1-bit) Holding Register Write (16-bit) Holding Register Read (16 bit) Holding Register – AND Mask (16-bit) Holding Register – OR Mask (16-bit) Input Register (16-bit) 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 data range address configured	-	-

Table 4-46. MODBUS Client 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 type	Size [bits]	Description
Coil Write	1	Write digital output
Coil Read	1	Read digital output
Holding Register Write	16	Write analog output
Holding Register Read	16	Read analog output
Holding Register- AND Mask	16	AND mask used in “write analog output”
Holding Register- OR Mask	16	OR mask used in “write analog output”
Input Register	16	Analog input that can only be read
Input Status	1	Digital input which can only be read

Table 4-47. Data types supported in MODBUS Client

Data Start Address: initial address of the MODBUS mapping data.

Data Size: the size value specifies 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.

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

Requests Configuration – Configuration via Symbolic Mapping

The setting of the MODBUS requests, displayed in Figure 4-29, follows the parameters described in Table 4-48:

Figure 4-29. 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 Coil 06– Write Register 15– Write multiple Coils 16 – Write Multiple Registers 22– Masked Writing of Register 23 –Read/Write Multiple Register
Polling (ms)	Period of communication (ms)	100	0 to 3600000
Read Data Start Address	Initial address of the MODBUS read data	-	1 to 65536
Read Data Size	MODBUS read data size	-	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	MODBUS write data size	-	Depends on the function used
Write Data Range	MODBUS write data address range	-	0 to 2147483647
Diagnostic	Diagnostic variable name	-	Name of a variable declared in a program

Variable			or GVL
Disabling Variable	Variable used to disable MODBUS	-	Field for symbolic variable used to disable MODBUS requests individually configured. This variable must be of BOOL type. The variable can be simple or array element and can be in structures.

Table 4-48. MODBUS Client Relations Configuration

Notes:

MODBUS Relation Settings: 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 functions (FC) available are as follows:

Function Type	Code		Description
	DEC	HEX	
Access to Variables	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 a coil (05 FC)
	6	0x06	Write a holding register (FC 06)
	15	0x0F	Write multiple coils (FC 15)
	16	0x10	Write holding registers (FC 16)
	22	0x16	Register write mask (FC 22)
	23	0x17	Read/Write holding registers (FC 23)

Table 4-49. MODBUS Client Functions

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 scan 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 (HR 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (HR 3): 125
- Read Input Registers (HR 4): 125
- Read/Write Holding Registers (FC 23): 121

Read Data Range: this field shows the MODBUS read data range configured for each request. The initial address plus 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 MODBUS write data.

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

- Write Coil (HR 5): 1
- Write Holding Registers (FC 6): 1
- Write Multiple Coils (FC 15): 1968
- Write Holding Registers (FC 16): 123
- Register write mask (FC 22):
- Read/Write Holding Registers (FC 23): 121

Write Data Range: this field shows the MODBUS write data range configured for each request. The initial address of writing plus the size of the write data will result in the range of write data for each request.

Diagnostic Variable: the configured MODBUS request diagnostics, either by symbolic mapping or by direct representation, are stored in variables of type T_DIAG_MODBUS_ETH_CLIENT_1 and the mapping by direct representation are in 4-byte and 2-word, which is described in Table 4-50 (n is the value configured in the field Diagnostics Initial Address in %Q).

Direct Representation Variable	Diagnostics of T_DIAG_MODBUS_ETH_MAPPING_1.* Type Variable		Size	Description
Communication Status Bits:				
%QX(n).0	byStatus.*	bCommIdle	BIT	Communication idle (waiting to be executed)
%QX(n).1		bCommExecuting	BIT	Active communication
%QX(n).2		bCommPostponed	BIT	Communication deferred, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of time-out. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true.
%QX(n).3		bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
%QX(n).4		bCommOk	BIT	Communication terminated previously was held successfully.
%QX(n).5		bCommError	BIT	Communication terminated previously had an error. Check error code.
%QX(n).6		bCommAborted	BIT	Previously terminated communication was interrupted due to connection failure.
%QX(n).7		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 4-51 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) *
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 4-50. MODBUS Client Relations Diagnostics

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.
6	ERR_INVALID_PROTOCOL_ID	Unidentified protocol. The reply's protocol is different than expected.
7	ERR_NOT_ECHO	The reply is not an echo of the request (FC 5 and 6).
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 while establishing transmission.
42	ERR_RECEIVE_TIMEOUT	Application level time-out while waiting for reply.
128	NO_ERROR	No error since startup.

Table 4-51. MODBUS Client error codes

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

MODBUS Ethernet Client 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 protocol client, such as: protocol and direct representation variables (%Q) to receive diagnostics.
- The descriptions of each configuration are listed below in this chapter.

- Add and configure devices by setting address, direct representation variables (%Q) to disable the relations and communication port.
- Add and configure MODBUS relations, specifying the data type and MODBUS function, time-outs, direct representation variables (%Q) to receive diagnoses of the relation and other to receive/write the data, amount of data to be reported and polling of the relation.

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

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

The General parameters, found on the home screen of MODBUS protocol configuration (Figure 4-30), are defined below.

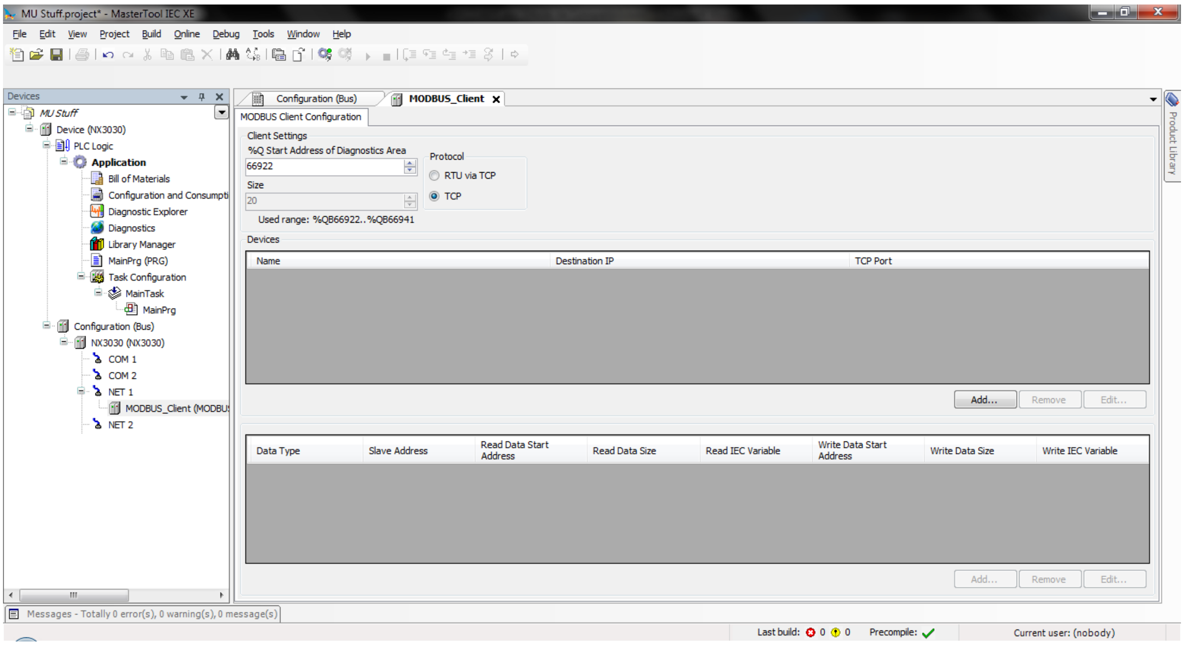


Figure 4-30. MODBUS Client Setup Screen

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 4-52. Protocol selection and direct representation variables for 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 chapter Technical Description.

Default Value: the default value cannot be defined for the Initial Address of Diagnostics in %Q 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 4-43.

Device Configuration – Configuration via Direct Representation (%Q)

The configuration of client devices, displayed in Figure 4-31, includes the following parameters:

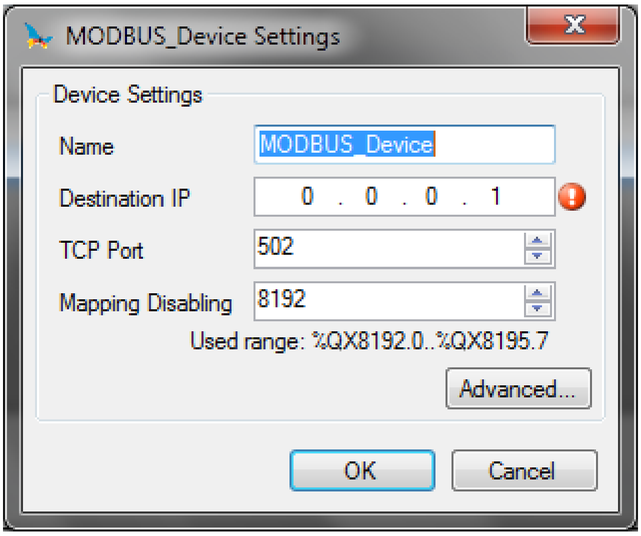


Figure 4-31. Configuring MODBUS Client

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 4-53. Configuration of Client Devices

Notes:

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. They are: 80, 8080, 1217, 1740, 1741, 1742,1743 and 11740.

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 chapter Technical Description - Specific Features.

Default Value: factory default cannot be set for the Disabling of Mappings 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 – MODBUS Master Protocol Configuration by Symbolic Mapping.

Mapping Configuration – Configuration via Direct Representation (%Q)

The setting of the MODBUS relations, displayed in Figure 4-32 and Figure 4-33, follows the parameters described in Table 4-54.

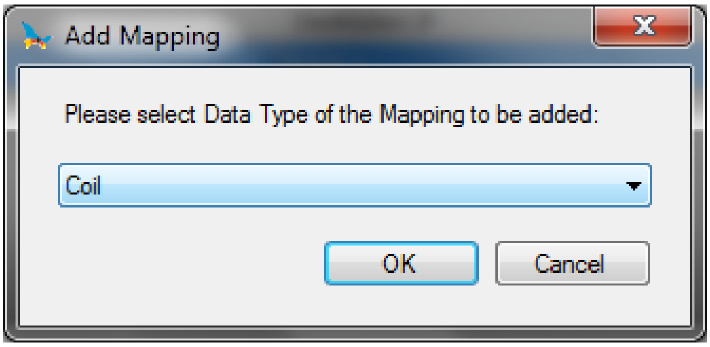


Figure 4-32. MODBUS Data Type

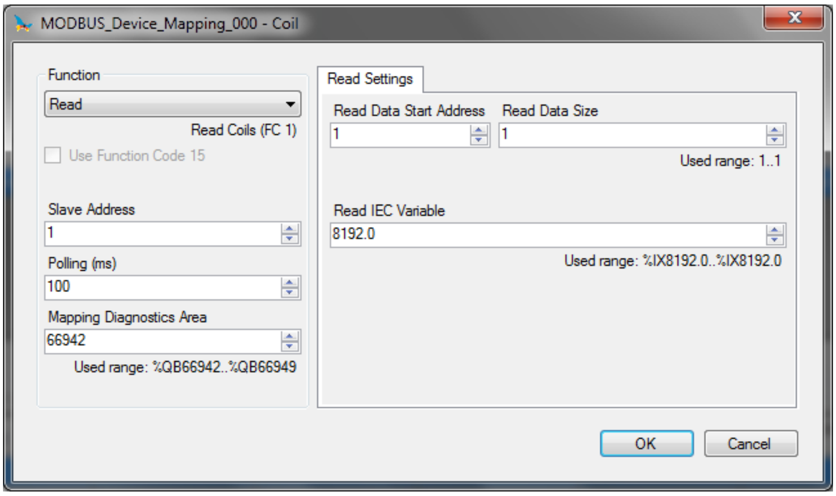


Figure 4-33. MODBUS Function

Configuration	Description	Default Value	Option
Function	MODBUS function type	Read	Read Write Read/Write Write Mask
Slave Address	The 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	Starting address of the read data MODBUS	1	1 to 65536
Read Data Size	Number of read data MODBUS	-	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

Write Mask of IEC Variables	Starting address of variables for write mask (%Q)	-	0 to 2147483644

Table 4-54. 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 TCP Gateway via TCP, and the same broadcasts the request to all network devices. When the address 0 is used, the client 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 chapter Technical Description - Specific Features. The configured MODBUS relations diagnostics are described in Table 4-50.

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.

Read IEC Variable: in case the MODBUS data type is Coil or Input Status (1-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-bit), 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 Technical Description chapter.

Write IEC Variable: in case the MODBUS data type is Coil or Input Status (1-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-bit), 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 Technical Description - Specific Features chapter.

Write Mask of IEC Variables: the Register Write Mask 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 chapter Technical Description - Specific Features.

Default Value: the factory default value cannot be set for the Mapping Diagnostic Area, IEC Read Variable, IEC Write Variable and Write Mask of 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 Reading/Writing 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.

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.

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 chapter.

MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol configuration initial screen (Figure 4-34), are defined as:

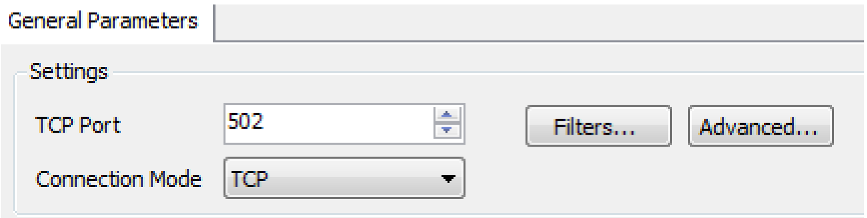


Figure 4-34. MODBUS Server General Parameters Configuration Screen

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

Table 4-55. 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. They are: 80, 8080, 1217, 1740, 1741, 1742,1743 and 11740.

The settings present on the "Filters ..." button, described in Table 4-56, 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 parameter IP filter 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 4-56. 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 Mask Filter and the IP address of the client. If the result is the same as the IP Filter for Writing, the client is entitled to write. For example, if the IP Filter for Writing = 192.168.15.0 and the Mask Filter for Writing = 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 Downtime Time-out.

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	10 to 3600

Table 4-57. 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.

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 4-58 (n is the value configured in the Initial Address of Diagnostics in %Q field):

Direct Representation Variable	Variable of type T_DIAG_MODBUS_ETH_SERVER_1 * Diagnostics		Size	Description
Diagnostic bits:				
%QX(n).0	tDiag.*	bRunning	BIT	The server is running
%QX(n).1		bNotRunning	BIT	The server is not running (see bit bInterruptedByCommand)

%QX(n).2		bInterruptedByCommand	BIT	The bit bNotRunning was enabled, because the server was interrupted by the user through the command bit
%QX(n).3		bConfigFailure	BIT	Discontinued diagnostic
%QX(n).4		bRXFailure	BIT	Discontinued diagnostic
%QX(n).5		bTXFailure	BIT	Discontinued diagnostic
%QX(n).6		bModuleFailure	BIT	Discontinued diagnostic
%QX(n).7		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		bRestart	BIT	Restart the server
%QX(n+2).2		bResetCounter	BIT	Reset diagnostics statistics (counters)
%QX(n+2).3		bDiag_19_reserved	BIT	Reserved
%QX(n+2).4		bDiag_20_reserved	BIT	Reserved
%QX(n+2).5		bDiag_21_reserved	BIT	Reserved
%QX(n+2).6		bDiag_22_reserved	BIT	Reserved
%QX(n+2).7		bDiag_23_reserved	BIT	Reserved
%QB(n+3)	byDiag_03_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)		wTimeoutClosedConnections	WORD	Connections counter, between the client and server, interrupted after a period of inactivity-time-out (0 to 65535).
%QW(n+8)		wClientClosedConnections	WORD	Connections counter interrupted due to customer request (0 to 65535).
%QW(n+10)		wRXFrames	WORD	Ethernet frames counter received by the server. An Ethernet frame can contain more than one request (0 to 65535).
%QW(n+12)		wRXRequests	WORD	Requests received by the server counter and answered normally (0 to 65535).
%QW(n+14)		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.
%QW(n+16)		wRXIllegalRequests	WORD	Illegal requests counter (0 to 65535)
%QW(n+18)		wDiag_18_reserved	WORD	Reserved

Table 4-58. MODBUS Server Diagnostics

Note:

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

Mapping Configuration – Configuration via Symbolic Mapping

The setting of the MODBUS Server mappings, visualized in Figure 4-35, follows the parameters described in Table 4-59.

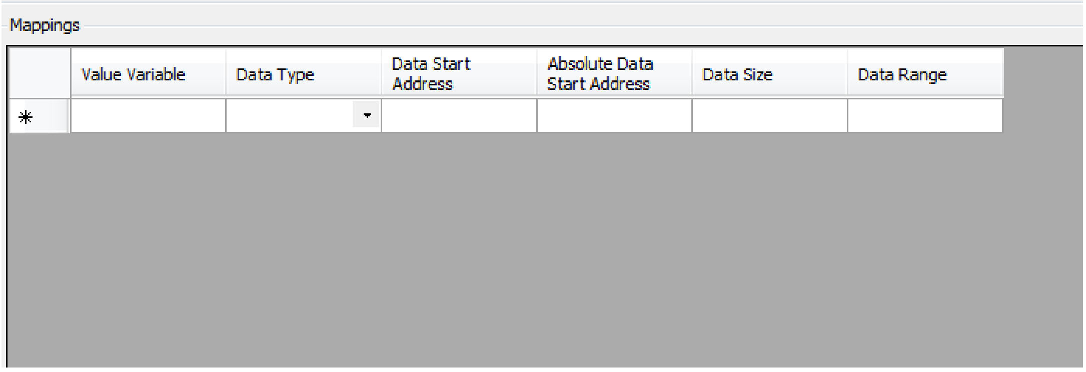


Figure 4-35. MODBUS Server Data Mappings Screen

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	The data range address configured	-	-

Table 4-59. 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.

Absolut 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 "Initial Address" and "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.

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 chapter.

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

The general parameters, found on the home screen of MODBUS protocol configuration (Figure 4-36), are defined below.

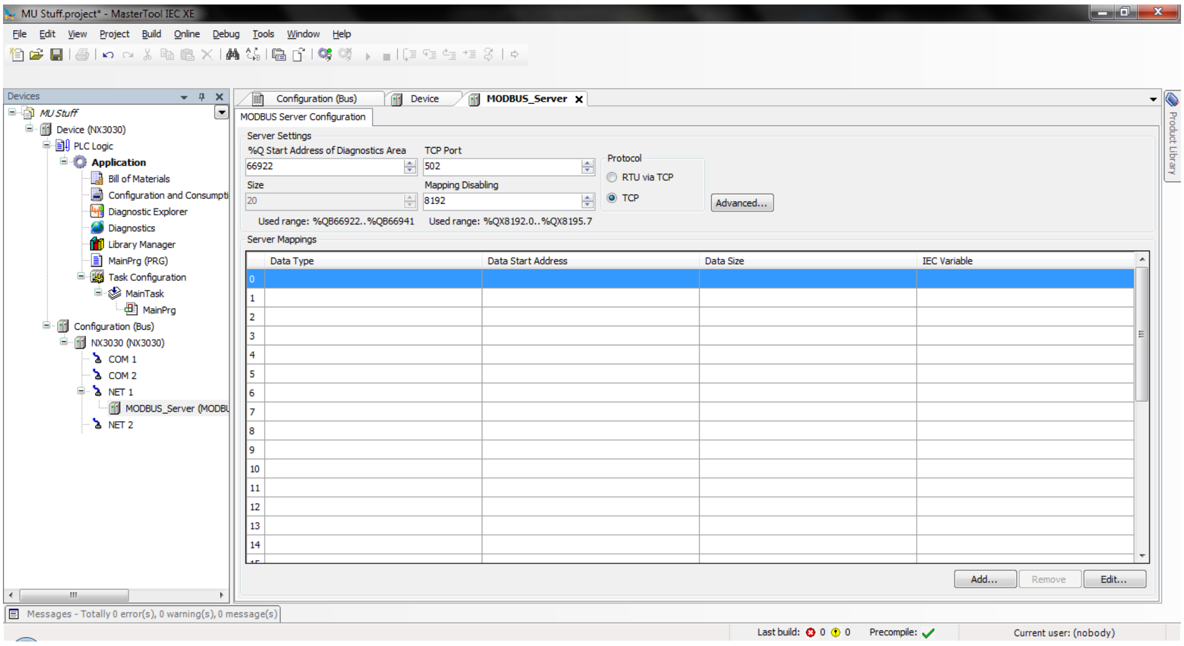


Figure 4-36. MODBUS Server Setup Screen

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 4-60. 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 chapter Technical Description - Specific Features.

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. They are: 80, 8080, 1217, 1740, 1741, 1742.1743 and 11740.

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.

Default Value: the factory default value cannot be set to the Initial Address of Diagnostics in %Q 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 communication times of the MODBUS Server protocol, found on the "Advanced ..." button of the configuration screen, are divided into: Task Cycle (ms) and Connection Downtime 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 4-58.

Mapping Configuration – Configuration via Direct Representation (%Q)

The MODBUS relation configuration, depicted on Figure 4-37 and Figure 4-38, follow the parameters described on Table 4-61:

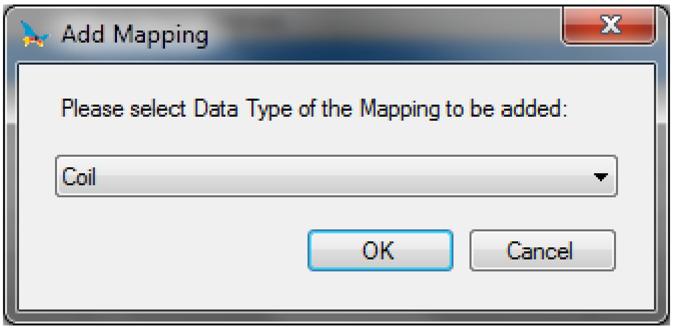


Figure 4-37. MODBUS Data Type

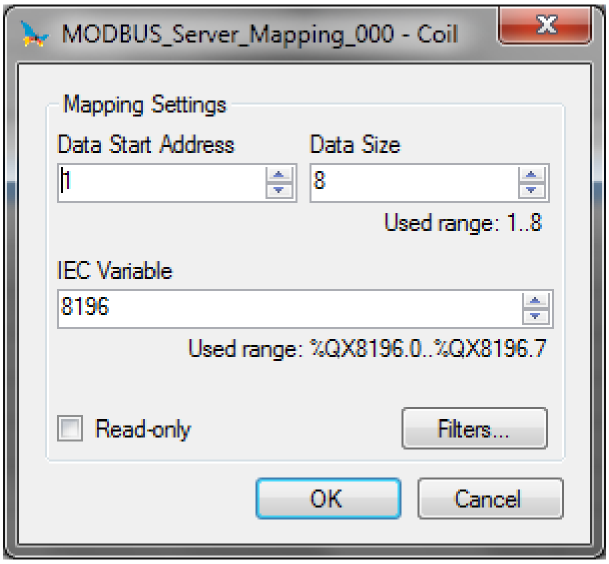


Figure 4-38. MODBUS Function

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 4-61. 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.

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 %Q10X.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 Technical Description - Specific Features chapter.

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: 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 applicative 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 configurations in the “Filters...” button, described on Table 4-62, are related to the TCP communication filters:

Configuration	Description	Default	Options
Write Filter IP Address	Specifies a IP interval with writing access to the declared variables in the MODBUS relation	0. 0. 0. 0	0.0.0.0 to 255.255.255.255
Write Filter Mask	Specifies a subnet mask and the parameter Write Filter IP	0. 0. 0. 0	0.0.0.0 to 255.255.255.255
Read Filter IP Address	Specifies a IP interval with reading access to the declared variables in the MODBUS relation	0. 0. 0. 0	0.0.0.0 to 255.255.255.255
Read Filter Mask	Specifies a subnet mask and the parameter Read Filter IP	0. 0. 0. 0	0.0.0.0 to 255.255.255.255

Table 4-62. IP Filters of Modbus Server

Note:

Filters: the filters are used to establish an IP addresses interval which have writing or reading access in the MODBUS relations, individually configured. The permission criteria is made through an AND logic operation between the Write Filter Mask and the client IP address. In case the result is the same as the Write Filter IP, the client has writing right. E.g. if the Write Filter IP = 192.168.15.0 and the Write Filter Mask = 255.255.255.0, then only clients with IP = 192.168.15.x will have writing right. The same proceeding is applied to the Read Filter parameters to define the reading 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 (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 Server MODBUS 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.

OPC DA

It's possible to communicate with the Nexto Series CPUs using the OPC DA (Open Platform Communication 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 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 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).

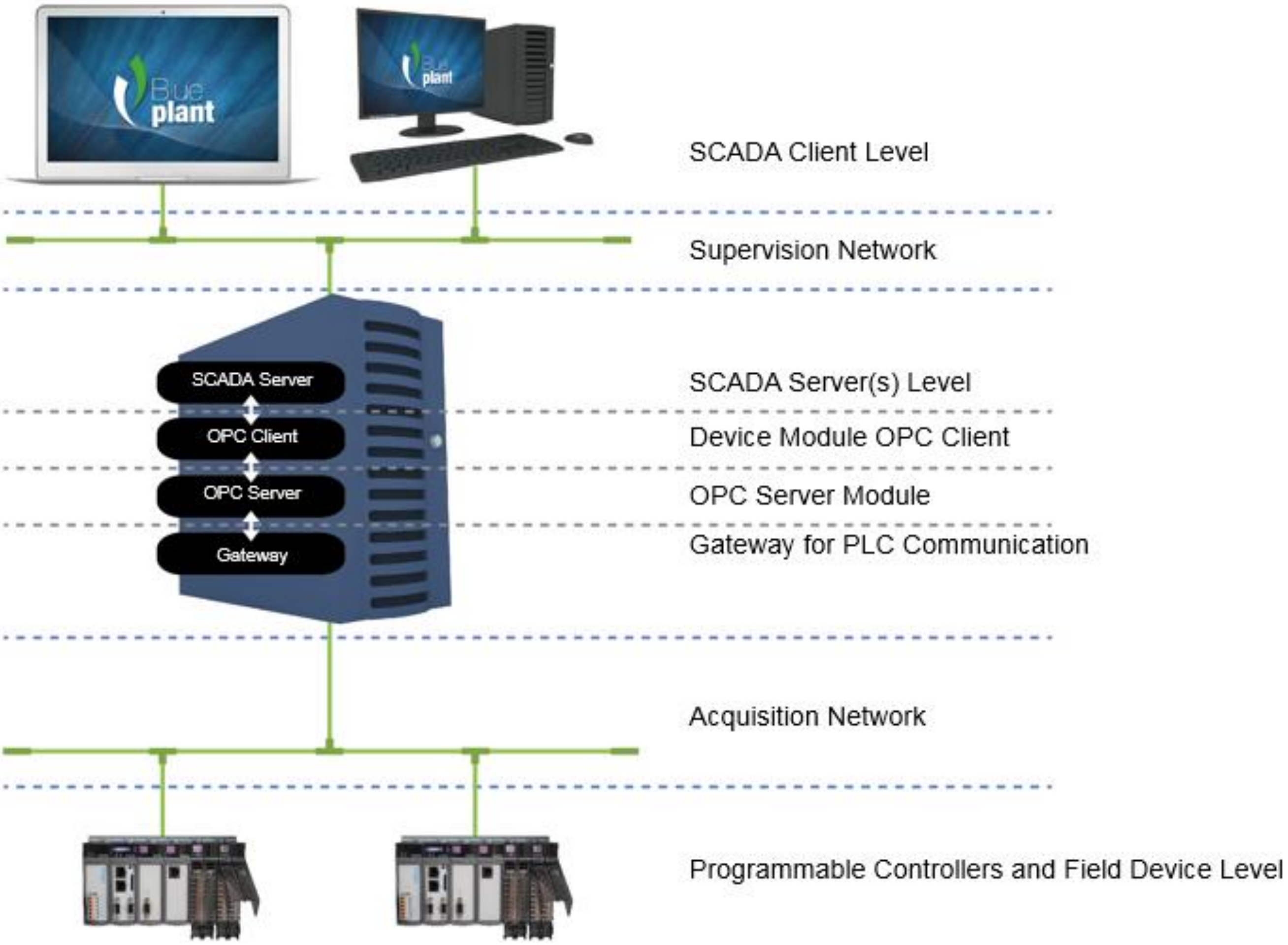


Figure 4-39. OPC Architecture

Figure 4-39 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 4-63.

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.
PLC Communication Gateway	A gateway enables the communication between the OPC 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 Modules	The OPC Server is a Module responsible of receiving the OPC DA requests and translate them to the communication with the field devices.
OPC Client Device Module	The OPC Client Device module is responsible for the requests to the OPC 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 4-63. Roles Description on an OPC 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 that 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 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.

Creating a Project for OPC Communication

Unlike the communication with drivers such as MODBUS and PROFIBUS DP, to set an OPC 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 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* e select the option.

ATTENTION:
The variables shown in the objects *IoConfig_Globals*, *IoConfig_Application_Mappings e IoConfig_Global_Mappings* are used internally for I/O control e 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 IEC61131-1. 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.

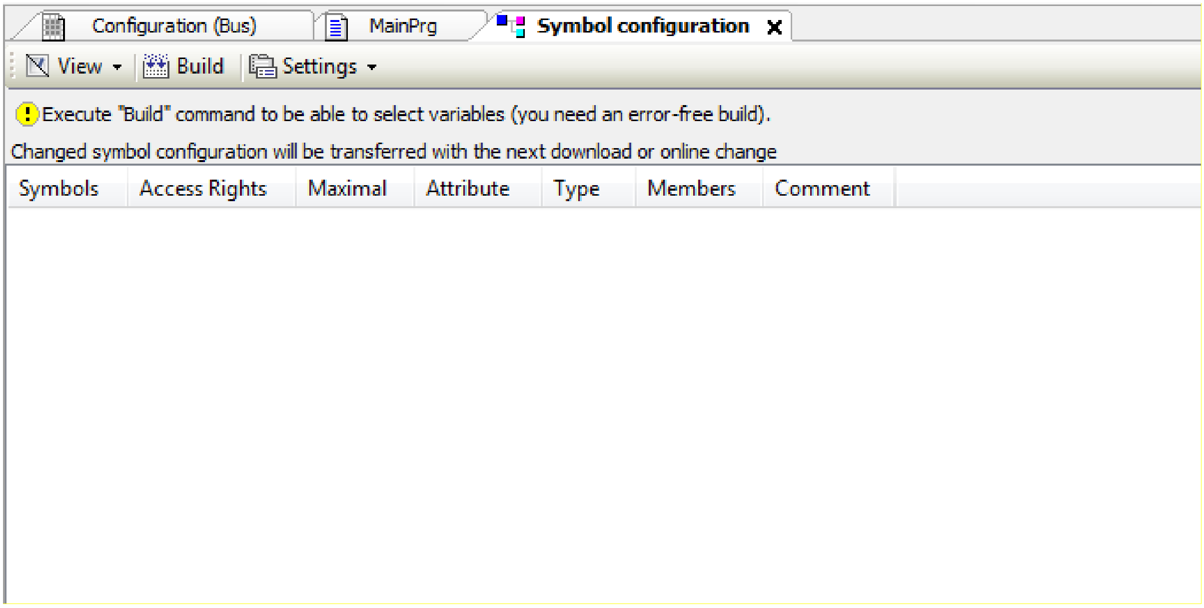


Figure 4-40. Symbol Configuration object

The Table 4-64 presents the descriptions of the *Symbol Configuration* object screen fields.

Field	Description
Symbols	Variable identifier that will be provided to the OPC Server
Access Rights	Indicate 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 Server.
Comment	Variable comment, inserted on the POU or GVL where the variable was declared. To show up as a variable comment here, the comment must be entered one line before the variable on the editor, while in text mode, or in the comment column when in tabular mode.

Table 4-64. Symbol Configuration object screen fields description

When altering the project settings, such as adding or removing variables, it's necessary to run the command *Build*, in order to refresh the list of variables. This command must be executed until the message in Figure 4-40 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 Server to be accessed by the Clients.

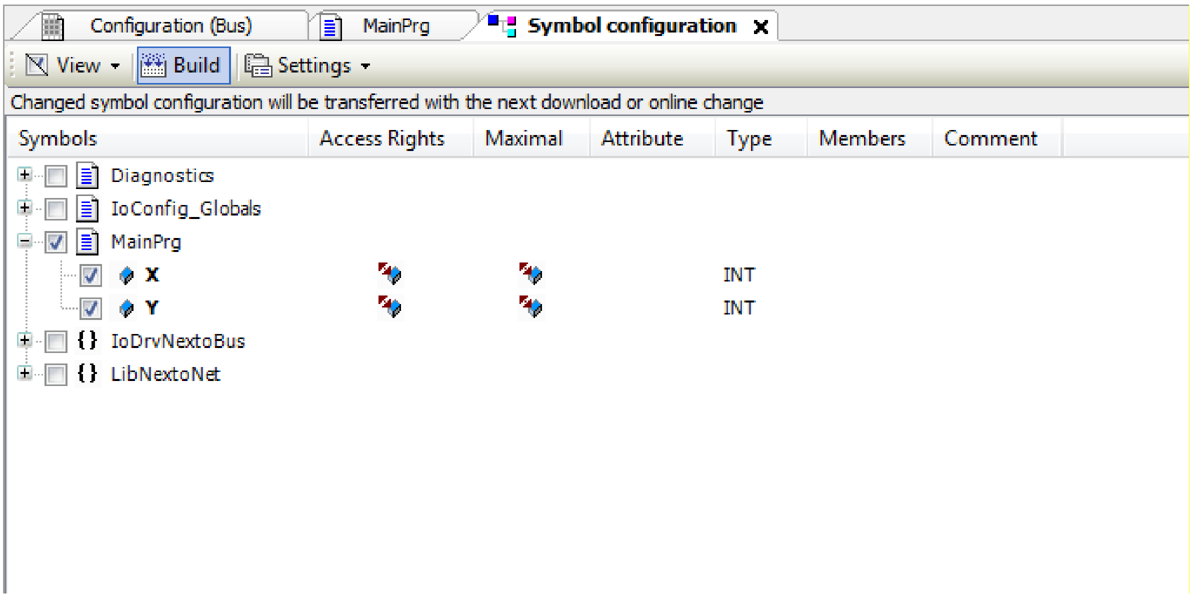


Figure 4-41. Selecting Variables on the Symbol Configuration

After this procedure, the project must be loaded into a PLC so the variables will be available for communication with the OPC 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 Server through an attribute inserted directly on the POUs or GVLs where the variables are declared. When the attribute {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 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 Server with read only access right.
- {attribute 'symbol' := 'write'} – when the attribute value is ‘write’, the variables will be available to the OPC Server with write only access right.
- {attribute 'symbol' := 'readwrite'} – when the attribute value is ‘read’, the variables will be available to the OPC Server with read and write access right.

In the following example of variable declaration, the variables A and B settings allow that an OPC 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 MainPrg
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 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.

Configuring a PLC on the OPC Server

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

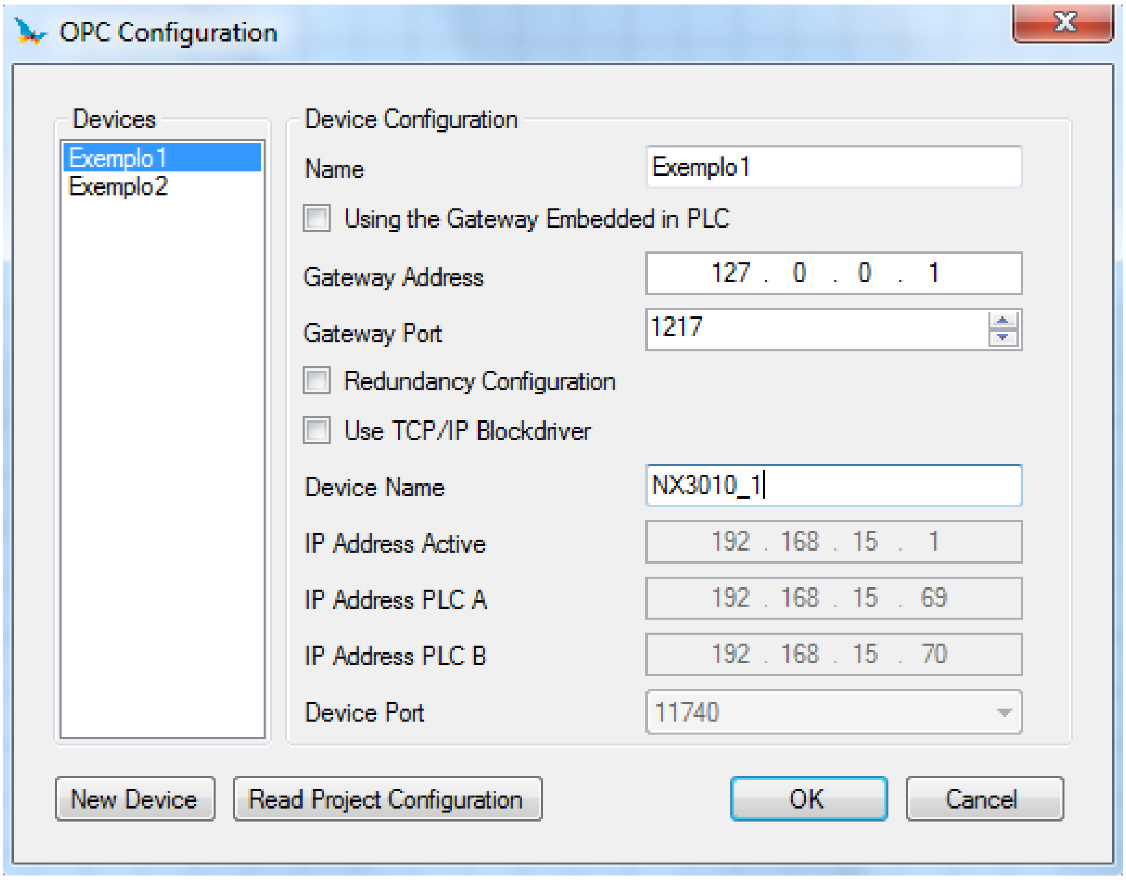


Figure 4-42. OPC Server Settings

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 Configuration**, 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 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 a 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 Server configurations.

Device Configuration	Description	Default Setting	Possibilities
Name	PLC description inside the OPC Server configuration file. This field can have any name, but for organizational purposes, it's	'PLC1'	This field is a STRING and it accepts alphanumeric (letters and numbers) characters and the "_" character. It's not allowed to initiate a

	recommended to use the project name that is loaded in the PLC.		STRING with numbers or with "_". It allows up to 49 characters.
Gateway Address	IP Address of the computer that the OPC 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 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 Setting* 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 CPA. 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 CPB. 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 4-65. Configuration Parameter of each PLC for the OPC Server

When a new PLC needs to be configured on the OPC Server, simply press the *New PLC* button and the configuration will be created. When the setup screen is accessed, a list of all PLCs already configured on the OPC 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 Server is 16.

If the automation architecture used specifies that the OPC 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 Server configuration in the same way as done in other situations.

ATTENTION: To store the OPC 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 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 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.

Importing a Project Configuration

Using the button *Read Project Configuration*, as shown in Figure 4 42, 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 NET1 Ethernet interface settings. The port for connection to the PLC is always assigned in this case as 11740.

Configuration with the PLC on the OPC Server with Connection Redundancy

It's possible to configure the OPC Server for it to operate with connection redundancy. This way, the OPC Server will communicate preferably with one PLC, but when, by any reason, it can't establish communication with this PLC, a second PLC, also configured, will be accessed. This configuration is especially important for the communication between SCADA systems and the Nexto Series PLCs with Half-Cluster redundancy, where there's a PLC in active state executing the process, and another PLC in stand-by state, ready to take control of the process if some kind of failure occurs.

The project setup in these cases is similar to what is described in Creating a Project for OPC Communication. However, when a Project is created with Redundant Half-Cluster and the communication with the supervisory system will be through the OPC Server, it's necessary to select the *OPC Redundancy Configuration* option as enabled during the MasterTool IEC XE Project Creation Wizard. By enabling this option, the project will create the code needed to run the communication with OPC connection redundancy.

In the redundant case, a variable is declared within the POU named *NonSkippedPrg*. This POU is executed in both PLCs, regardless of redundancy state. Within this POU, a BOOL type variable is created, used to control the connection with the OPC Server named *OPCRedundancyActive*. This variable can be accessed from any application point through the whole context, i.e. *Application.NonSkippedPrg.OPCRedundancyActive*. It is declared in the Symbol Configuration object with the right read only by the SCADA. When the value of the variable is TRUE, data is read by connecting with this PLC. This way, every time there is a status change among PLCs, the variable state will also change, remaining in the state TRUE in the PLC which is in the redundancy active state.

The *NonSkippedPrg* program code, in ST language, is as follow:

```
PROGRAM NonSkippedPrg
VAR
      {attribute 'symbol' := 'read'}
      OPCRedundancyActive : BOOL;
END_VAR

IF fbRedundancyManagement.m_fbDiagnosticsLocal.eRedState =
REDUNDANCY_STATE.ACTIVE THEN
      OPCRedundancyActive := TRUE;
ELSE
      OPCRedundancyActive := FALSE;
END_IF
```

The *NonSkippedPrg* program code can be edited as long as the user watch out not to change the above code. This code tests the state of redundancy and writes a BOOL type variable called *OPCRedundancyActive* with it. If the PLC is the active, the variable value is TRUE, otherwise it's FALSE. This variable receives the attribute 'symbol': = 'read' to allow the OPC Server to access the content and define where the information should be read.

If it's decided to add OPC communication after the creation of the project, it is possible to configure the OPC by adding the above code in the *NonSkippedPrg* program and adding the *Symbol Configuration* object to the project.

For the configuration of the redundant PLC on the OPC Server, it's necessary to enable the *Redundancy Configuration* option in the configuration screen as shown in Figure 4-42. When this option is selected, the option Use Driver Blocking TCP / IP will always be used. In addition, the *IP Address PLC A* and *IP Address PLC B* fields will be enabled as described in Table 4-65. These IP Addresses are configured in the same Ethernet interfaces within the PLC project with Half-Cluster redundancy. For ease of configuration when a redundant project is open, the *Read Project Configuration* button can be used.

ATTENTION:
The OPC Server connection redundancy is done through only one Server. For the cases in which a better data availability for the supervision systems is desired, a redundant SCADA Server architecture must be adopted. In this cases it isn't required any OPC Server configuration. Refer to the SCADA system documentations to see which configurations are needed for the operation of the redundant architecture.

OPC Communication Quality and Status Variables

For each PLC created in the OPC Server, status variables are generated, named*_CommState* and *_ComStateOK*. The *_CommState* variable indicates the communication between the OPC and the PLC state. This state can interpreted by the OPC Clients according to Table 4-66.

State	Value	Description
STATE_TERMINATE	-1	If the communication between the OPC 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 on the OPC 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 Server is connected.

		This is a transitory state during the connection.
STATE_NO_SYMBOLS	2	There are no symbols (variables) available in the PLC configured in the OPC 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 Server. This is a transitory state during the connection.
STATE_RUNNING	4	After the reading of the symbols (variables) the OPC 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 Server.
STATE_NO_CONFIGURATION	6	When the OPC configuration (stored in a OPCServer.ini file) has a wrong syntax, the variable value will be this. Generally, this behavior is not observed for the MasterTool IEC XE maintains this configuration valid.

Table 4-66. Description of the Communication states between OPC Server and the PLC

The_CommStateOk is a variable of the Bool type that indicates if the communication between the OPC 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 4-67. Description of the OPC Quality value

Table 4-67 presents the possible quality values. The OPC 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 Server, the Substatus and Limit fields shall be null e a Good Quality will be presented as the value 192 and a Bad Quality will be value 0.

OPC Server Communication Limits

To communicate with the OPC Server, there are some limitations that must be respected for the correct functioning of the application. There can't be more than 20.000 variables configured as available in the OPC Server for each PLC. Therefore, 20.000 variables is the maximum limit to communicate with a single PLC.

Furthermore, when configuring the variables to be available in the OPC Server, the quantity of declared variables in each POU or GVL can't exceed the limit of 5.000. Table 4-68 presents the OPC Server configuration limits.

Maximum number of variables communicating with a single PLC	20.000
Maximum number of variables declared in a single POU or GVL	5.000
Maximum number of PLCs in a OPC Server	16
Maximum number of simultaneous connections of an OPC Server in a single PLC	8

Table 4-68. OPC Server Communication Limits

ATTENTION:
The Maximum number of simultaneous connections of an OPC Server in a single PLC is shared with connections made with the MasterTool IEC XE. I.e. the sum of connections of OPC Server and MasterTool IEC XE should not exceed the maximum quantity defined in Table 4-68.

The communication between the OPC 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 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.

The approximate time of this initial stage, with 5.000 variables is about 2 minutes. In cases in which more variables are used, this time can raise to up to 4 minutes, depending on the data type and network settings.

Accessing data Through an OPC Client

After the configuration of the OPC 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 Server must be selected. In this case the name is CoDeSys.OPC.DA. The Figure 4-43 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*.

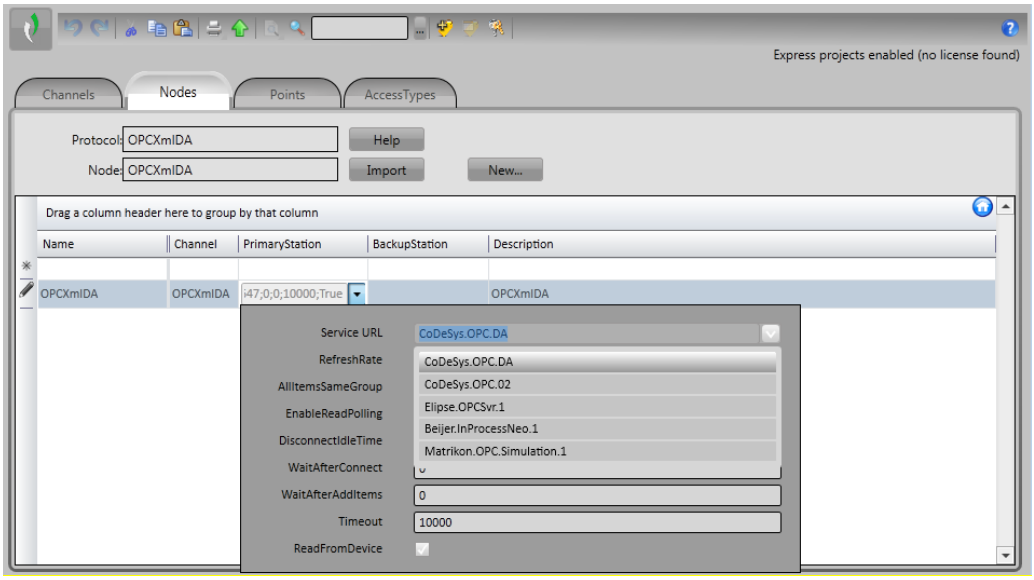


Figure 4-43. Selecting the OPC Server in the Client Configuration

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 t*unneller* 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 a **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.

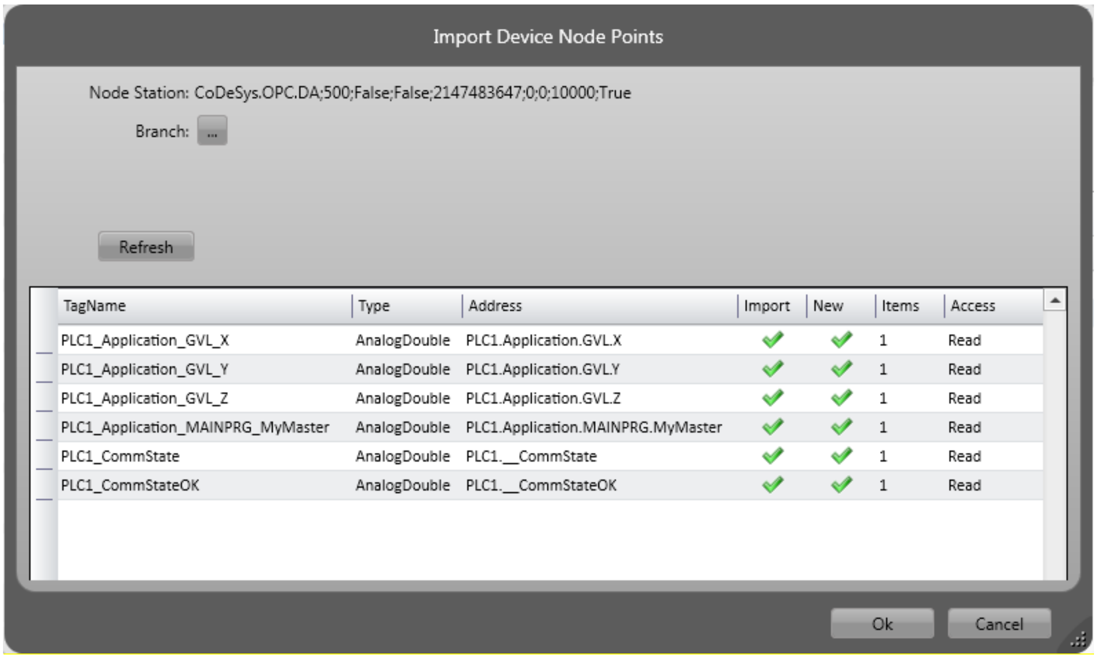


Figure 4-44. Symbols list consulted by the OPC Client

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 Master Tool IEC XE User Manual – MU299609.

EtherCAT

EtherCAT (Ethernet Control Automation Technology) is a master-slave architecture protocol with high performance, for deterministic Ethernet, that allows real time performance as it updates 1000 distributed I/O in 30µS or 100 servomotors axis each 100µS using twisted pair cables or optic fiber. Besides, it supports flexible topology, allowing for line, tree and/or star connections.

An Ethernet frame can be processed in real time instead of being received, interpreted and copied as process data in each connection. The FMMU (Fieldbus Memory Management Unit) in each Slave node reads the data that are addressed to it at the same time that the telegram is forwarded to the next device. In a similar way, the input data are inserted as the telegram is passed. Because of this, the frames are delayed just a few nanoseconds. Access on the Ethernet terminals can be made in any order as the data sequence is independent of the physical order. It can perform Broadcast, Multicast and between slaves communications.

The EtherCAT protocol allows a precise synchronization, that is required, for example, in applications where several axis simultaneously perform coordinated movements, it can be done through an exact adjust of the *Distributed Clock*. There's also the possibility to configure devices that, as opposed to synchronous communication, have an elevated tolerance degree inside the communication system.

The configuration of EtherCAT modules is initially determined by the *Device Description Files* of the Master and Slave devices used, and can be modified by the user in the *Configuration Editor* dialog boxes. However, for conventional applications and with the desire of an as easy as possible manipulation, large-scale configurations can be automated by choosing the *Autoconfiguration* mode in Master Parameters.

Note the possibility of modifying the Master and Slave configuration parameters also in operational mode, through the Master and Slave instances, according to the availability of the device in question.

Installing and inserting EtherCAT Devices

In order to be able to insert and configure EtherCAT devices as objects in the device tree, the Slave devices must be installed.

The Master device is automatically installed by the default MasterTool IEC XE installation. The EtherCAT Master defines which Slaves can be inserted.

To install the Slave devices the *Device Repository* must be opened, use the *EtherCAT XML Device description Configuration File (*.xml)* filter and select the device description files (*EtherCAT XML Device Description* / *ESI EtherCAT Slave Information*), supplied with the hardware. The Slave descriptions are available as XML files (file type: *.xml).

An EtherCAT Master can be added to the *Devices Tree* through the *Add Device* command, through the context menu of the CPU NET connectors.

Under a master, one or more slaves can be added, selecting an EtherCAT Master and running the *Add Device* command (context menu of the EtherCAT Master) or running the Scan command.

In addition to the line and tree topologies, MasterTool IEC XE also supports the star topology in EtherCAT. For the configuration of an EtherCAT star topology, special EtherCAT junctions are needed (in the example: Beckhoff EK1122). A modular star EtherCAT can be done using various junctions. Individual devices or a complete EtherCAT line section can be connected in the junction ports. A EherCAT junction is marked with the icon. The *Device Tree* example in Figure 4-45 shows different possibilities.

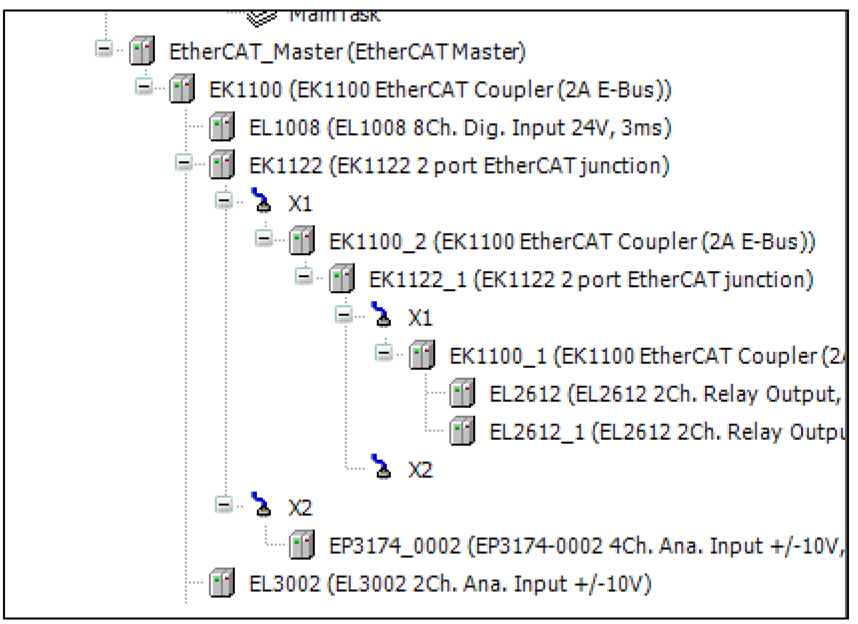


Figure 4-45. EtherCAT configuration example

ATTENTION:
- Only one EtherCAT Master instance per project is allowed.
- Only supported by the NX3020 and NX3030 CPU models.
- Only available on the NET connectors of the PLC.
- It cannot be used when the NETs are set as redundant.
- It cannot be used when Project has cluster redundancy.
- Other drivers cannot be instanced in the same NET port as the EtherCAT Master.
- It supports a maximum of 128 EtherCAT slaves per project.

Scan For Devices

The *Scan for Devices* command, available in the EtherCAT Master context menu, runs a search for the Slave devices physically installed in the EtherCAT network of the PLC currently connected. This means that with this command it's possible to detect and visualize the hardware components in the window presented in the figure bellow, allowing the user to map them directly in the project *Device Tree*.

It's noteworthy that, when the *Scan for Devices* command is selected, a connection with the PLC will be automatically established before the search begins and terminated when the search ends. So, for the first execution of this command, the Gateway connection must be configured and a program with the EtherCAT Master configured must be loaded into the PLC. In case of future additions of Slave devices, in order to run this command, it's necessary that the EtherCAT network is stopped. To do this, put to TRUE the *bStopBus* bit, present in the variables of the EtherCAT Master Diagnostics.

When the command is executed, the *Scanned Devices* field will contain a list of all devices and modules found during the last scan. To add them to the project, just click on the button *Copy All Devices To Project*. It's also possible to perform a comparison of the devices found in the search with the ones in the project by selecting the box *Show differences to project*.

If you add an EtherCAT Master module to the Project and use the *Scan for Devices* command, you will have a list of all the available EtherCAT Slaves. Entries in bold will be shown, if there's more than one device with the same description. With a double click on the entrance a list will open, and so the desired device can be selected.

After completing the changes in the EtherCAT network configuration, it’s necessary to do a new project download, for the changes to take effect.

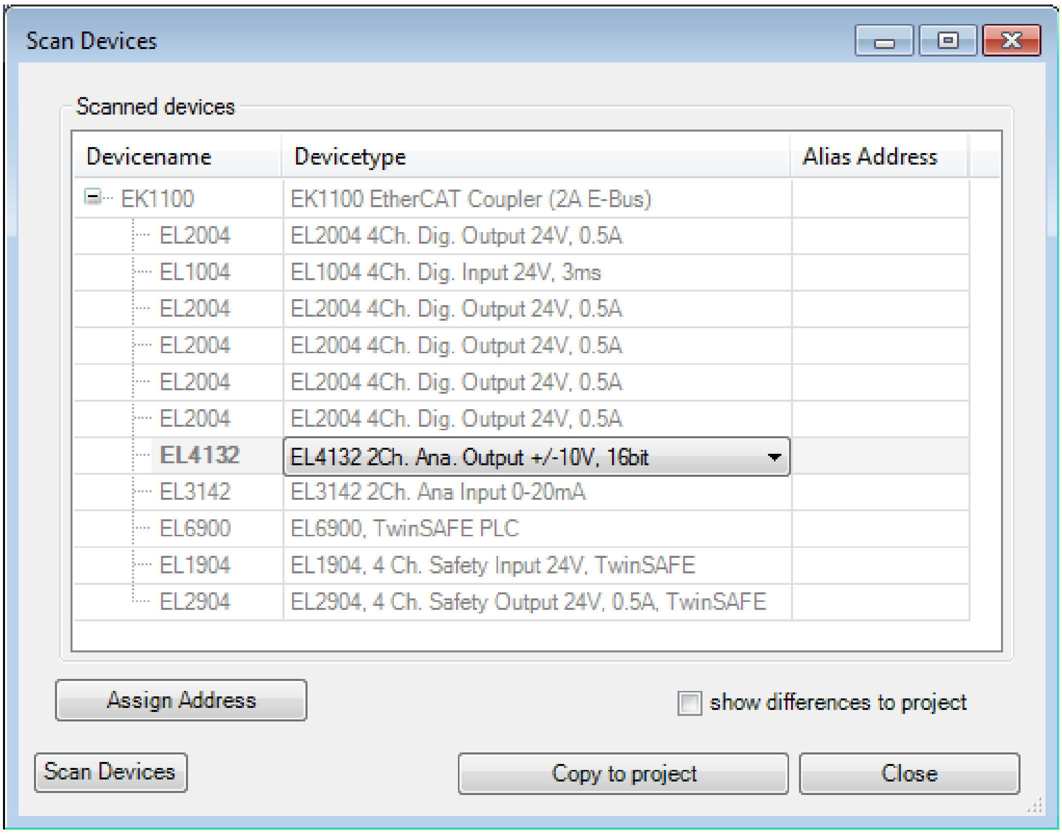


Figure 4-46. EtherCAT Devices Search Dialog

Diagnostic Variables

By inserting an EtherCAT Master and Slave, a diagnostic variable is added for the device in the GVL *Diagnostics*. This variable provides information on the device status. There are two types of variables, one for the EtherCAT Master and one for the EtherCAT slaves. Each variable has specific information about the device. The diagnostic and commands supplied are described in the following tables.

Variable DG_EtherCAT_Master*	Type	Possible Values	Description
tDiag.bRunning	BOOL	FALSE or TRUE	If this variable is TRUE, the transfer of all configuration parameters was successfully completed without errors found and the bus was not stopped by user command. Communication is running.
tDiag.bError	BOOL	FALSE or TRUE	This variable will be TRUE if an error is detected during startup of the EtherCAT stack or during operation, communication with the slaves is stopped if any more messages can be received (e.g. due to a fault in the cable). The cause of the error can be discovered with the aid of the logs list via error STRING.
tDiag. eLastErrorCode	ETC_LASTERROR		The information regarding the possible values for this diagnostic, as well as it description, can be found in Table 4-70.
tDiag.bDistributedClock	BOOL	FALSE or TRUE	If DC is used, the PLC will be synchronized with

InSync			the first EtherCAT slave whose DC setting is active. This variable is TRUE shortly after this synchronization is successfully completed. This signal, for example, can be used to initialize SoftMotion function blocks in case of compatibility with the device after the CLP is in synchronized mode, otherwise jumps in position may occur. At the PLC startup this variable is FALSE and will change to TRUE after a few seconds. If synchronization is lost due to any failure, the variable will change to FALSE.
tDiag.bReserved_00	BOOL	-	Reserved space.
tDiag.bReserved_01	BOOL	-	Reserved space.
byReserved_00	BYTE	-	Reserved space.
tCommand.bRestart	BOOL	FALSE or TRUE	In rising edge, the master will restart completely. All configuration parameters will be reloaded.
tCommand.bStopBus	BOOL	FALSE or TRUE	When this variable is TRUE, the communication is stopped. EtherCAT packages will not be sent. On most devices a restart is required, because they are in an error state.
tCommand. wDCInSyncWindow	BOOL	0..65535	Time window to bDistributedClockSync. Jitter must be inside this window so that the bDistributedClockSync stays TRUE. Default value: 50 microseconds.
tCommand. bySlaveUpdatedbyCycle	BOOL	0..128	This value defines the number of slaves that will be read each cycle to fill the slaves diagnostic variables. Value 0 means that no slave diagnostic will be updated.
tCommand. bReserved_00	BOOL	-	Reserved space.
tCommand. bReserved_01	BOOL	-	Reserved space.
byReserved_01	BYTE	-	Reserved space.

Table 4-69. EtherCAT Master Diagnostics and Commands

Code	Enum	Description
00	NO_ERROR	No error, running.
01	NO_COMM	No communication. Over 100 packages weren’t received. Possible failure in the master cable.
02	WRONG_WORKING_COUNTER	Working counter to processdata is wrong. One or more slaves aren’t operational or missing and the expected working counter isn’t found.
03	DC_TIME_ZERO	Slave DC Time is always zero -> maybe IN and OUT connectors are wrong and no time can be read from the time reference.
04	OPEN_FIRSTADAPTER_FAILED	First network adapter can’t be opened.
05	OPEN_SECONDADAPTER_FAILED	Second network adapter can’t be opened.
06	ADAPTER_MISMATCH	Second network adapter uses the MAC-ID as first interface.
07	NO_SLAVES_FOUND	Error in slaves startup: There’s possibly missing slaves or with no communication.
08	VENDOR_ID_WRONG	VendorID is not equal.
09	PRODUCT_ID_WRONG	ProductID is not equal.
10	NUMBER_DEVICE_MISMATCH	Reading ProductID or Vendor ID is unsuccessful, more slaves in the configuration than in real architecture.
11	SDO_WRITE_ERROR	SDO writing error during startup.
12	SDO_TIMEOUT	SDO timeout during startup.
13	EMERGENCY_RECEIVED	Emergency received from the device.
14	IDN_WRITE_ERROR	IDN writing error during startup.
15	IDN_TIMEOUT	IDN timeout during startup.

Table 4-70. EtherCAT Master Error Codes

Variable DG_Slave*	Type	Possible Values	Description
tDiag.wState	ETC_SLAVE_STATE	ETC_SLAVE_BOOT=3 ETC_SLAVE_INIT=1 ETC_SLAVE_PREOPERATIONAL=2 ETC_SLAVE_SAVEOPERATIONAL=4 ETC_SLAVE_OPERATIONAL=8	Current Slave State.
tDiag. dwVendorID	BOOL	Any DWORD	After the EtherCAT stack initialization, this variable return the VendorID read from the slave.
tDiag. dwProductID	ETC_LASTERROR	Any DWORD	After the EtherCAT stack initialization, this variable returns the ProductID read from the slave.
tDiag. dwRevisionID	BOOL	Any DWORD	After the EtherCAT stack initialization, this variable return the RevisionID read from the slave.
tDiag. tLastEmergency.	ETC_CO_Emergency		If a message is received then this information is stored in the slave and it can be read in the application through this variable. Also a log message is added. More information about this diagnosis is found in Table 4-72.
tDiag.bReserved_01	BOOL	-	Reserved space.
byReserved_00	BYTE	-	Reserved space.

Table 4-71. EtherCAT Slave Diagnostics

Variable DG_Slave.tDiag.tLastEmergency.*	Type	Hexa Code	Description
wErrorCode	WORD	00XX	Reset Error or No Error.
		10XX	Generic Error.
		20XX	Current.
		21XX	Current, inside the device
		22XX	Current inside the device.
		23XX	Current, outside the device.
		30XX	Voltage .
		31XX	Main Voltages.
		32XX	Voltage inside the device.
		33XX	Output voltage.
		40XX	Temperature.
		41XX	Ambient Temperature.
		42XX	Device Temperature.
		50XX	Device Hardware.
		60XX	Device Software.
		61XX	Internal Software.
		62XX	User Software.
		63XX	Data Set.
		70XX	Additional Modules.
		80XX	Monitoring.
		81XX	Communication.
		82XX	Protocol Error.
		8210	PDO not processes due to length error.
		8220	PDO length exceeded.

		90XX	External Error.
		A000	Unsuccessful PRE- OPERATIONAL to SAFE-OPERATION transition.
		A001	Unsuccessful SAFE- OPERATIONAL to OPERATION transition.
		F0XX	Additional Functions.
		FFXX	Device Specific.
byErrorRegister	BYTE	Faixa BYTE	Register error.
tDiag.tLastEmergency.abyErrorField	ARRAY[0..5] OF BYTE	0000-9FFF	Manufacturer Specific Error Field.
		A000-EFFF	Diagnostic Data.
		F000-FFFF	Manufacturer Specific Error Field.

Table 4-72. ETC_CO_Emergency Content

EtherCAT Master Settings

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

Master Parameters

Below are the general parameters found in the initial screen of the EtherCAT Master configuration, Figure 4-47.

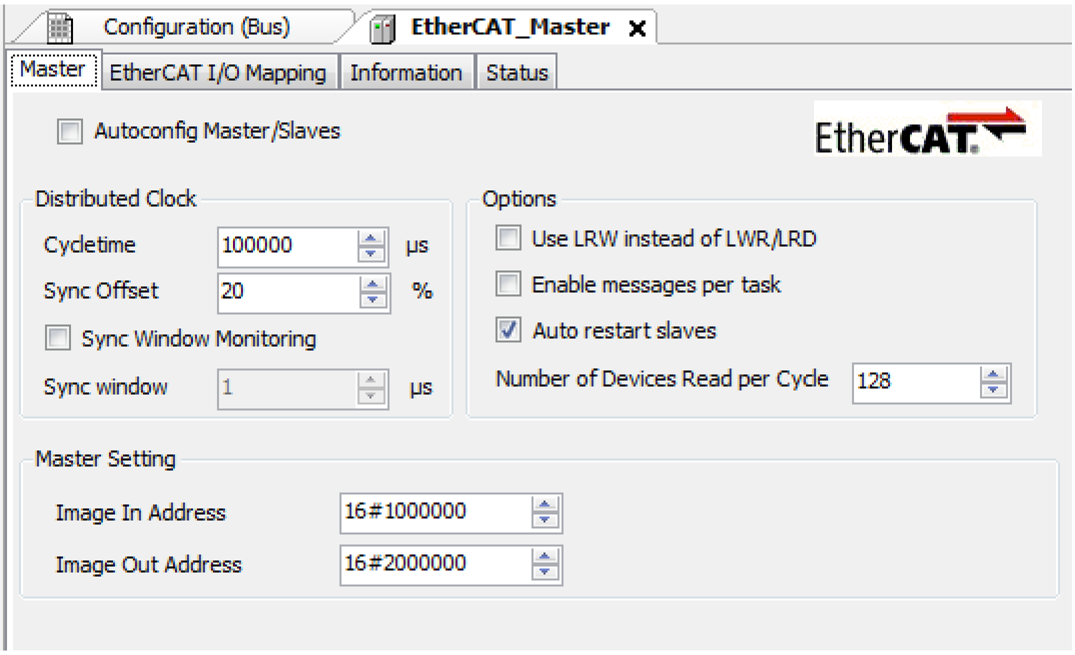


Figure 4-47. EtherCAT Master Configuration Dialog

Device Configuration	Description	Factory Default	Possible Values
Autoconfig Master/Slaves	Enable the Master and Slave automatic configuration.	Marked	Marked Unmarked
Cycletime [µs]	Sets the time period in which a new data telegram must be send to the bus.	100000	2000 to 1000000
Sync Offset [%]	Adjust the offset, from the PLC cycle, of the EtherCAT Slave synchronization interrupt.	20	-50 to 50

Sync Window Monitoring	If enabled, this option allows monitoring the Slave synchronization.	Unmarked	Marked Unmarked
Synch Window [µs]	Time for the Sync Window Monitoring.	1	1 to 32768
Use LRW instead of LWR/LRD	Enabling of the combined read and write commands.	Unmarked	Marked Unmarked
Enable Messages per Task	If enabled, the read and write commands that are dealing with input and output messages can be done in different tasks.	Unmarked	Marked Unmarked
Auto Restart Slaves	Restart the devices when the communication is aborted.	Marked	Marked Unmarked
Number of Devices Read per Cycle	Sets the number of Slaves that are read per cycle to fill the diagnostic variables. The value '0' means that no Slave diagnostic will be updated.	128	0 to 128
Image In Address	First input address of the first Slave.	16#1000000	16#1 to 16#1F000000
Image Out Address	First output logic address for the first Slave.	16#2000000	16#1 to 16#1F000000

Table 4-73. EtherCAT Master Configuration

Notes:

Autoconfig Master/Slaves: If this option is enabled, most of Master and Slave configuration will be made automatically, based on the description files and implicit calculations. In this case, the *FMMU / Sync* dialog will not be available. If it's unchecked the *Image In Address and Image Out Address* options will be available to the user.

ATTENTION: The Autoconfig mode is enabled by default and usually enough and highly recommended for standard applications. If it's disabled, all configuration definitions will have to be made manually, and thus, some specialized knowledge is required. To configure a Slave-to-Slave communication, the Autoconfiguration option must be disabled.

Cycletime: Time period after which, a new data telegram must be sent to the bus. If *Distributed Clock* functionality is enabled, the value of this parameter will be transferred to the Slaves clocks. This way, a precise data exchange synchronization can be achieved, which is especially important in cases where the distributed process demands simultaneous actions. So, a very precise time base, with a jitter significantly smaller than a microsecond, for all the network can be achieved.

Sync Offset: This value allows the adjustment of the offset of the EtherCAT Slave synchronization interrupt to the PLC cycle. Normally, the PLC task cycle begins 20% later than the Slaves synchronization interruption. This means that the PLC task can be delayed by 80% of the cycletime and no message will be lost.

Sync Window: If the synchronization of all Slaves are inside this time window, the EtherCAT Master *bDistributedClockInSync* diagnostic will be set to TRUE, otherwise it will be set to FALSE.

When *Distributed Clock* is used, it's highly recommended to use a dedicated task with high priority as the *Bus Cycle Task* of the EtherCAT Master. To do this, it's necessary to use Project Profiles that allows the creation of new tasks, then create a cyclic task with priority 0 (real time task) and link it to the master *Bus Cycle Task* on the *EtherCAT I/O Mapping* tab of the EtherCAT Master. The user can also change the value of the *wDCInSyncWindow* variable, configuring the maximum jitter allowed on the synchronization between master and slaves.

Use LRW instead of LWR/LRD: Activating this option enables the Slave-to-Slave communication because, instead of using separated reading (LRD) and write (LWR) commands, combined reading/writing (LRW) commands will be used.

Auto Restart Slaves: By enabling this option, the Master will restart the Slaves as soon as the communication is aborted, so, the *bError* EtherCAT Master diagnostic in the *Diagnostic* GVL won't go to TRUE.

Image In Address and Image Out Address: These definitions can only be edited if the Autoconfig mode is disabled, otherwise this will be done automatically and this parameter will be invisible.

Diagnostics Message: it shows some information or error messages from the stack. The messages are also logged in the PLC *Log* Tab, access through the *Device* icon in the *Device Tree*. This option is only visible when the EtherCAT Master is online.

Bus Load: This value shows the busload on the network adapter (0 – 100%). This option is only visible when the EtherCAT Master is online.

I/O Mapping

This EtherCAT Master configuration editor tab offers the possibility to assign the project variables to its corresponding EtherCAT inputs and outputs. Thus, the EherCAT Master variables can be controlled by *User Application*.

Furthermore, it's possible to choose which task the *Bus Cycle Task* will use, through the options in the selection list. This task serves to do the EtherCAT Master operations. *MainTask* is the default option of this field.

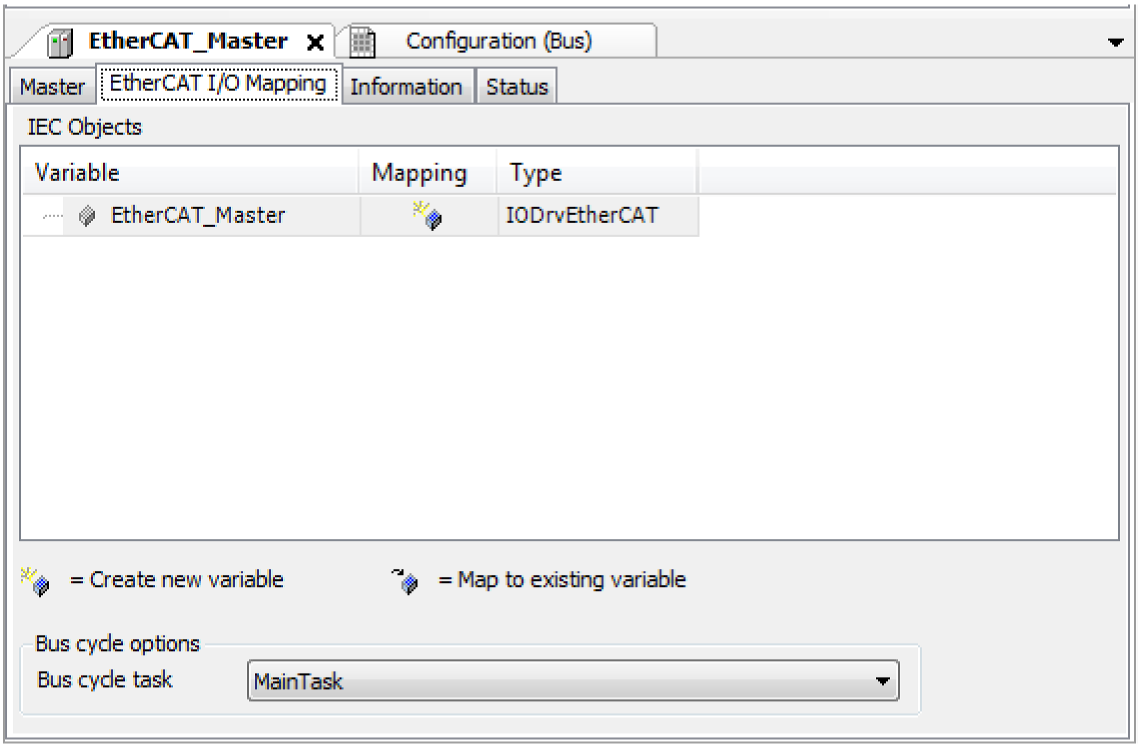


Figure 4-48. I/O Mapping Dialog

Status and Information Tabs

The *Status* tab of the EtherCAT Master configuration editor provides status information (e.g. 'Running', 'Stopped') and diagnostic messages specific of the device and the internal bus system.

The *Information* tab, present on the EtherCAT Master configuration editor, shows, if available, the following general information about the module: Name, Vendor, Type, Version Number, Category, Order Number, Description, Image.

EtherCAT Slave Configuration

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

Slave Parameters

Below are presented the general parameters found in EtherCAT Slave configuration initial screen. This field is only available if the Autoconfig mode (Master) isn't enabled.

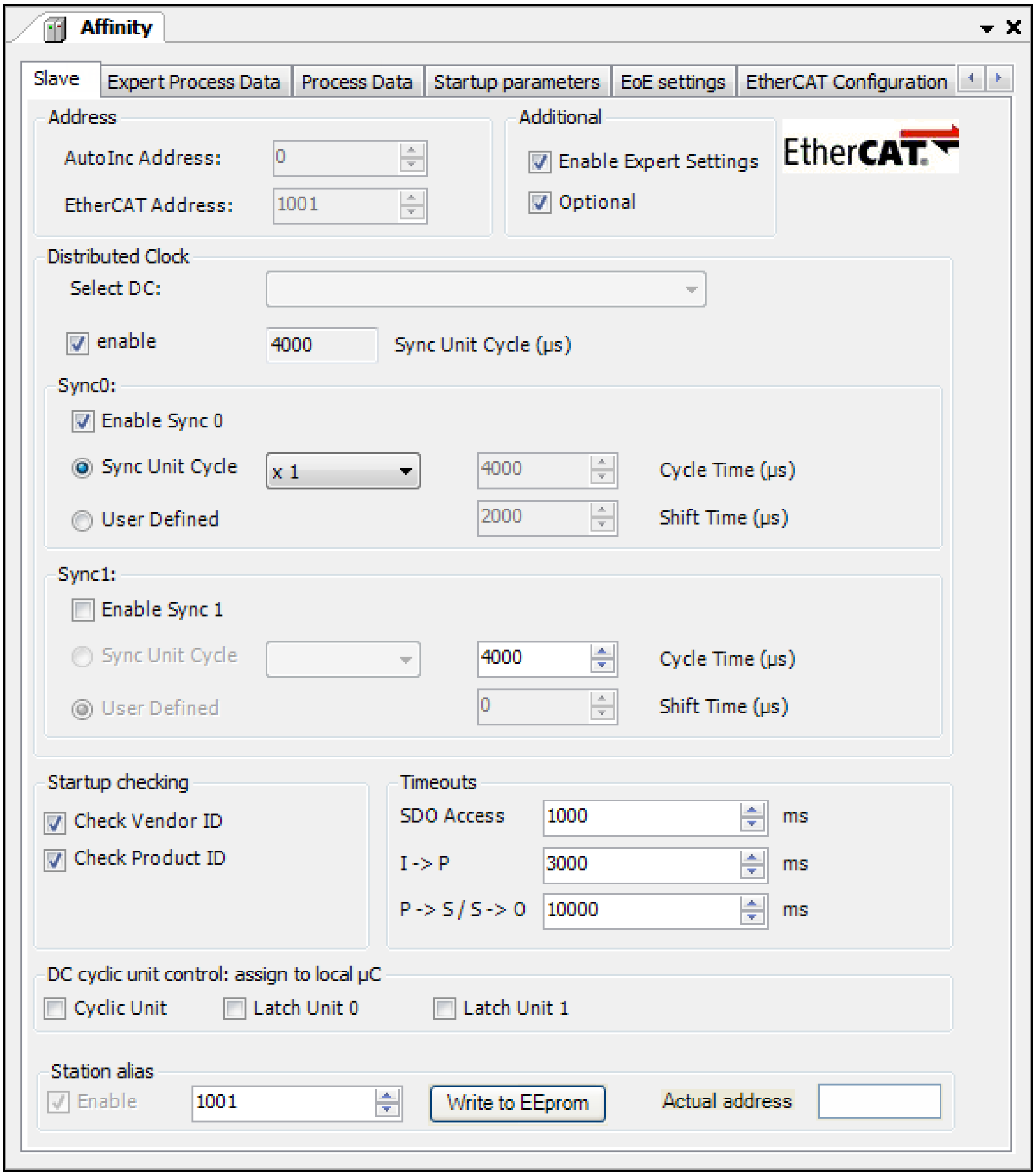


Figure 4-49. EtherCAT Slave Configuration Dialog

Device Configuration	Description	Default Value	Possibilities
AutoInc Address	Autoincremental Address (16-bit) defined by the Slave position in the	-	-65535 to 0

Device Configuration	Description	Default Value	Possibilities
	network.		
EtherCAT Address	Slave final address, assign by the Master during startup. This address is independent from the position in the network.	-	1 to 65535
Enable Expert Settings	Enable the Slave advanced Settings options.	Unmarked	Marked Unmarked
Optional	Declare the Slave as Optional.	Unmarked	Marked Unmarked
Select DC	Show all Distributed Clock configurations provided by the device description file		
Enable Distributed Clock	Enable the Distributed Clock configuration options.	Unmarked	Marked Unmarked
Sync Unit Cycle [µs]	Show the Cycle Time set in Master	100000	2000 to 1000000
Enable Sync 0	Enable the Sync 0 synchronization unit configurations.	Unmarked	Marked Unmarked
Sync Unit Cycle (Sync 0)	By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle.	Unmarked	Marked Unmarked
User Defined (Sync 0)	If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time (µs) field.	Unmarked	Marked Unmarked
Cycle Time [µs] (Sync 0)	Show the cycle time currently set.	100000	1 to 2147483647
Shift Time [µs] (Sync 0)	Time between the sync events and the “Output Valid” or “Input Latch” time.	0	-2147483648 to 2147483647
Enable Sync 1	Enable the Sync 1 synchronization unit configurations.	Unmarked	Marked Unmarked
Sync Unit Cycle (Sync 1)	By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle.	Unmarked	Marked Unmarked
User Defined (Sync 1)	If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time (µs) field.	Unmarked	Marked Unmarked
Cycle Time [µs] (Sync 1)	Show the cycle time currently set.	100000	1 to 2147483647
Shift Time [µs] (Sync 1)	Time between the sync events and the “Output Valid” or “Input Latch” time.	0	-2147483648 to 2147483647
Check Vendor ID	If unmarked, it will disable the Vendor ID Check	Marked	Marked Unmarked
Check Product ID	If unmarked, it will disable the Product ID Check	Marked	Marked Unmarked

Device Configuration	Description	Default Value	Possibilities
SDO Access	Set a time reference (in microseconds) for the timeout check of a SDO Access.	-	0 to 100000
I -> P (Timeouts)	Set a time reference (in microseconds) for the timeout check of the switch from Init to Pre-Operation mode.	-	0 to 100000
P -> S/S -> O	Set a time reference (in microseconds) for the timeout check of the switch from Pre-Operation and to Safe-Operation and from Safe-Operation to Operational modes.	-	0 to 100000
Cycle Units	Set the Unit Cycle to the local microprocessor.	Unmarked	Marked Unmarked
Latch Unit 0	Set the Latch Unit 0 to the local microprocessor.	Unmarked	Marked Unmarked
Latch Unit 1	Set the Latch Unit 1 to the local microprocessor.	Unmarked	Marked Unmarked

Table 4-74. EhterCAT Slave Configurations

Notes:

AutoInc Address: This address is used only during startup, when the Master is assigning the EtherCAT addresses to the Slaves. When, for this matter, the first telegram runs through the Slaves, each run-through Slave increases its AutoInc by 1. The Slave with address 0 finally will receive the data.

Optional: If a Slave is declared as *Optional*, no error message will be created in case the device doesn't exist in the bus system. Thus a *"Station alias"* address must be defined and written to the EEprom. This option is only available if the option *"Autoconfig Master/Slaves"* in the settings of the EtherCAT Master is activated and if this function is supported by the EtherCAT Slave.

Enable Distributed Clock: If the "Distributed Clock" functionality is enabled, the data exchange cycle time, displayed in the Sync Unit Cycle (µs) field will be determined by the Master Cycle Time. Thus the master clock can synchronize the data exchange within the network.

Enable Sync 0: If this option is activated, the *Sync0* synchronization unit is used. A synchronization unit describes a set of process data which is exchanged synchronously.

Sync Unit Cycle (Sync 0): If this option is activated, the *Master Cycle Time*, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The "*Cycle Time (µs)*" field shows the currently set cycle time.

Shift Time (µs): The Shift Time describes the time between the sync events (*Sync0*, *Sync1*) and the *Output Valid* or *Input Latch* times. Writable value, if the slave supports shifting of *Output Valid* or *Input Latch*.

Enable Sync 1: If this option is selected, the synchronization unit *Sync1* is used. A synchronization unit is a set of process data which is exchange synchronously.

Sync Unit Cycle (Sync1): If this option is activated, the *Master Cycle Time*, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The "*Cycle Time (µs)*" field shows the currently set cycle time.

Current State: The current state is displayed. Possible values: Init, Pre-Operational, Safe Operational and Operational. If the state is Operational, the slave configuration has been terminated correctly.

Check VendorID and ProductID: By default, at startup of the system the Vendor ID and/or the Product ID will be checked against the current configured settings. If a mismatch is detected, the bus will be stopped and no further actions will be executed. This serves to avoid the download of an erroneous configuration. This option is intended to switch off the check, if necessary.

SDO Access: By default there's no timeout set for the SDO list submit action at system startup. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

I->P: By default there's no timeout set for the state transition between *Init* to *Pre-Operational*. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

P -> S / S-> O: By default there's no timeout set for the state transition between *Pre-Operational* to *Safe-Operational* and between *Safe-Operational* to *Operational*. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

DC cycle unit control: Choose the desired option(s) concerning the *Distributed Clock* functions in order to define, which should be assigned to the local microprocessor. The control is done in register 0x980 in the EtherCAT slave: The possible settings: *Cyclic Unit, Latch Unit 0, Latch Unit 1*.

Station Alias: These settings are only visible if the option *Optional* is activated or if the slave device supports alias addresses (defined in the *Device Description File*).

Enable: If the setting *Optional* is not activated, this setting can be activated if explicitly supported by the device description of the slave. It allows direct assignment of an alias address in order to get the slaves address independent of its position within the bus. If the option *Optional* is activated, this checkbox is disabled.

Write to EEPROM: This command is only visible in online mode. It allows to write the defined address to the EEPROM of the slave. If not supported by the slave this command will have no effect and the device will not work as *Optional Slave*.

Actual Address: This field, only visible in online mode, displays the actual address of the slave. It can be used to check the success of the *Write to EEPROM* command.

FMMU/Sync

This dialog will only be provided on a tab of an EtherCAT Slave configuration editor if the *Autoconfig Mode* in the Master is disabled. It shows the FMMUs and Sync Managers of the slave as defined by the device description file. These settings may be reworked, for example for configuring a slave-to-slave communication.

ATTENTION: These are Expert Settings, which usually are not necessary for standard applications.

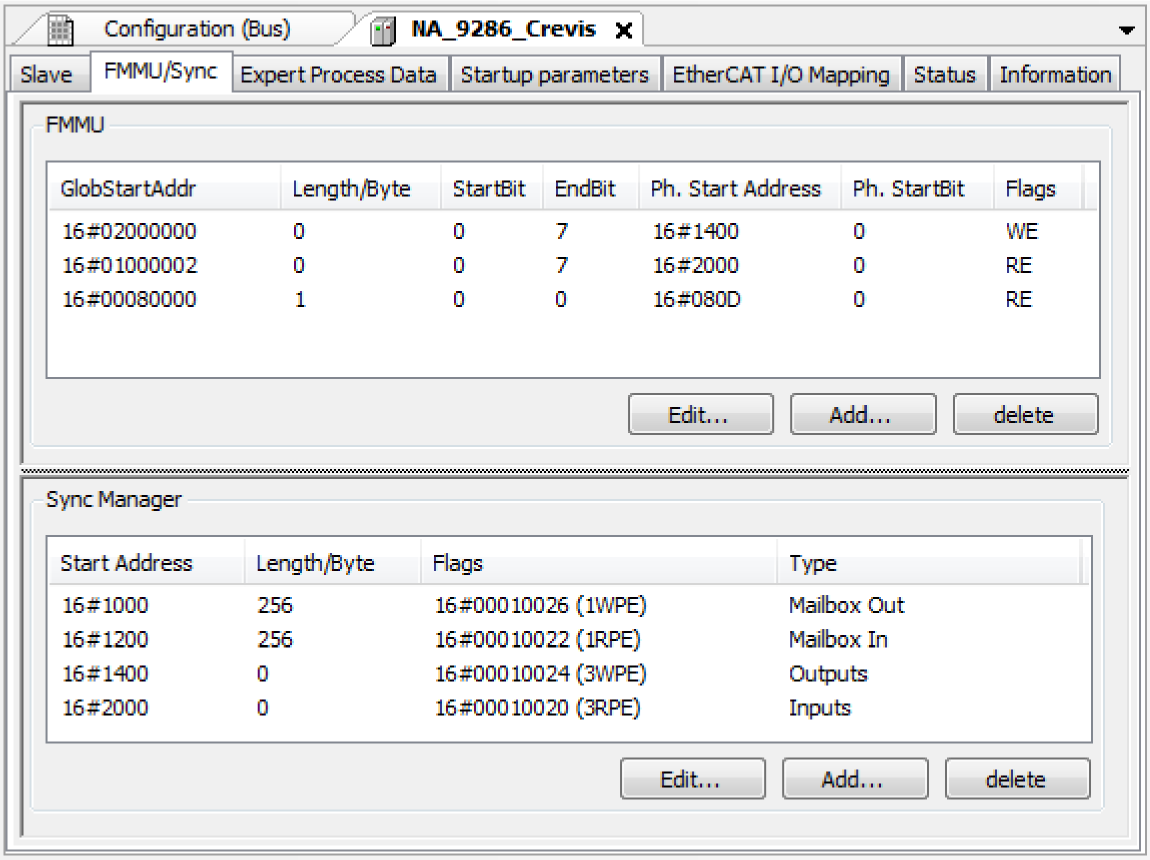


Figure 4-50. FMMU/Sync Dialog

FMMU

This table shows the Fieldbus Memory Management Units (FMMUs) of the slave which are used for handling the process data. Each mapping of the logical address *GlobStartAddr* on a physical address *Ph. StartBit* is defined. Bitwise mapping is possible. New units can be added and existing ones can be edited by the Edit FMMU dialog, to be opened via the *Add…* and *Edit…* buttons.

Sync Manager

This table shows the Synchronization Manager(s) of the slave. For each available Sync Manager Type (Mailbox In, Mailbox Out, Inputs, Outputs) a physical Start Address, the Access type, the Buffer and the physical address, where the Interrupts have to be sent to, are defined. New Sync Managers can be added resp. the existing ones can be edited in the dialog Edit SyncMan, which is to be opened by buttons *Add…* and *Edit…*

Process Data and Expert Process Data

The Process Data tab of the EtherCAT Slave configurator editor shows the slave input and output process data, each defined by name, type and index by the device description file, as seen in Figure 4-51.

The selected input (to be read) and output (to be written) of the device are available in the I/O Mapping dialog as PLC inputs and outputs to which project variables might be mapped.

In order to modify the current selection, first you must click on the checkbox before the currently selected data in order to cancel the selection. After that you can set another one.

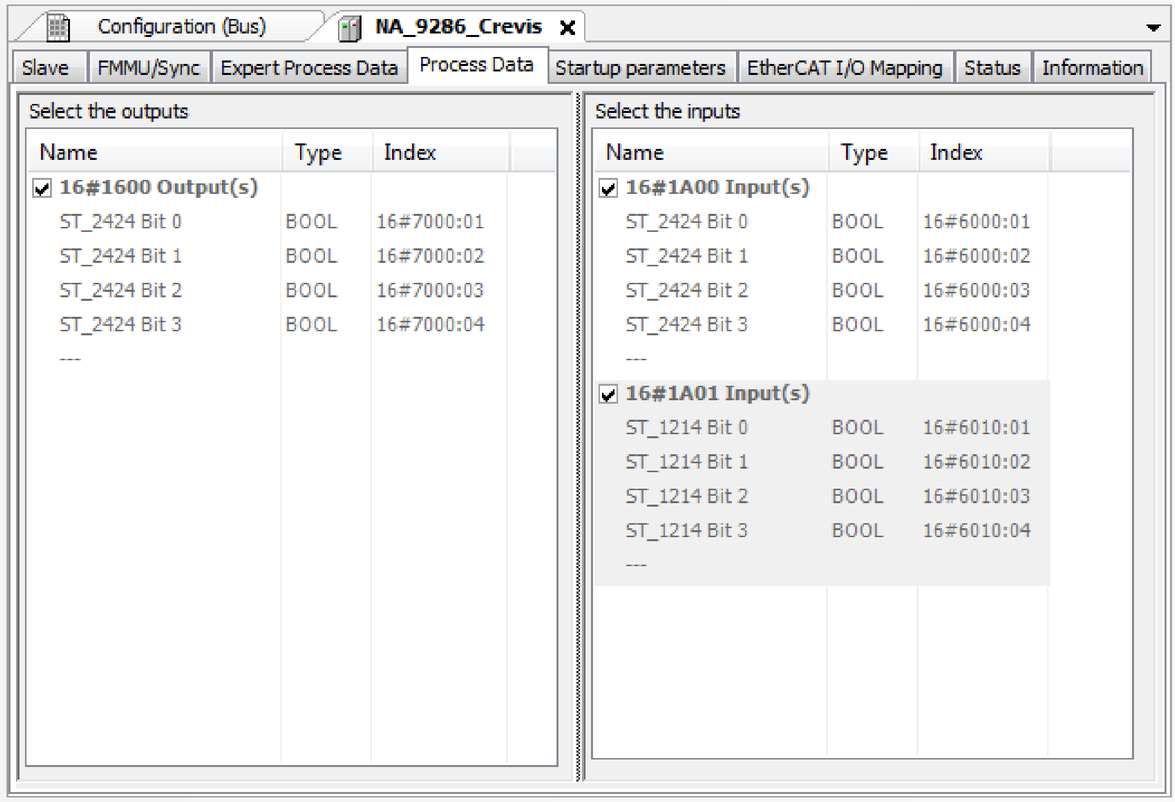


Figure 4-51. Process Data Dialog

The *Expert Process Data* dialog will only be available in the EtherCAT Slave configuration editor if the *Enable Expert Settings* option is activated. It provides another, more detailed, vision of the process data, adding to what is presented in the *Process Data* tab. Furthermore, the download of the PDO Assignment and the PDO Configuration can be activated in this dialog.

ATTENTION: If the Slave doesn't accept the PDO Configuration, it will stay in Pre-Operational state and none real time data exchange will be possible

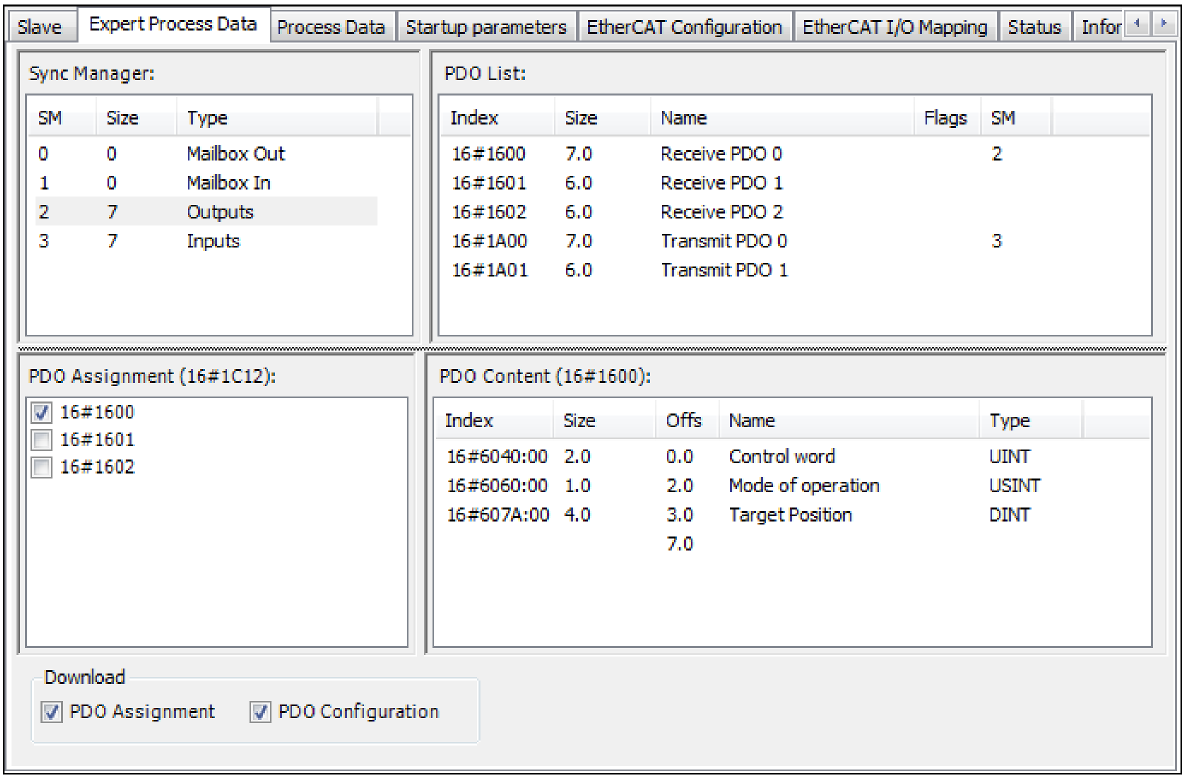


Figure 4-52. Expert Process Data Dialog

This dialog id divided in four sections and two options:

- *Sync Manager*: List of Sync Manager with data size and type of PDOs
- *PDO Assignment*: List of PDOs assigned to the selected Sync Manager. The checkbox activates the PDO and IO channels are created. It is similar to the simple PDO configuration windows. Here only PDOs can be enabled or disabled.
- *PDO List*: List of all PDOs defined in the device description file. Single PDOs can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Content*: Displays the content of the PDO selected in the section above. Entries can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Assignment*: If activated a CoE write command will be added to index 0x1CXX to write the PDO configuration 0x16xx or 0x1a00.
- *PDO Configuration*: If activated several CoE write commands will be added to write the PDO mapping to the slave.

ATTENTION: If a Slave doesn’t support the PDO configuration, a download may result in a Slave error. This function should only be used by experts.

Editing the PDO List

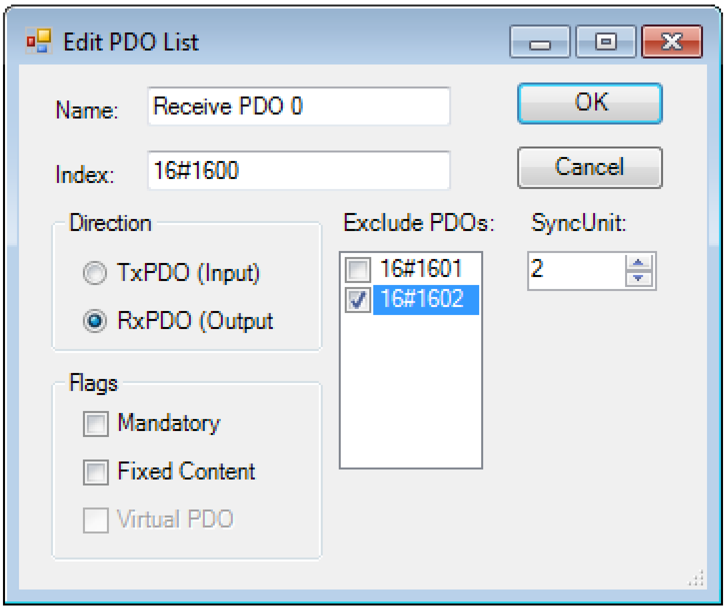


Figure 4-53. Edit PDO List Dialog

This dialog is opened through the context menu from the PDO List area, presented in Figure 4-52. Below are some explanations on the configuration options presented in this dialog.

- *Name*: Name of the PDO entry.
- *Index*: Index of the PDO in being edited.
- *TxPDO (Input)*: If activated, the PDO will be transferred from the Master to the Slave.
- *RxPDO (Output)*: If activated, the PDO will be transferred from the Master to the Slave.
- Mandatory: The PDO is necessary and can’t be unchecked in the PDO Assignment area.
- *Fixed Content*: The PDO content is fixed and can’t be changed. It’s not possible to add entries in the PDO Content panel.
- *Virtual PDO*: Reserved for future use.
- *Exclude PDOs*: It’s possible to define a list of PDO that can, or can’t, be selected along with the PDO being edited in the *PDO Assignment* area, or in the *Process Data* tab. If a PDO is marked in this list, it can’t be selected, turning into gray in the PDO Assignment area when the PDO in edition is selected.
- *SyncUnit*: ID of the Sync Manager the PDO shall assigned to.

Definition of the PDO Content

This dialog is accessed through the context menu in the PDO Content area, and its content, beyond the possibility to access this windows, varies according to the EtherCAT Slave in use.

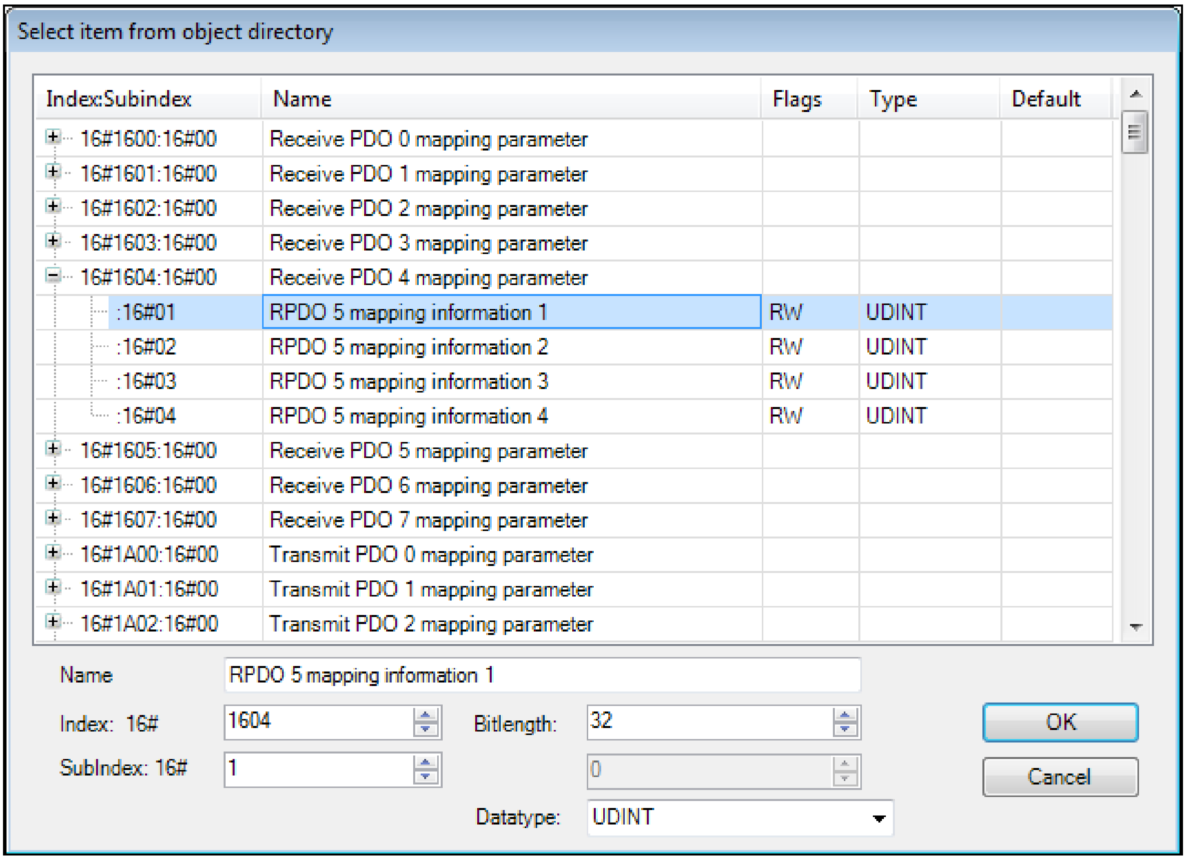


Figure 4-54. 'Select item from object directory' dialog

Startup Parameters

In the *Startup Parameters* tab, parameters for the device can be defined, which will be transferred by *SDOs* (Service Data Objects) or IDN at the system's startup. The options available in this tab, as well as the access possibilities, vary according to the EtherCAT Slave used and they are present in the *Device Description File*.

Online

The *Online* dialog is only available in the EtherCAT Slave configuration editor, if the option *Enable Expert Config* of the Slave is active and the *Application* is connected to the device. It provides a view with status information of the Slave and functions to transfer files to the Slavers over EtherCAT (FoE).

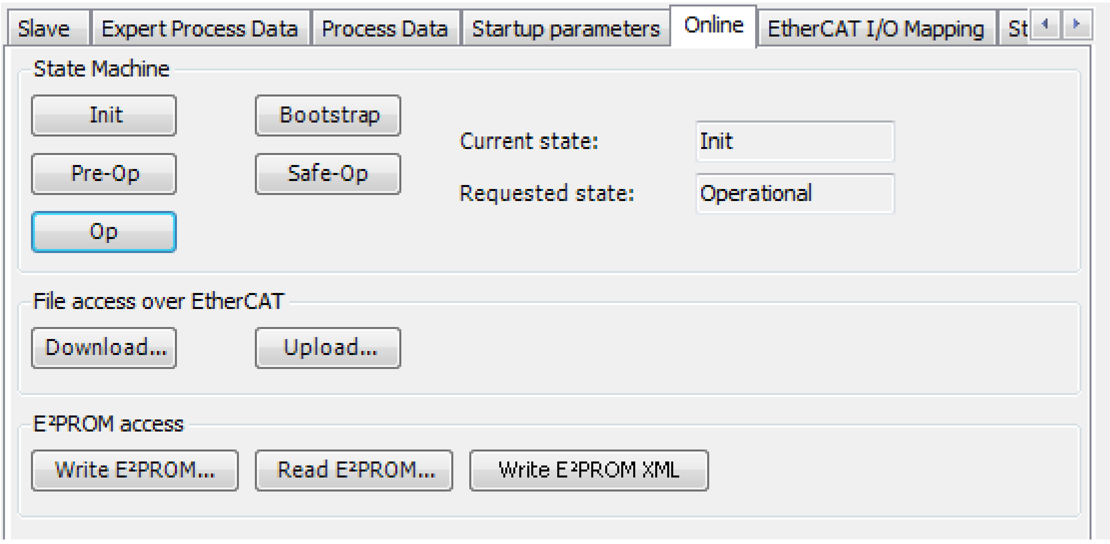


Figure 4-55. Online Dialog

This tab is divided in the following functionality group.

- **State Machine:** The buttons *Init*, *Pre-Op* (Pre-Operational), *Op* (Operational) and *Safe-Op* (Safe-Operational) can be used for debugging purposes. They make the slave transition to the respective state.
- **File access over EtherCAT:** If you want to transfer firmware files to or from the Slave, you have to click on the Bootstrap button to switch the slave in 'Bootstrap Mode'. The Download and Upload of firmware files can be done with the corresponding buttons. A dialog to save or open the firmware file will open. In this dialog, a string and password are required to execute the file transfer. This information is provided by the slave device and documented in the datasheet of the slave.
- **E2PROM access:** The slave configuration can be read from, or written to, the E^2PROM. Here, as well as for the firmware transfer, a dialog to open or save files will open. The command 'Write E^2PROM XML' can be used to write the Slave configuration directly from the XML file to the device. This command is only enabled if there's configuration data in the XML file (section <ConfigData>).

I/O Mapping

This tab of the EtherCAT Slave configuration editor offers the possibility to assign the project variables to the EtherCAT inputs or outputs. This way, the EtherCAT Slave variables can be controlled by the *User Application*.

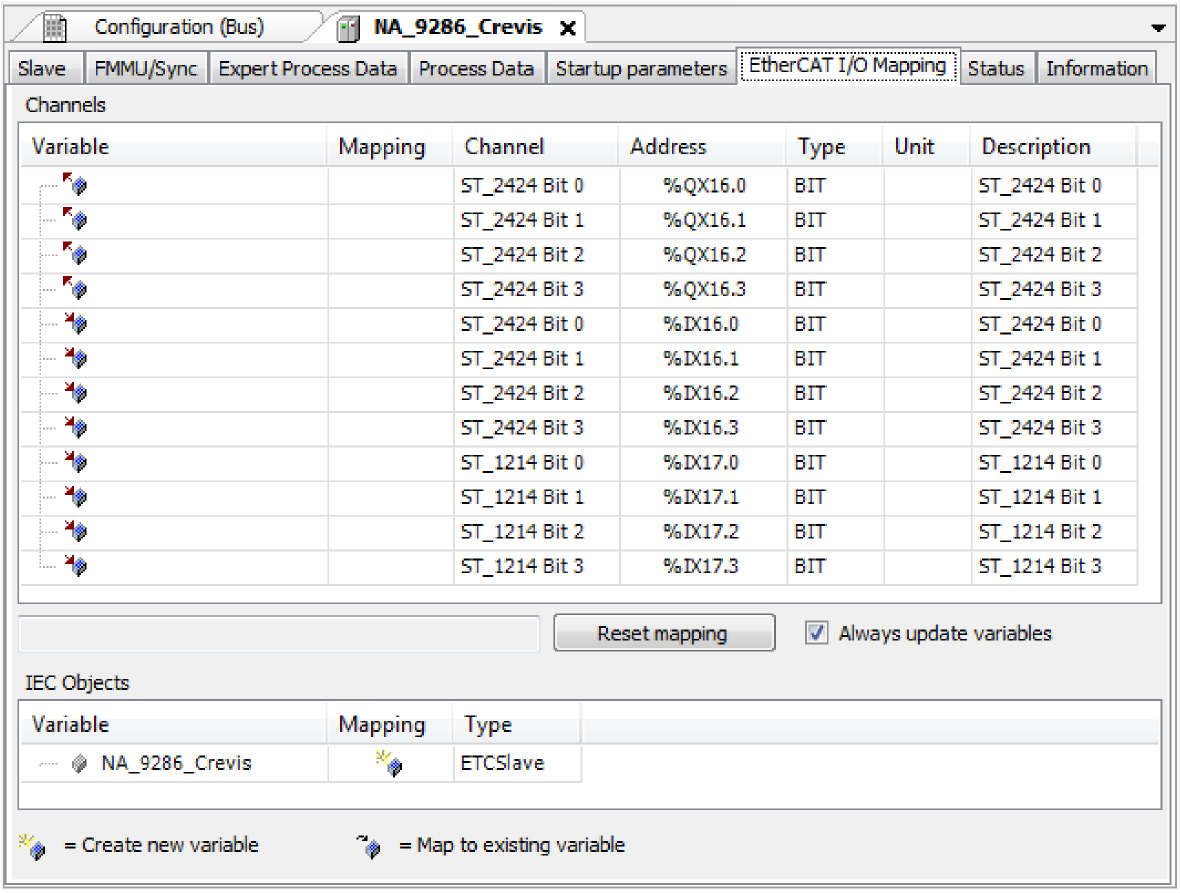


Figure 4-56. I/O Mapping Dialog

Status and Information tabs

The *Status* tab of the EtherCAT Slave provides status information (e.g. ‘Running’, ‘Stopped’) and device-specific diagnostic messages, also on the used card and the internal bus system.

The *Information* dialog, presented in the EtherCAT Slave configuration editor, show, if available, the following general information about the module: Name, Vendor, Type, Version Number, Categories, Order Number, Description, Image.

EtherNet/IP

The EtherNet/IP is a master-slave architecture protocol, whitch consist of an EtherNet/IP Scanner (the master) and one or more EtherNet/IP Adapter (the slave). The EtherNet/IP editor provides dialogs for setting parameters and for mapping inputs/outputs to variables.

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 the transport of other informations related to the system being controlled, such configuration parameters and diagnostics. The first one is done through implicit messages, while the second one is done through explicit messages.

This protocol is available to the following Nexto Series CPUs: NX3005, NX3010, NX3020 and NX3030. 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 a Ethernet expansion module.

Each device standing one or two EtherNet/IP Scanners support up to 32 Adapters under it/them. Being 32 the adapters number’s maximum limit. If there are two Scanners, both of them summed can’t have more then this limit. And each one of those Adapters can’t have more than 50 Inputs/Outputs.

An EtherNet/IP Adapter supports up to 64 modules of input or output. These modules can by of type BYTE, WORD, DWORD or REAL. And the MainTask's interval of a device running an Adapter must be lesser or equal the RPI.

ATTENTION
EtherNet/IP can't be used together with ethernet NIC-Teaming nor with Half-Cluster's redundancy.

ATTENTION
EtherNet/IP requires a cyclic MainTask, which means it can't be used with any configuration that sets the MainTask as freewheeling. As example, a Basic Profile project doesn't support EtherNet/IP.

ATTENTION
To avoid communication issues, EtherNet/IP Scanner can only have Adapters configured within the same subnetwork

EtherNet/IP Interface

To add an EtherNet/IP Scanner or Adapter it's needed to add an EtherNet/IP Interface under the NET interface. This can be done through the command "Add Device". Under this EtherNet/IP Interface it's possible to add a Scanner or an Adapter

EtherNet/IP Scanner Configuration

The Scanner requires at least one Adapter with which it will exchange data. New Adapters can be installed on MasterTool with the EDS and DCF Files. The configuration options may difer depending on the device description file of the added Adapter.

General

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 type Generic, then the fields "Device Type", "Vendor Code", "Product Code", "Major Revision" and "Minor Revision" must be fulfilled with the correct information and the boxes checked as long as needed. The verification can be switched from "Compatibility Check" to "Strict Identity Check".

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.

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)	Requested Packet Interval: exchange interval of the input and output data.	10 ms	Multiple of MainTask interval
O -> T Size (Bytes)	Size of the producer data from the Scanner to the Adapter (Originator --> Target).	Depends on adapter's EDS	0 - 400
T -> O Size (Bytes)	Size of the consumer data from the Adapter to the Scanner (T --> O).	Depends on adapter's EDS	0 - 400
Config #1 Size (Bytes)	Size of configuration data 1.	Depends on adapter's EDS	-
Config #2 Size (Bytes)	Size of configuration data 2.	Depends on adapter's EDS	-
Connection Path	Address of the - configuration objects - input objects - output objects.	Depends on adapter's EDS	-

Table 4-75. EtherNet/IP Connection parameters

To add new connections there is the button "Add Connection…" which will open the "New connection" window. On this window it's possible to configure a new type of connection from the ones predefined on Adapter's EDS or a generic conection from zero.

Assemblies

The upper area of the Assemblies tab displays a list of all configured connections. When a connection is selected, the associated assemblies in the lower area of the tab are displayed.

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. The order in the list determines the order in the I/O mapping.
Move Down	
Name	These values can be changed by double-clicking into the text field.
Help String	
Bit Length	This value must not be edited.

Table 4-76. EtherNet/IP Assemblies tab

Dialog box "Add Input/Output":

Configuration	Description
Name	Name of the input/output to be inserted,
Datatype	Type of the input/output to be inserted. This type also define its bitlength

Table 4-77. EtherNet/IP "Add Input/Output" window

EtherNet/IP I/O Mapping

I/O Mapping tab shows the name of the automatically generated instance of the Adapter below IEC Objects in the Variable column. In this way, the instance can be accessed by the application. Here the project variables are mapped to adapter's inputs and outputs.

The "Always update variables" option must be keeped as default in "Enable 1".

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 (INPUT BYTE, OUTPUT BYTE, etc)

Module Types

There are 8 diferent modules which can be added under the adapter. Four outputs and four inputs. They are of type BYTE, WORD, DWORD and REAL. These types can be chosen in the general tab of the module

EtherNet/IP Module I/O Mapping

It shows the name of the automatically generated instance of the module below IEC Objects in the Variable column. In this way, the instance can be accessed by the application

The “Always update variables” option must be keeped as default in Enable 1.

Communication Performance

Communication Rate of a MODBUS Server Device

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 cycles (cycle time of the MODBUS task), whereas 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 a 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.

CPU's Local Interfaces

For a device MODBUS Ethernet Server, we can assert that the device is capable to answer an 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 4-78 must be viewed for information.

The Table 4-78 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 20 ms
1 Connection	160	47
2 Connections	300	95
3 Connections	395	142
4 Connections	470	190
5 Connections	515	237
6 Connections	570	285
7 Connections	605	332
8 Connections	640	380
9 Connections	665	427
10 Connections	690	475

Table 4-78. Communication Rate of an MODBUS Server at Local Interface

ATTENTION:
The communication performances mentioned in this chapter are just examples, using a CPU with only one device MODBUS TCP Server, with no logic to be executed inside the application that could delay the communication. Therefore, these performances must be taken as the maximum rates.

For cycle times equal or greater than 20 ms, the increase of the answer rate is linear, and may be calculated using an equation:

N =- C x (Z – (Z / (T x 1000)))

Z = 1 / T

Where *N* is the medium number of answers per second, *C* is the number of active connections and *T* is equal to the cycle time of the MODBUS task (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; **Z** = 1 / 0.05 = 20;

N = 1 x (20 – (20 / (0.05 x 1000))) = 1 x (20 – (20 / 50)) = 1 x (20 – 0.4) = 1 x 19.6

N = 19.6

That is, in this configuration the MODBUS Server answers, on average, 19 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.

Remote Interfaces

The performance of a device MODBUS Server in one remote Ethernet interface is similar to the performance in the CPU's local interfaces.

However, due to time of the communication between the CPU and the modules, the maximum performance is limited. For only one active connection, the number of answers is limited in the maximum of 18 answers per second. With more active connections, the number of answers will increase linearly, exactly like the local interfaces, however being limited at the maximum of 90 answers per second. So, for a remote Ethernet interface, we will have the following forms to calculate his performance:

For $T \leq 55\ ms$ is used:

N = C x (18,18 – (18,18 / (0,055 x 1000)))

And for $T \geq 55\ ms$ is used:

N = C x (Z – (Z / (T x 1000)))

Where *N* is the medium number of answers per second, *C* is the number of active connections and *T* is equal to the cycle time of the MODBUS task (in seconds).

The user must pay attention to the fact that the maximum performance of a device MODBUS Server in one remote Ethernet interface is 90 answers of requisitions per second.

Communication Rate of a Device with OPC Server

Communication performance with OPC 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 {attribute 'symbol' := 'readwrite'}, to make the data available to the OPC 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 4-79

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 4-79.

Total quantity of variables in the PLC's project	Variable update time at OPC Client		
	5% of CPU Busy	50% of CPU Busy	80% of CPU Busy
1,000	600 ms	800 ms	1400 ms
2,000	800 ms	900 ms	2800 ms
5,.000	1000 ms	2000 ms	6500 ms
10,000	2000 ms	4000 ms	13700 ms
15,.000	3200 ms	6400 ms	20000 ms
20,000	4000 ms	8100 ms	25000 ms

Table 4-79. Communication Rate of an OPC Server

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 applicative 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

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 Simple 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 using 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 (NX3010, NX3020 and NX3030), 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.

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 4-80 illustrates the average time that is added to the MainTask:

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

Table 4-80. 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.

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 Configuration - Memory Card chapter.

RTC Clock

Nexto Series 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 chapter). Figure 4-57 shows how to include the blocks in the project:

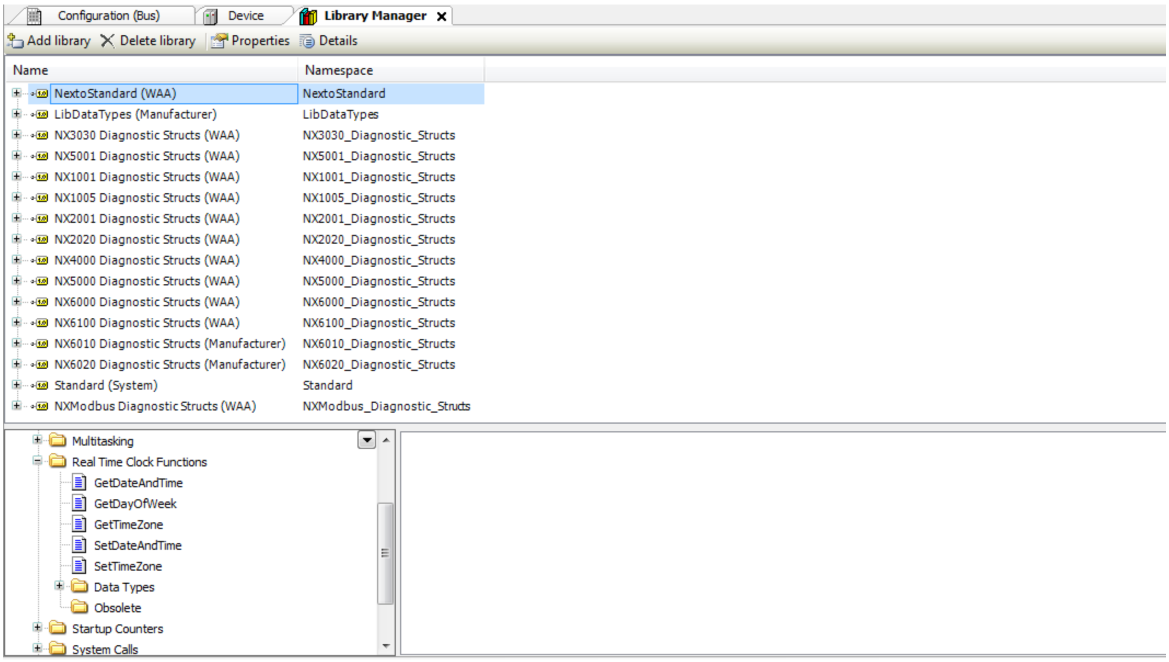


Figure 4-57. Clock Reading and Writing Blocks

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 e NextoGetTimeZone, NextoSetDateAndTime, NextoSetDateAndTimeMs e NextoSetTimeZone.

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 (GetDateAndTime, and SetTimeZone). The proceedings to configure these two blocks are described below:

ATTENTION: The function blocks for RTC (RTC (NextoGetDateAndTime, NextoGetDateAndTimeMs, NextoSetDateAndTime e NextoSetDateAndTimeMs) reading and writing cannot be used in the area of redundant data in redundant projects. The function blocks can be used only in non-redundant POUs, as the POU NonSkippedPrg. For more details on the functioning of POU NonSkippedPrg see NonSkippedPrg.

Function Blocks for RTC Reading

The clock reading can be made through the following functions:

GetDateAndTime:

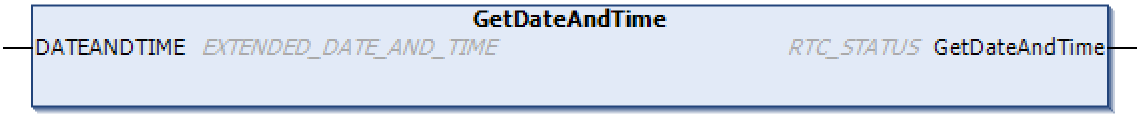


Figure 4-58. Date and Hour Reading

Input and Output 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 4-90

Table 4-81. Input and Output Parameters of GetDateAndTime

Input Parameters	Type	Description
GETDATEANDTIME	RTC_STATUS	Returns the function error state, see Table 4-92

Table 4-82. Output Parameters of GetDateAndTime

Utilization example in ST language:

```
PROGRAM MainPrg
VAR
Result : RTC_STATUS;
DATEANDTIME : EXTENDED_DATE_AND_TIME;
xEnable : BOOL;
END_VAR
--------------------------------------------------------------------------
IF xEnable = TRUE THEN
Result := GetDateAndTime(DATEANDTIME);
xEnable := FALSE;
END_IF
```

GetTimeZone

The following function reads the Time Zone configuration, this function is directly related with time in Time Zone at SNTP synchronism service.

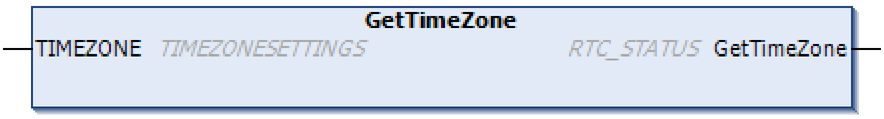


Figure 4-59. Configuration Reading of Time Zone

Input and Output Parameters	Type	Description
TimeZone	TIMEZONESETTINGS	This variable present the reading of Time Zone configuration.

Table 4-83. Input and Output Parameters of GetTimeZone

Output Parameters	Type	Description
GetTimeZone	RTC_STATUS	Returns the function error state, see Table 4-92.

Table 4-84. Output Parameters of GetTimeZone

Utilization example in ST language:

```
PROGRAM MainPrg
VAR
GetTimeZone_Status    : RTC_STATUS;
TimeZone              : TIMEZONESETTINGS;
xEnable : BOOL;
END_VAR
--------------------------------------------------------------------------
IF xEnable = TRUE THEN
GetTimeZone_Status := GetTimeZone(TimeZone);
xEnable := FALSE;
END_IF
```

GetDayOfWeek

GetDayOfWeek function is used to read the day of the week.

Figure 4-60. Day of Week Reading

Output Parameters	Type	Description
GetDayOfWeek	DAYS_OF_WEEK	Returns the day of the week, See Section EXTENDED_DATE_AND_TIME

Table 4-85. 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 MainPrg
VAR
   DayOfWeek   : DAYS_OF_WEEK;
END_VAR
--------------------------------------------------------------------------
DayOfWeek := GetDayOfWeek();
```

Function Blocks and Functions of RTC Writing and Configuration

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

SetDateAndTime

SetDateAndTime function is used to write the settings on the clock.

Figure 4-61. SetDateAndTime

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 EXTENDED_DATE_AND_TIME

Table 4-86. 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 Reading/Writing.
STATUS	RTC_STATUS	Returns the error occurred during the reading/configuration. See section RTC Data Structures - RTC_STATUS.

Table 4-87. 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 MainPrg
VAR
   SetDateAndTime : SetDateAndTime;
   xRequest : BOOL;
   DateAndTime : EXTENDED_DATE_AND_TIME;
   xDone : BOOL;
   xExec : BOOL;
   xError : BOOL;
   Status : RTC_STATUS;
   xWrite : BOOL;
END_VAR
--------------------------------------------------------------------------
IF (xWrite = TRUE) THEN
   SetDateAndTime(
   request := xRequest,
   done => xDone,
   exec => xExec,
   error => xError,
   status => status,
   DateAndTime := DateAndTime);
   xWrite := FALSE;
END_IF
```

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 pm, it is converted to 1 day and 6 hours, and so on.

SetTimeZone

The following function block makes the writing of the time zone settings:

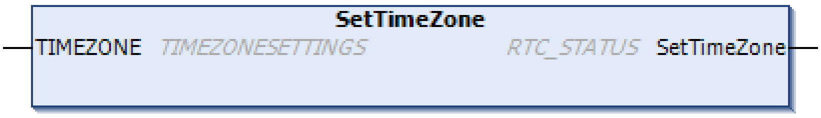


Figure 4-62. Writing of the Time zone Settings

Input parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	Structure with time zone to be configured. See section RTC Data Structures - TIMEZONESETTINGS.

Table 4-88. SetTimeZone Input Parameters

Output parameters	Type	Description
SetTimeZone	RTC_STATUS	Returns the error occurred during the reading/setting. See section RTC Data Structures - RTC_STATUS.

Table 4-89. 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 MainPrg
VAR
    Status : RTC_STATUS;
    TimeZone : TIMEZONESETTINGS;
    xWrite : BOOL;
END_VAR
------------------------------------
//FB SetTimeZone
IF (xWrite = TRUE) THEN
    Status := SetTimeZone(TimeZone);
    xWrite := FALSE;
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.

RTC Data Structures

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

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 4-90:

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	wMiliseconds	Stores the milliseconds of the set date.

Table 4-90. EXTENDED_DATE_AND_TIME

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 4-91. DAYS_OF_WEEK Structure

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 4-92:

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 4-92. RTC_STATUS

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 4-93:

Structure	Type	Variable	Description
TIMEZONE SETTINGS	INT	iHour	Set time zone hour
	INT	iMinutes	Set time zone minute

Table 4-93. 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.

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 Technical Description - Specific Features chapter).

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 4-63.

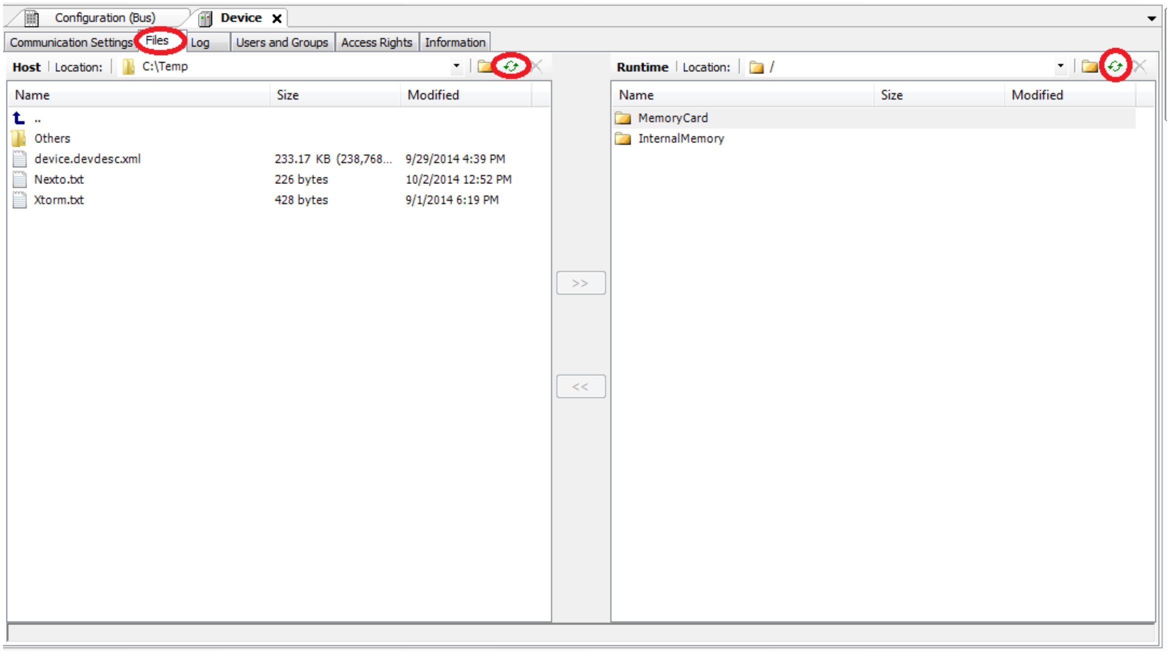


Figure 4-63. User Files Access

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 4-64. 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.

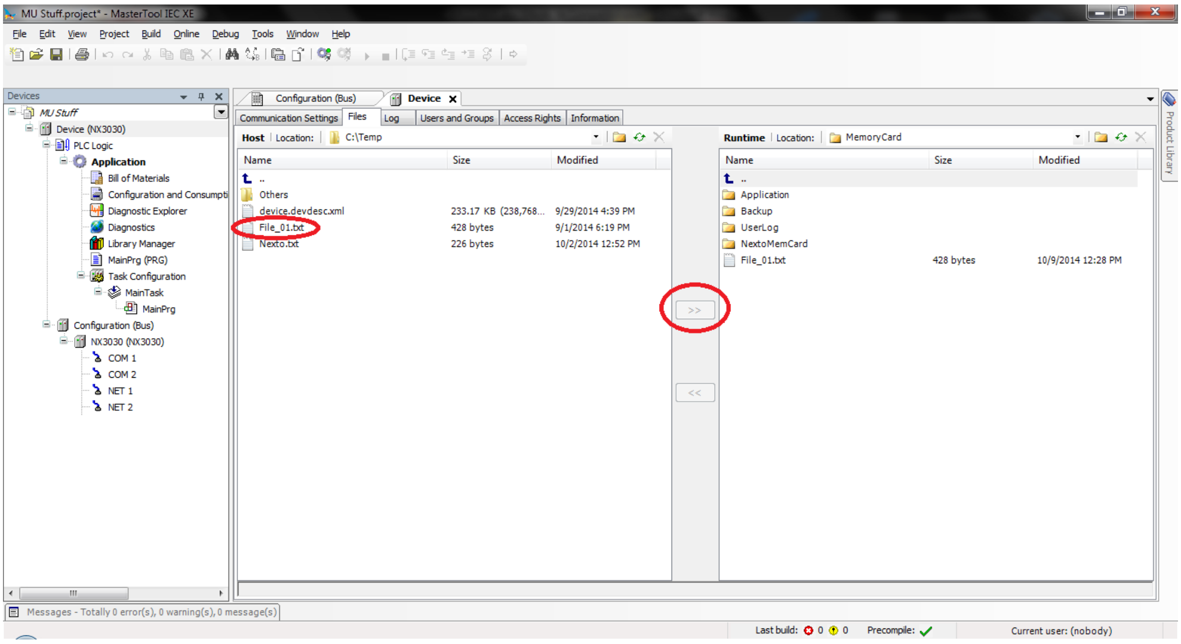


Figure 4-64. Files Transference

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

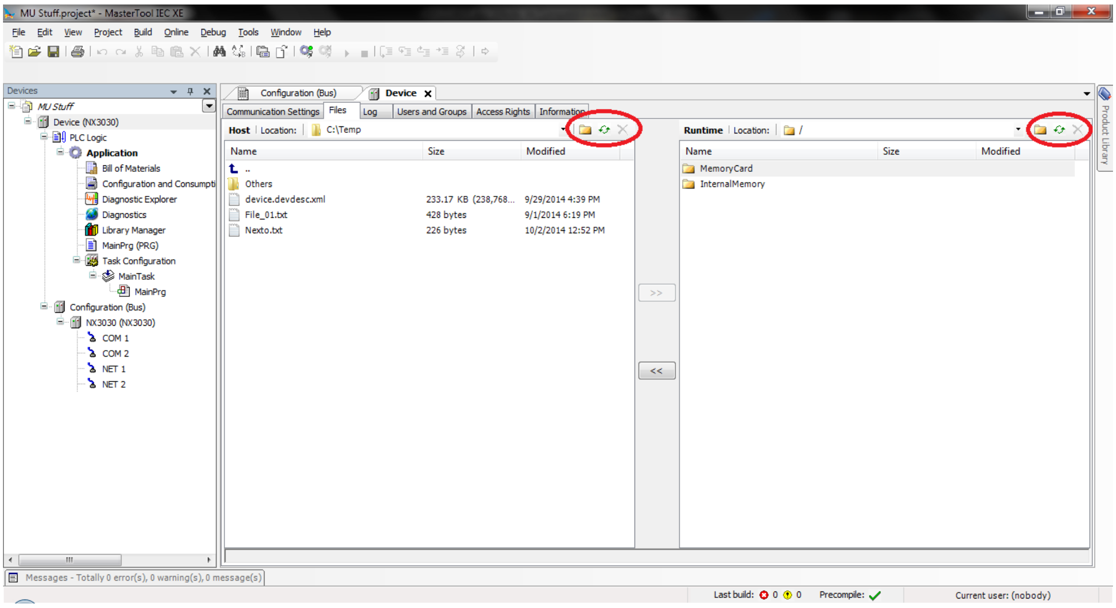


Figure 4-65. Utilization Options

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

Memory Card

Among other memories, the NX3010, NX3020 and NX3030 Nexto Series CPUs allow the user the utilization of a memory card. It is defined according the features described in Memory Card Interface chapter 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 slower memory card 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.

During the project configuration, in the MasterTool IEC XE software, the user enables the CPU project copy option for the memory card or the project copy from the card to the CPU. On this same screen the user can configure the passwords, which control the information use. For further information regarding the table, see Project Parameters chapter.

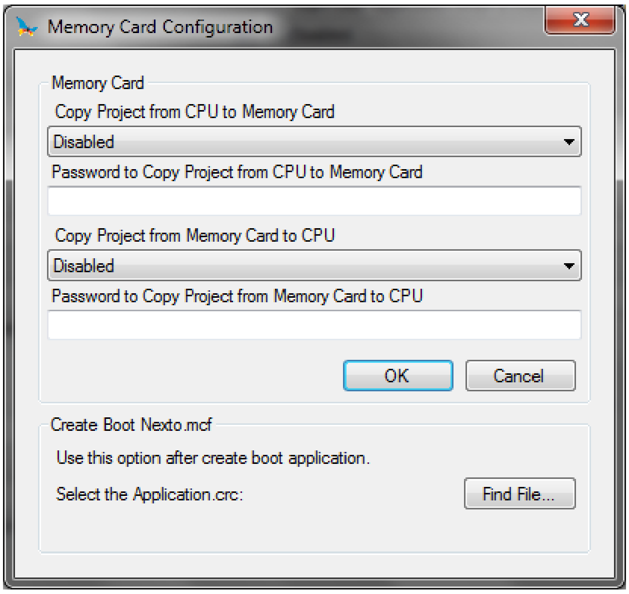


Figure 4-66. Memory Card Configuration

When a password is configured for the memory card in MasterTool, it is necessary to perform the following steps so that when the project is sent, the encrypted file which is created by MasterTool has the password included in its content and it is also sent.

First set up (s) password (s) and click on "OK". The password in this case should contain only numeric characters. At this time passwords were recorded and the next step is run in the Communication menu the command "Create boot Application". In order to perform this procedure it is not possible to be logged on the CPU. After running this command, two files are created: one with the "app" extension and another with "crc" extension. In order to finish the setup operation of the password(s) it is necessary to click again the "Memory Card" key which is in the configuration of the CPU General Parameters and then locate the file with the "crc" generated in the previous step, using the "Find File ..." key. Performed these steps, the MasterTool IEC XE will send all the files necessary to perform the operations of sending and receiving projects via memory card.

In case the card is mounted, the password will be recorded on it, otherwise the set password in MasterTool will be requested if the user try to transfer the project from the CPU to the card.

To execute the CPU sending to the memory card or vice-versa, the user must enable the function in the MasterTool IEC XE software, type the password and access the “Memory Card” menu in the CPU using the diagnostics key, and then select the desired transference option. After this, the password will be requested if the user configured so during the application. So, with a short touch on the diagnostic key the digits are incremented and with a long touch are confirmed. At the sixth confirmed digit, the CPU will consider the password and begin the process. When both the passwords from the application which is in the CPU and from the application which is in the memory card are equal, it is not requested the passwords insertion in the CPU menu in order to make the application transferences. For further information regarding the diagnostic key utilization, see One Touch Diag chapter.

To remove the memory card, press the MS button with a long touch and wait until the card icon disappears from the status screen on the graphic display.

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.

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 “Memory Card”. “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.

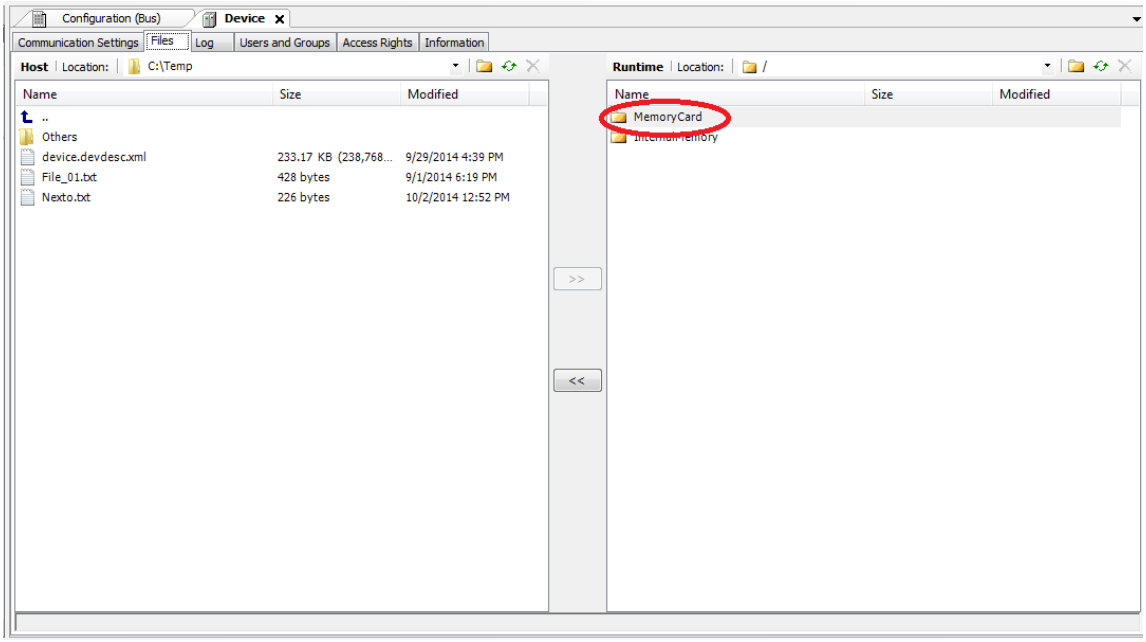


Figure 4-67. Root Directory with Memory Card Inserted

The file transference happens in a similar way to the user memory utilization (User Files Memory). Access the folder “MemoryCard” and send the files as depicted on Figure 4-68.

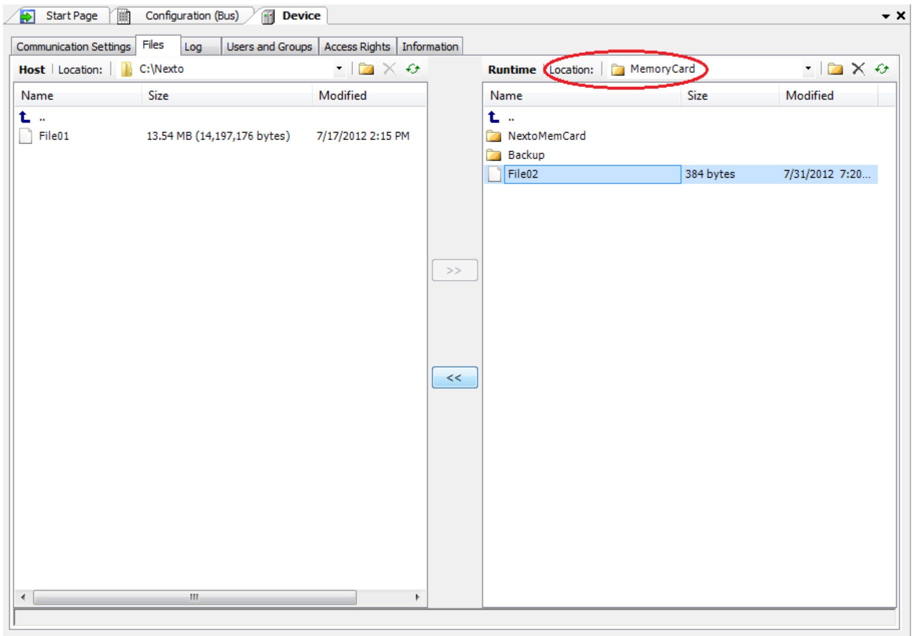


Figure 4-68. Files Stored in the Memory Card

Inside the memory card directory, additionally to the files which are stored into the card, it will be also the “NextoMemCard” and “Backup” folders. In these folders both the application and the current project are saved in case the user chooses to transfer them or to make a backup of them through the CPU menu.

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.

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 chapter to verify the functioning and the difference between the diagnostics button touch types.

Table 4-94 shows the menu levels and each screen type available in the CPU, 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 HOUR	-	Informative
	RETURN	-	Return level
LANGUAGES	ENGLISH	>ENGLISH	Configurable
	PORTUGUESE	>PORTUGUES	Configurable
	SPANISH	>ESPANOL	Configurable
	RETURN	-	Return level
NETWORK	END. IP NET 1	-	Informative
	MASK NET 1		Informative
	END. IP NET 2		Informative
	MASK NET 2		Informative
	RETURN		Return level
REDUNDANCY	IDENT. CP	-	Informative
	ESTADO REM.		Informative
	SINCR. PROJ.		Informative
	RETURN		Return level
SOFTWARE	FIRMWARE	-	Informative
	BOOTLOADER		Informative
	PROC. AUX.		Informative
	RETURN		Return level
MEM. CARD	CARD > CPU	CPU PASSWORD	Configurable
	CPU > CARD	CM PASSWORD	Configurable
	FORMAT	CONFIRM ?	Configurable
	RETURN	-	Return level
RETURN	-	-	Return level

Table 4-94. 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.

Network: the NET 2 interface items are only available in the NX3020 and NX3030 CPUs.

Redundancy: the "Redundancy" menu will only be available in case the NX3030 CPU is identified as Redundant.

As shown on Table 4-94, 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)
- Changing the menu language on the CPU:
  - PORTUGUESE – Changes the language to Portuguese
  - ENGLISH – Changes the language to English
  - SPANISH – 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 – Sub net mask (Ex.: 255.255.255.0)
  - NET 2 IP ADDR - address (Ex.: 192.168.0.2)
  - NET 2 MASK – Sub net mask (Ex.: 255.255.255.0)
- Information about the software versions:
  - FIRMWARE – CPU software version (Ex.: 1.0.0.0)
  - BOOTLOADER – CPU boot loader version (Ex.: 1.0.0.0)
  - AUX. PROC. – CPU auxiliary processor version (Ex.: 1.0.0.0)
- Access to the PLC redundancy information:
  - PLC ID – Informs the PLC identification in the redundancy. Possible information:
    - CPA
    - CPB
  - REMOTE STATE – Informs the state of the remote redundant PLC. Possible states:
    - ACTIVE
    - STANDBY
    - INACTIVE
    - NOT CONFIG.
    - STARTING
    - UNAVAILABLE
  - PROJ. SYNC. – Informs if the synchronization of the projects is enabled
  - CONNECTED
  - NOT CONNEC.
  - DISABLED
  - START SYNC.
  - SYNCHRONIZED
- 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

Figure 4-69 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, 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.

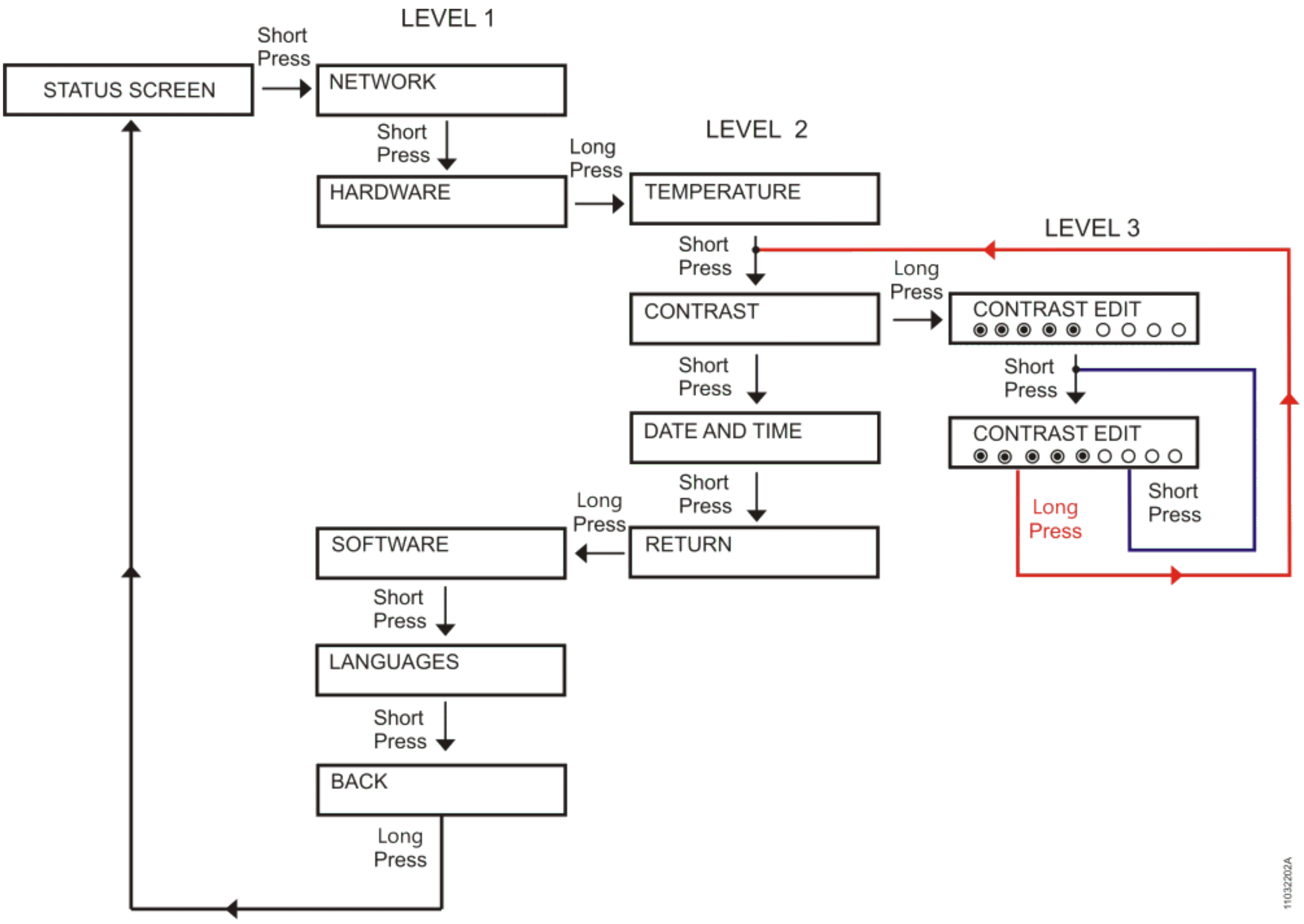


Figure 4-69. Contrast Adjust

Besides the possibility of the Nexto CPUs menu to be closed through a long touch on the screen diagnostic button RETURN 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.

Function Blocks and Functions

Special Function Blocks for Serial Interfaces

The special function blocks for serial interfaces make possible the local access (COM1 AND COM2) 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 Nexto Serial 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 – MP399608, chapter Library).

The special function blocks for serial interfaces can take several cycles (consecutives 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.
	RS232_RTS_CTS	Controls the RS-232C port of the CPU Nexto. 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 CPU Nexto. The user is responsible to control all the signals (RTS, DTR, CTS, DSR, DCD).
SERIAL_MODE	NORMAL_MODE	Serial Communication Normal Operation mode.
	EXTENDED_MODE	Serial Communication Extended Operation mode in which are provided information about the received data frame.
SERIAL_PARAMETERS	Defines all configuration parameters of the serial port:	

Data type	Options	Description
	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.
	RX_QUEUE_EXTENDED	When true, select the RX line and the block extended mode. SERIAL_RX_EXTENDED must be used to receive data. On the other hand, for the RX line with normal format, the SERIAL_RX block must be used to receive data.
	MODE	Defined at SERIAL_MODE
	ENABLE_RX_ON_TX	When true, all the received byte during the transmission will be discharged instead going to the RX line. Used to disable the full-duplex operation in the RS-422 interface.
	ENABLE_DCD_EVENT	When true, generates an external event when the DCD is modified.
	ENABLE_CTS_EVENT	When true, generates an external event when the CTS is modified.
SERIAL_PARITY	PARITY_NONE	List all parity possibilities.
	PARITY_ODD	
	PARITY_EVEN	
	PARITY_MARK	
	PARITY_SPACE	
SERIAL_PORT	COM 1	List all available serial ports (COM 10, COM 11, COM 12, COM 13, COM 14, COM 15, COM 16, COM 17, COM 18, and COM 19 – expansion modules).
	COM 2	
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

Data type	Options	Description
		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.
SERIAL_STATUS	List of critic error codes that can be returned by the serial function block. Each block returns specific errors, which will be described below:	
	NO_ERROR	No errors.
	ILLEGAL_*	Return the parameters with invalid values or out of range: - SERIAL_PORT - BAUDRATE - DATA_BITS - PARITY - STOP_BITS - HANDSHAKE - UART_RX_THRESHOLD - 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_TIME-OUT_ON	Time-out while waiting for the CTS enabling, in the RS-232 RTS/CTS handshake, in the SERIAL_TX block.
	CTS_TIME-OUT_OFF	Time-out while waiting for the CTS disabling, in the RS-232 RTS/CTS handshake, in the SERIAL_TX block.
	TX_TIME-OUT_ERROR	Time-out while waiting for the transmission ending in the SERIAL_TX block.
	RX_TIME-OUT_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 R232_MANUAL.
	FB_GET_CTRL_NOT_ALLOWED	The GET_CTRL block can't be used in case the handshake is different from R232_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 isn't available for the RX queue, normal mode.
	DCD_INTERRUPT_NOT_ALLOWED	The interruption by the DCD signal can't be enabled in case the serial port doesn't have the respective pin.
	CTS_INTERRUPT_NOT_ALLOWED	The interruption by the CTS signal can't be enabled in case the handshake is different from

Data type	Options	Description
		R232_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.
	NOT_CONFIGURED	The function block can't be used before the serial port configuration.
SERIAL_STOPBITS	STOPBITS_1	List all stop-bits possibilities.
	STOPBITS_2	
	STOPBITS_1_5	

Table 4-95. Serial Function Blocks Data types

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 4-70. 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 4-96 Serial_CFG Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - ILLEGAL_SERIAL_MODE, - ILLEGAL_BAUDRATE - ILLEGAL_DATA_BITS - ILLEGAL_PARITY - ILLEGAL_STOP_BITS - ILLEGAL_HANDSHAKE - ILLEGAL_UART_RX_THRESHOLD

		- PORT_BUSY - HW_ERROR_UART HW_ERROR_REMOTE - DCD_INTERRUPT_NOT_ALLOWED - CTS_INTERRUPT_NOT_ALLOWED - DSR_INTERRUPT_NOT_ALLOWED

Table 4-97. Output Parameters Serial_CFG

Utilization example in ST language, after the library Nexto Serial is inserted in the project:

```
PROGRAM MainPrg
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's necessary to treat the error
```

SERIAL_GET_CFG

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

Figure 4-71. 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 4-98. 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.
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 4-99. SERIAL_GET_CFG Output Parameters

Utilization example in ST language, after the library is inserted in the project:

```
PROGRAM MainPrg
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's necessary to treat the error.
Parameters := GetConfig.PARAMETERS;    //Receive the parameters of desired
serial port.
```

SERIAL_GET_CTRL

This function block is used to read the CTS, DSR and DCD control signals, in case they are available in the serial port. A false value will be returned when there are not control signals.

Figure 4-72. 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 4-100. 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 4-101. SERIAL_GET_CTRL Output Parameters

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

```
PROGRAM MainPrg
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's necessary to treat the error.
Get_Control.CTS_VALUE;
Get_Control.DSR_VALUE;
Get_Control.DCD_VALUE;
```

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 4-73. 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 4-102. 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 4-103. 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 MainPrg
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’s necessary to treat the
error
Status_RX := Get_Status.RXQ_STATUS;    // If it’s necessary to treat the
error of the RX queue
```

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.

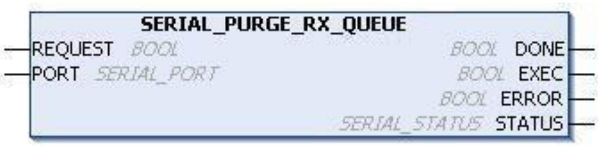


Figure 4-74. Block Used to Clean the RX Queue

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 4-104. 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 4-105. SERIAL_PURGE_RX_QUEUE Output Parameters

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

```
PROGRAM MainPrg
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’s necessary to treat the
error.
```

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 4-75. 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).
RX_TIME-OUT	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 4-106. 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_TIME-OUT_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 4-107. SERIAL_RX Output Parameters

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

```
PROGRAM MainPrg
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’s necessary to treat the error.
Receive.RX_RECEIVED;
Receive.RX_REMAINING;
```

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.

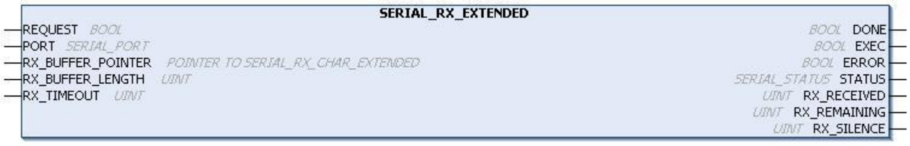


Figure 4-76. Block Used for Reception Buffer Reading

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.
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_TIME-OUT	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 4-108. 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_TIME-OUT_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

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 10µs. 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_REMANING = 0.

Table 4-109. SERIAL_RX_EXTENDED Output Parameters

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

```
PROGRAM MainPrg
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's necessary to treat the
error
Receive_Ex.RX_RECEIVED;
Receive_Ex.RX_REMAINING;
Receive_Ex.RX_SILENCE;
```

SERIAL_SET_CTRL

This block is used to write on the control signals (RTS and DTR), when they are available in the serial port. It can also set a busy condition for the transmission, through BREAK parameter and it can only be used if the modem signal is configured for RS232_MANUAL.

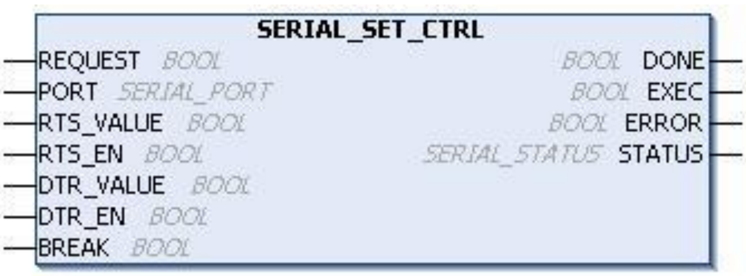


Figure 4-77. Block for Control Signals Writing

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 DTS_VALUE parameter writing.
BREAK	BOOL	In case it's true, enables logic 0 (busy) in the transmission line.

Table 4-110. 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 4-111. SERIAL_SET_CTRL Output Parameters

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

```
PROGRAM MainPrg
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's necessary to treat the
error.
```

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 4-78. 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).
TX_TIME-OUT	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	Specified 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 4-112. 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_TX_BUFF_LENGTH

		- CTS_TIME-OUT_ON - CTS_TIME-OUT_OFF - TX_TIME-OUT_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 4-113. SERIAL_RX Output Parameters

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

```
PROGRAM MainPrg
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's necessary to treat the error.
Transmit.TX_TRANSMITTED;
```

Inputs and Outputs Update

Functionality used to update inputs and outputs in the applicative. It is not necessary to wait until the cycle is finished.

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

ATTENTION:
At the startup of a Nexto CPU, 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.

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 task.
> 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 4-1

> 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).

Figure 4-79. Block for Input Updating

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

Table 4-114. REFRESH_INPUT Input Parameters

Possible ERRORCODE:

- NoError: Execution success.
- IOModuleAbsent: The module was configured but is absent.
- IOModuleNotConfigured: The module was not configured.
- ParameterMismatch: This error is returned in case the Always Update Variable option is not set or in case the REFRESH_INPUT function is called for a module that has only outputs.
- InputReadFail: Module internal critical failure (the function transmitted frame was not returned within the defined time-out).
- FrameTransmitError: Module internal critical failure (error during the frame transmission in the function).
- BusBusy: Module internal critical failure (the bus is not enabled for frame transmission in the function).

Utilization example in ST language:

```
PROGRAM MainPrg
VAR

    Info: ERRORCODE;
    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.
```

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 task 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 4-1.

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).

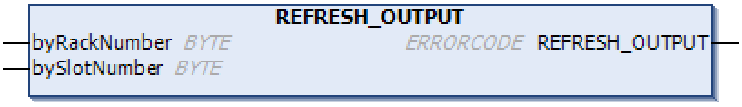


Figure 4-80. Block for Output Updating

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

Table 4-115. REFRESH_OUTPUT Input Parameters

Possible ERRORCODE:

- NoError: Execution success.
- IOModuleAbsent: The module was configured but is absent.
- IOModuleNotConfigured: The module was not configured.
- ParameterMismatch: This error is returned in case the Always Update Variable option is not set or in case the REFRESH_OUTPUT function is called for a module that has only inputs.
- OutputWriteFail: Module internal critical failure (the function transmitted frame was not returned within the defined time-out).
- FrameTransmitError: Module internal critical failure (error during the frame transmission in the function).
- BusBusy: Module internal critical failure (o the bus isn't enabled for frame transmission in the function).

Utilization example in ST language:

```
PROGRAM MainPrg
VAR
    Info: ERRORCODE;
    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.
```

PID Function Block

The PID function block is used to control a real process. The block is always available in the NextoPID library which must be added to the project (for the library insertion proceeding, see MasterTool IEC XE Programming Manual – MP399608, chapter Library).

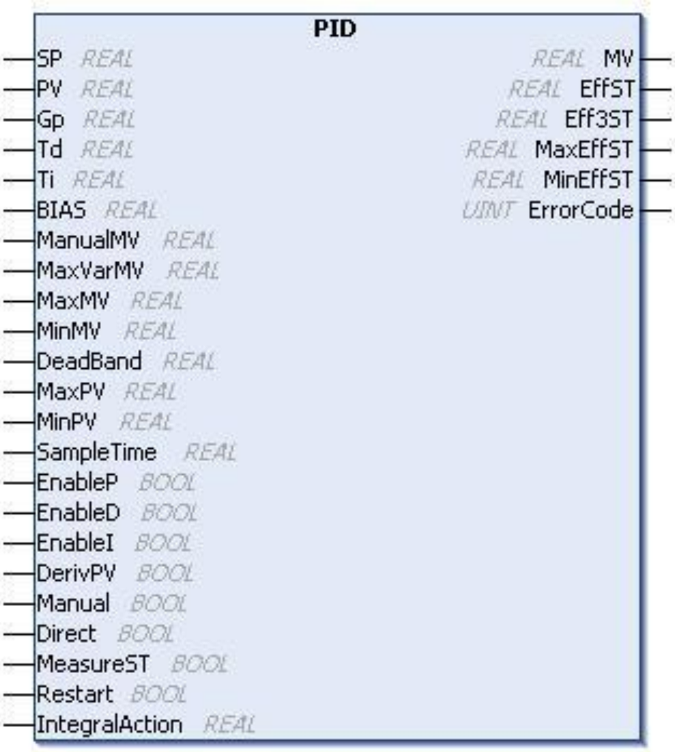


Figure 4-81. PID Block

Input parameters	Type	Description
SP	REAL	Set point. The unit and the interval must be the same in comparison with the PV as both variables can be compared.
PV	REAL	Process variable. The unit and the interval must be the same in comparison with the SP as both variables can be compared.
Gp	REAL	Proportional gain used to calculate the PID block proportional action.
Td	REAL	Derivative time, in seconds, used to calculate the PID block derivative action.
Ti	REAL	Integral time, in seconds, used to calculate the PID block integral action.
BIAS	REAL	Compensation added to the handled variable.
ManualMV	REAL	Value attributed to the manipulated variable when using the manual mode.
MaxVarMV	REAL	Manipulated variable maximum variation between the actual cycle and the previous cycle. In case is

Input parameters	Type	Description
		zero or negative, the PID block has no MV variation limit.
MaxMV	REAL	Manipulated variable maximum value. In case the calculated value is higher than the configured, the MV will be the same as MaxMV.
MinMV	REAL	Manipulated variable minimum value. In case the calculated value is smaller than the configured, the MV will be the same as MinMV.
DeadBand	REAL	Dead band. Minimum error value able to generate a MV correction in automatic mode, in other words, little errors (smaller than the DeadBand) won't cause any variation in the defined variable.
MaxPV	REAL	Process variable maximum value. In case the PV value is higher than the configured, the PID block will stop the calculus and an error code will be generated in the output.
MinPV	REAL	Process variable minimum value. In case the PV value is smaller than the configured, the PID block will stop the calculus and an error code will be generated in the output.
SampleTime	REAL	Sample time. Defines the PID block call period, in seconds, varying from 0.001s to 1000s. This parameter is not considered if the MeasureST is true.
EnableP	BOOL	When true, enables the PID block proportional action. In case is false, the proportional action is zero.
EnableD	BOOL	When true, enables the PID block derivative action. In case is false, the derivative action is zero.
EnableI	BOOL	When true, enables the PID block integral action. In case is false, the integral action is zero.
DerivPV	BOOL	When true, the derivative action is calculated in the process variable. It's different than zero only when PV is changed. In case if false, the derivative action is calculated in the error, when depends on the SP and PV variables.
Manual	BOOL	When true, enables the manual mode. In case is false, enables the automatic mode. The PID block control mode affects the way the MV and the integral action are calculated.
Direct	BOOL	When true, select the direct control when the MV is included in the answer to be included in PV. In case is false, select the reverse control when MV is subtracted from the answer to be included in PV.
MeasureST	BOOL	When true, the sample time is measured. When false, the sample time is informed by the user in the SampleTime variable.
Restart	BOOL	When true, the PID block is restarted, initializing all variables. It can also be used to erase the integral and derivative actions and the error codes in the block output.
IntegralAction	REAL	Stores the integral action which is eliminated in error state.

Table 4-116 PID. Block Input Parameters

Output parameters	Type	Description
MV	REAL	Manipulated Variable.
EffST	REAL	Real sample time, in seconds, used for the derivative action and MV limit rate calculus.
Eff3ST	REAL	Real sample time from the last three cycles, in seconds, used for the derivative action calculus.
MaxEffST	REAL	Real sample time maximum value, in seconds, since the PID block initialization.
MinEffST	REAL	Real sample time minimum value, in seconds, since the PID block initialization.
ErrorCode	UINT	Error code showed by the PID block. To remove it, the problem must be solved and the block restarted through the Restart variable. The error codes are described below: 0: no error 1: MaxMV < MinMV 2: MaxPV < MinPV 3: PV > MaxPV 4: PV < MinPV 5: Ti < 0,001 s (with the integral action enabled) 6: Td < 0 s (with the derivative action enabled) 7: Gp ≤ 0 8: MaxVarMV < 0 9: DeadBand < 0 10: SampleTime < 0,001 s or SampleTime > 1000 s (with MeasureST = false)

Table 4-117 PID. Block Output Parameters

Figure 4-82 shows the block diagram of a PID loop, as the Nexto CPU execution.

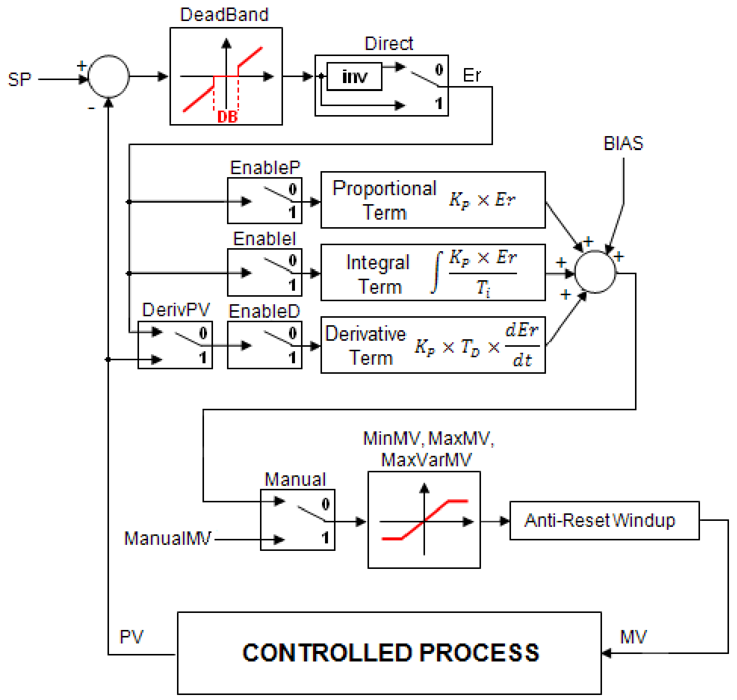


Figure 4-82. PID Diagram

Timer Retain

The time 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 Cold Reset, Original Reset 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 Nexto library are described below (for the library insertion proceeding, see MasterTool IEC XE Programming Manual – MP399608, chapter Library).

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 4-83 depicts the TOF_RET block and Figure 4-84 shows its graphic behavior.

Figure 4-83. 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 4-118. 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 4-119. TOF_RET Output Parameters

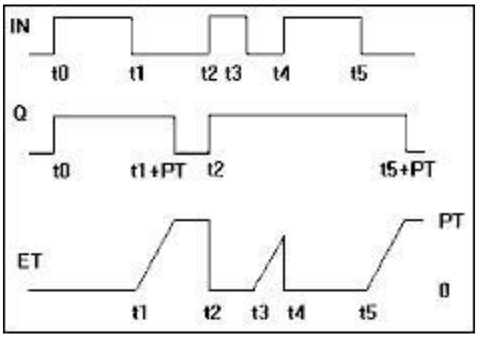


Figure 4-84. TOF_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM MainPrg
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
```

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 4-85 depicts the TON_RET block and Figure 4-86 shows its graphic behavior.

Figure 4-85. 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 4-120. 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 4-121. TON_RET Output Parameters

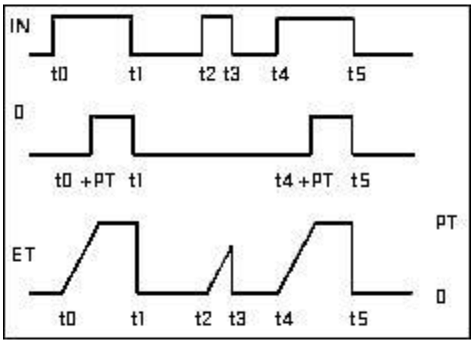


Figure 4-86. TON_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM MainPrg
VAR RETAIN
    bStart : BOOL;
    TON_RET : TON_RET;
END_VAR

// When 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
```

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 4-87 depicts the TP_RET and Figure 4-88 shows its graphic behavior.

Figure 4-87. 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 4-122. TP_RET Input Parameters

Output parameters	Type	Description
Q	BOOL	This variable is true during the counting, otherwise is false.
ET	TIME	This variable shows the current time delay.

Table 4-123. TP_RET Output Parameters

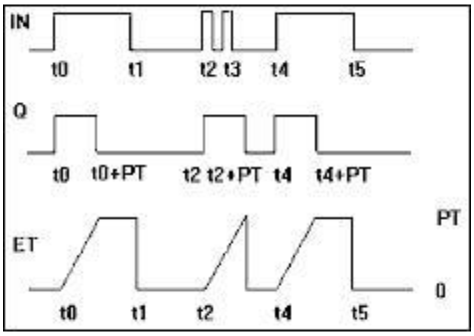


Figure 4-88. TP_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM MainPrg
VAR RETAIN
    bStart : BOOL;
    TP_RET : TP_RET;
END_VAR

// Configure TP_NR
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
```

Non-Redundant Timer

The non-redundant timer is used in applications for the redundant NX3030 CPU which need a timer in the non-redundant program of a half-cluster. This timer does not use the IEC timer, therefore, it will not be synchronized in case the reserve half-cluster assumes the active status and the active one goes for reserve.

The three types of blocks already available in the NextoStandard library of the MasterTool IEC XE software are describe as follows (for doing the procedure of library's inclusion, check MasterTool IEC XE Programming Manual – MP399608, chapter Library.

TOF_NR

The TOF_NR function block implements a delay time for disabling an output and has its functioning and configuration similar to the TOF_RET function block, differentiating itself only for not being redundant nor retentive.

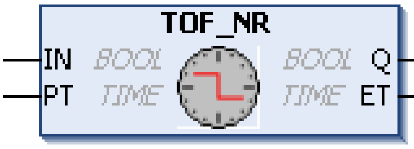


Figure 4-89. TOF_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedProg
VAR
    bStart : BOOL := TRUE;
    TOF_NR : TOF_NR;
END_VAR

// When bStart=FALSE starts the counting
TOF_NR( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TOF_NR.Q = FALSE) THEN
    bStart := TRUE;
END_IF
```

TON_NR

The TON_NR function block implements a delay time to enable an output and has its functioning and configuration similar to the TON_RET function block, differentiating only for not being redundant nor retentive.

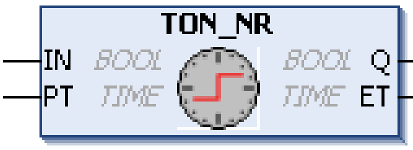


Figure 4-90. TON_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedProg
VAR
    bStart : BOOL;
    TON_NR : TON_NR;
END_VAR

// When bStart=TRUE starts the counting
TON_NR( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TON_NR.Q = TRUE) THEN
    bStart := FALSE;
END_IF
```

TP_NR

The TP_NR function block works as a trigger and has its functioning and configuration similar to the TP_RET function block, differentiating only for not being redundant nor retentive.

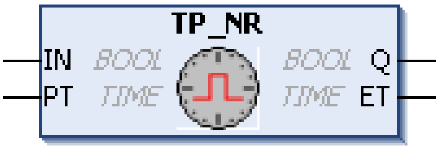


Figure 4-91. TP_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedProg
VAR
   bStart : BOOL;
   TP_NR : TP_NR;
END_VAR

// Configure TP_NR
TP_NR( IN := bStart,
PT := T#20S);

bStart := FALSE;

// Actions executed during the counting
IF (TP_NR.Q = TRUE) THEN
   // Executes while the counter is activated
ELSE
   // Executes when the counter is deactivated
END_IF
```

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 chapter Configuration 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:

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 chapter Maintenance - 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 chapter Maintenance - 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 4-124. Data Type for User Log

The following are described the two functions available in the Lib Logs library on MasterTool IEC XE 1.40. To perform the procedure of inserting a library, see the MasterTool IEC Programming Manual – MP399609, chapter 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.

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 chapter Configuration Memory Card. After the operation the function returns the options for the given type USER_LOG_ERROR_CODES as Table 4-124. When the function return is not USER_LOG_OK, the message was not registered and the function UserLogAdd should be re-executed with the 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 4-92 represents the function UserLogAdd and Table 4-125 the input parameters:

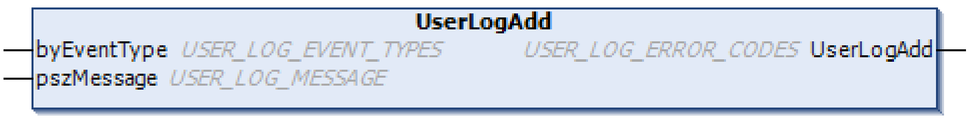


Figure 4-92. UserLogAdd Function

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 4-125. 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 filename 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 MainPrg
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; NX3030
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°
```

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 4-93 represents the function UserLogDeleteAll:

Figure 4-93. UserLogDeleteAll Function

Utilization example in ST language:

```
PROGRAM MainPrg
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.

SNMP

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.

SNMP in Nexto CPUs

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. Table 4-126 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 required objects are described in RFC-1213.

OID	Nome	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 table (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
1.3.6.1.2.1.31	ifMib	Extension to Interfaces, OID 1.3.6.1.2.1.2

Table 4-126. MID 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 information agent is via the Ethernet interfaces NET 1 and NET 2 of the Nexto Series CPUs on TCP port 161. So when the service is active, the agent information can be accessed through any one of the two Ethernet interfaces, if available. It is not possible to provide agent information through Ethernet interfaces NX5000 modules. In Figure 4-94 an example is shown SNMP manager, in which some values are read.

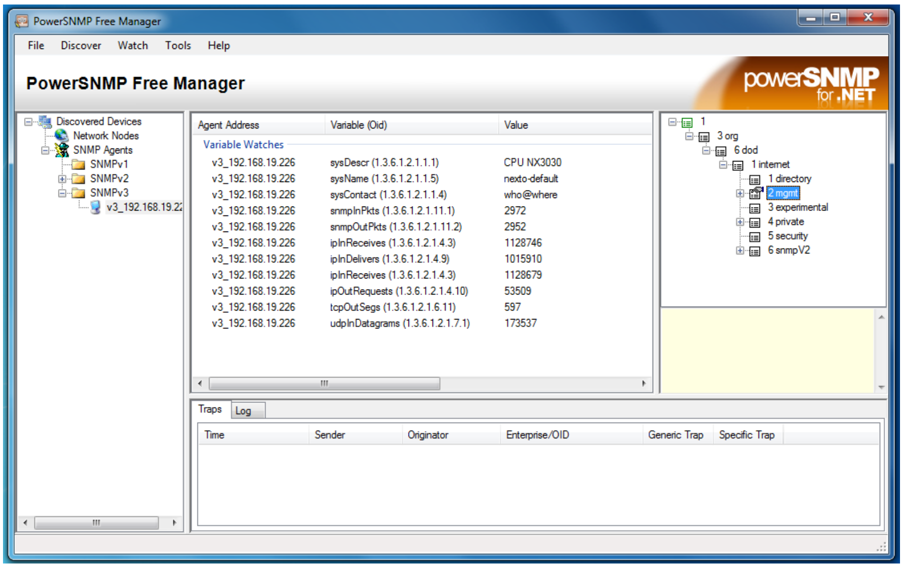


Figure 4-94. SNMP Manager Example

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 web page of the CPU. For details, see the Configuration section.

Private MIB

In addition to support MIB-II, the Nexto Series CPUs support Private MIB from firmware version 1.4.0.25. For it has been reserved for PEN (Private Enterprise Number) with the unique number 43427 for products Altus. Thus, all private objects of Nexto Series can be accessed through the OID .1.3.6.1.4.1.43427.1 (iso.org.dod. internet.private.enterprise.Altus SA.Nexto). In this OID three relevant subdivisions Nexto CPUs, as seen in Figure 4-95.

All private objects are described in MIBs ALTUS-NEXTO-NX3004-MIB, ALTUS-NEXTO-NX3005-MIB, ALTUS-NEXTO-NX3010-MIB, ALTUS-NEXTO-NX3020-MIB and ALTUS-NEXTO-NX3030-MIB and can be found in www.altus.com.br/site_en/.

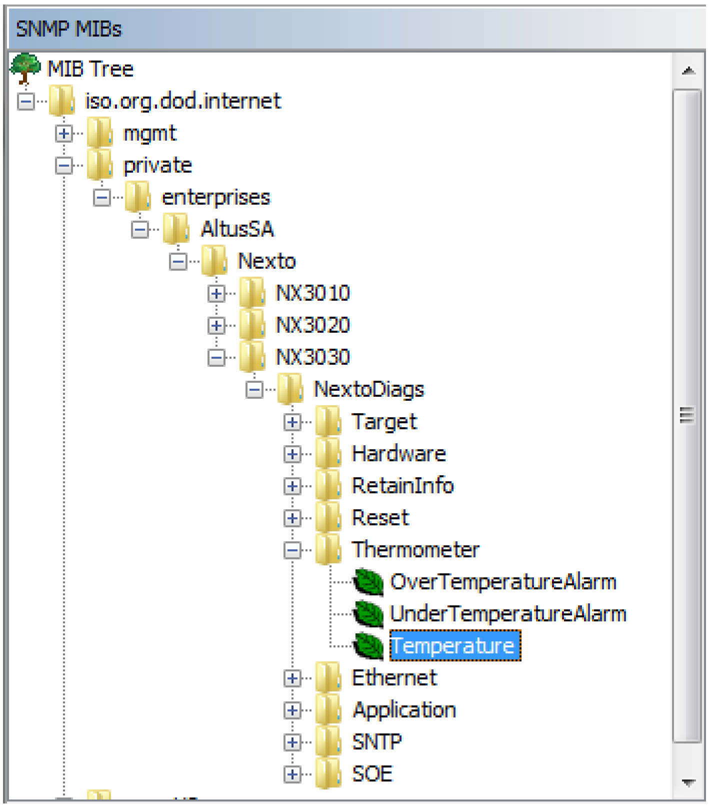


Figure 4-95. OID Tree View

The objects available via SNMP in the Nexto Series CPUs are existing diagnostics, which are important for managing networks. These objects can be seen in Table 4-127.

To access them through an SNMP manager the user must make requests from OID .1.3.6.1.4.1.43427.1.4.1 to NX3004, .1.3.6.1.4.1.43427.1.5.1 to NX3005, .1.3.6.1.4.1.43427.1.1.1 to NX3010, 1.3.6.1.4.1.43427.1.2.1 to NX3020 and.1.3.6.1.4.1 .43427.1.3.1 to NX3030.

For example, if the user wants to monitor the internal temperature of a CPU NX3030, the corresponding OID in this case will be .1.3.6.1.4.1.43427.1.3.1.5.3 (iso.org.dod.internet.private.enterprise.AltusSA.Nexto .NX3030.NextoDiags.Thermometer.Temperature).

Groups	Diagnostic	Description
Target	CPUModel	NX30XX
	CPUVersion	Firmware Version
	BootloadVersion	Bootloader Version
	AuxProcVersion[1]	Aux Proc Version
Hardware	AuxProcFailure[1]	Failure between Aux Proc and Main Proc
	RTCFailure	The main Proc is not able to communicate with the RTC (CPU clock)

Groups		Diagnostic	Description
		ThermometerFailure	Communication failure between Thermometer and Main Proc
		LCDFailure[1]	Communication failure between the LCD and Main Proc
RetainInfo		CPUInitStatus	Initialization State of CPU: 01: Hot Start 02: Warm Start 03: Cold Start note: These variables are reset in all energization.
		CPUColdStartCounter	Cold Start Counter: It will only be increased due to hot removal of the CPU of the bus and not due to the Reset Cold command the of MasterTool IEC XE (0 to 65535)
		CPUWarmStartCounter	Warm Start Counter: It will only be incremented during energization sequence of the system and not due to the Reset Warm command of MasterTool IEC XE (0 to 65535)
		CPUHotStartCounter[1]	Counter of disorders smaller than the time of power failure to support the CPU (0 to 65535)
		RTSResetCounter	Reset counter made by the RTS (Runtime System) (0 to 65535)
Reset		BrownOut	CPU reboot due to a power supply failure in the last initialization
		Watchdog	CPU reboot due to watchdog active in the last initialization
Thermometer		UnderTemperatureAlarm	Alarm generated due to internal temperature be in 0° C or below
		OverTemperatureAlarm	Alarm generated due to internal temperature be in 85° C or above
		Temperature	Internal Temperature
Ethernet	NET 1	ModbusRTUEthClient1	Modbus RTU via TCP Client
		ModbusEthClint1	Modbus TCP Client
		ModbusRTUEthServer1	Modbus RTU via TCP Server
		ModbusEthServer1	Modbus TCP Server
	NET 2	ModbusRTUEthClient2[1]	Modbus RTU via TCP Client
		ModbusEthClint2[1]	Modbus TCP Client
		ModbusRTEthServer2[1]	Modbus RTU via TCP Server
		ModbusEthServer2[1]	Modbus TCP Server
Application		CPUState	Reports the operating status of the CPU: 01: All user applications are in Start Mode 03: All user applications are in Stop Mode
		ForcedIOs	There are one or more forced IO points
SNTP		ServiceEnable	SNTP service enabled
		ActiveTimeServer	Indicates which server is active: 00: No server active 01: Primary server active 02: Secondary server active
		PrimaryServerDownCount	Counter of how many times primary server was unavailable (0 to 65535)
		SecondaryServerDownCount	Counter of how many times secondary server was unavailable (0 to 65535)
		RTCTimeUpdatedCount	Counter of times the RTC has been updated by the NTP service (0-4294967295)
		LastUpdateSuccessful	Indicates status of last update: 00: Not updated 01: Last update failed 02: Last update had success
		LastUpdateTimeServer	Indicates which server was used in the last update: 00: Any update 01: Primary Server 02: Secondary Server
		LastUpdateTimeDayOfMonth	Day of last RTC update

Groups		Diagnostic	Description
		LastUpdateTimeMonth	Month of last RTC update
		LastUpdateTimeYear	Year of last RTC update
		LastUpdateTimeHours	Hour of last RTC update
		LastUpdateTimeMinutes	Minute of last RTC update
		LastUpdateTimeSeconds	Seconds of last RTC update
		LastUpdateTimeMilliSeconds	Milliseconds of last RTC update
SOE	SOE1	ConnectionStatus1[1]	Connection status of client 01
		OverflowStatus1[1]	Client 01 event queue status: FALSE – OK TRUE – Queue limit exceeded
		EventsCounter1[1]	Client 01 event counter on queue
	SOE2	ConnectionStatus2[1]	Connection status of client 02
		OverflowStatus2[1]	Client 02 event queue status: FALSE – OK TRUE – Queue limit exceeded
		EventsCounter2[1]	Client 02 event counter on queue

Table 4-127. Diagnostics via SNMP

(1) These diagnostics was not available at the CPUs NX3004 and NX3005.

> ATTENTION:
> Items Ethernet NET2, SNTP and SOE are available only for CPUs NX3020 and NX3030

Configuration

SNMP settings can be changed through the web page, in the CPU Management tab in the SNMP menu. To access the settings, you must first log in, as shown in Figure 4-96.

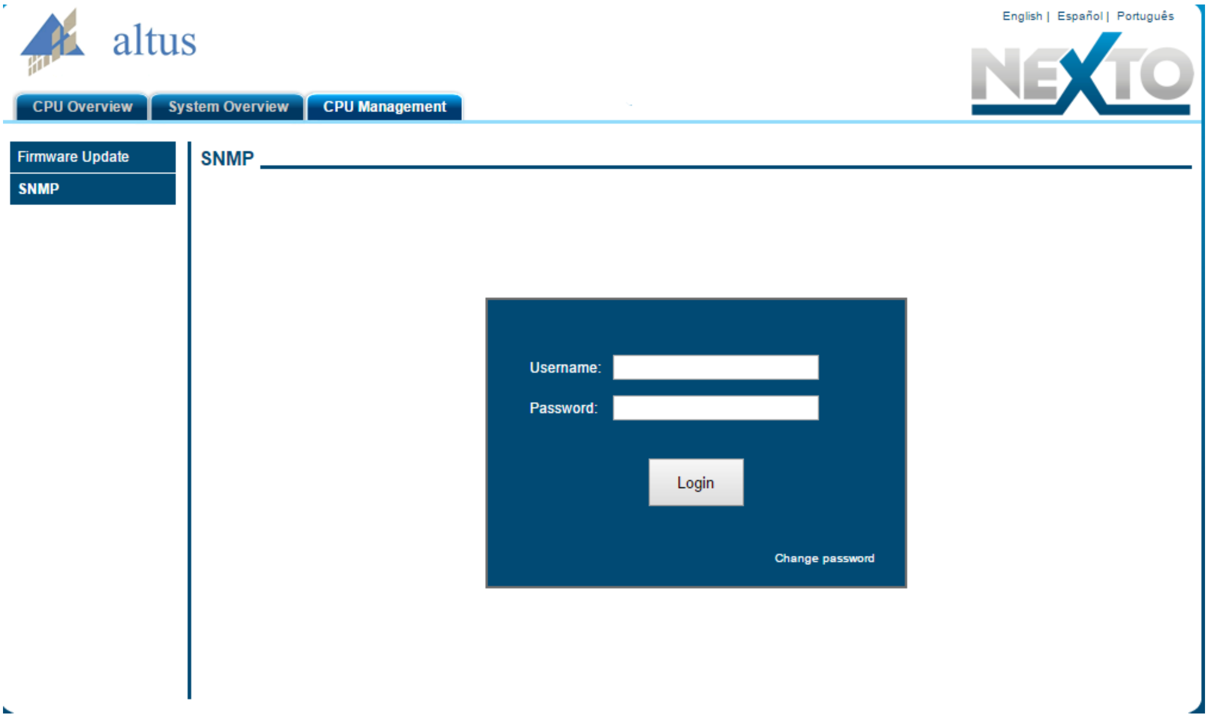


Figure 4-96. SNMP Login screen

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.

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 fields and rwcommunity 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 can be visualized in Figure 4-97.

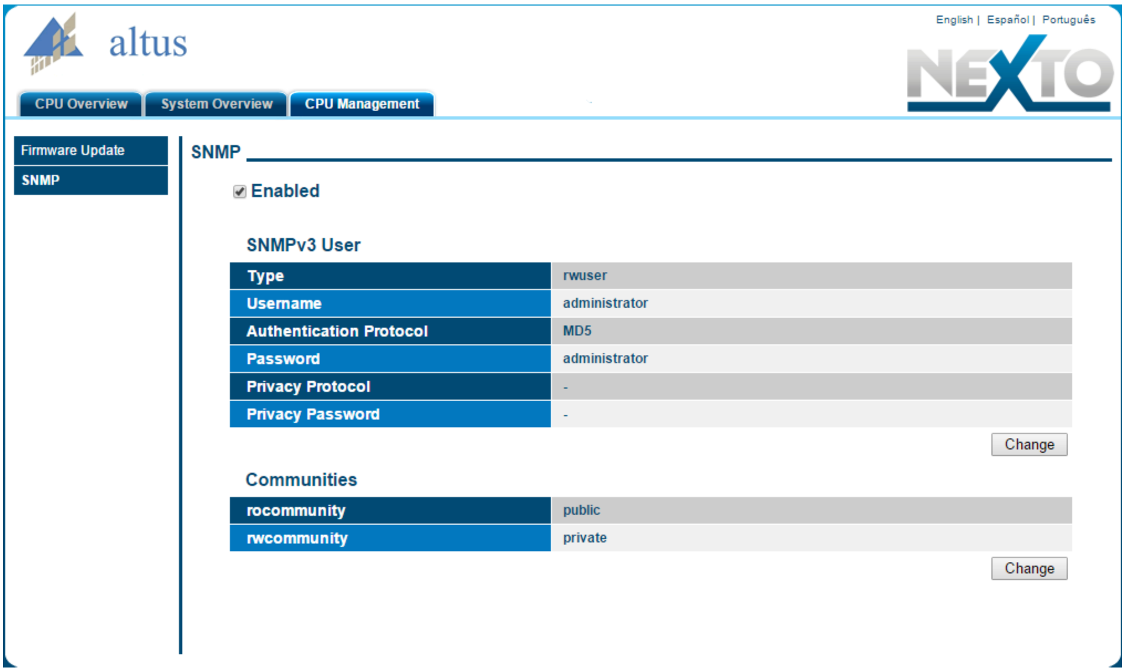


Figure 4-97. SNMP status configuration screen

ATTENTION: If the displayed screens are different from displaying in the browser, a browser cache cleanup is necessary.

ATTENTION: The user and password to login on the website of SNMP settings and to access the agent via SNMP protocol are equal.

User and SNMP Communities

To access the SNMPv1 / v2c of the Nexto Series CPUs, there are two communities, according to Table 4-128.

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

Table 4-128. SNMP v1/v2c Default Communities info

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

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

Table 4-129. SNMP v3 User info

For all settings of communities, user and password, some limits must be followed, as the Table 4-130:

Configurable item	Minimum Size	Max Size	Allowed Characters
rocommunity	-	30	[0-9][a-z][A-Z]@$*_.
rwcommunity	-	30	[0-9][a-z][A-Z]@$*_.
V3 User	-	30	[0-9][a-z][A-Z]@$*_.
Password v3	8	30	[0-9][a-z][A-Z]@$*_.

Table 4-130. SNMP settings limits

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 a 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 MT85000 – MU299609, in the User Management and Access Rights chapter.

5. Initial Programming

The main goal of this chapter is to help in the programming and configuration of Nexto Series CPUs so that the user will be able to take the first steps before starting a controller programming.

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 User Manual MasterTool IEC XE - MU299609, Programming Manual MasterTool IEC XE - MU399609 or IEC 61131-3 standard.

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 5-1, where, for CPUNEXTO string, the letter U 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 5-1, shows little and big endian organization.

MSB ← Little-endian (Traditional) → LSB								
BYTE	%QB7	%QB6	%QB5	%QB4	%QB3	%QB2	%QB1	%QB0
	C	P	U	N	E	X	T	O
WORD	%QW3		%QW2		%QW1		%QW0	
	CP		UN		EX		TO	
DWORD	%QD1				%QD0			
	CPUN				EXTO			
LWORD	%QL0							
	CPUNEXTO							
HSB <– Big-endian (NEXTO) –> 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 5-1. Example

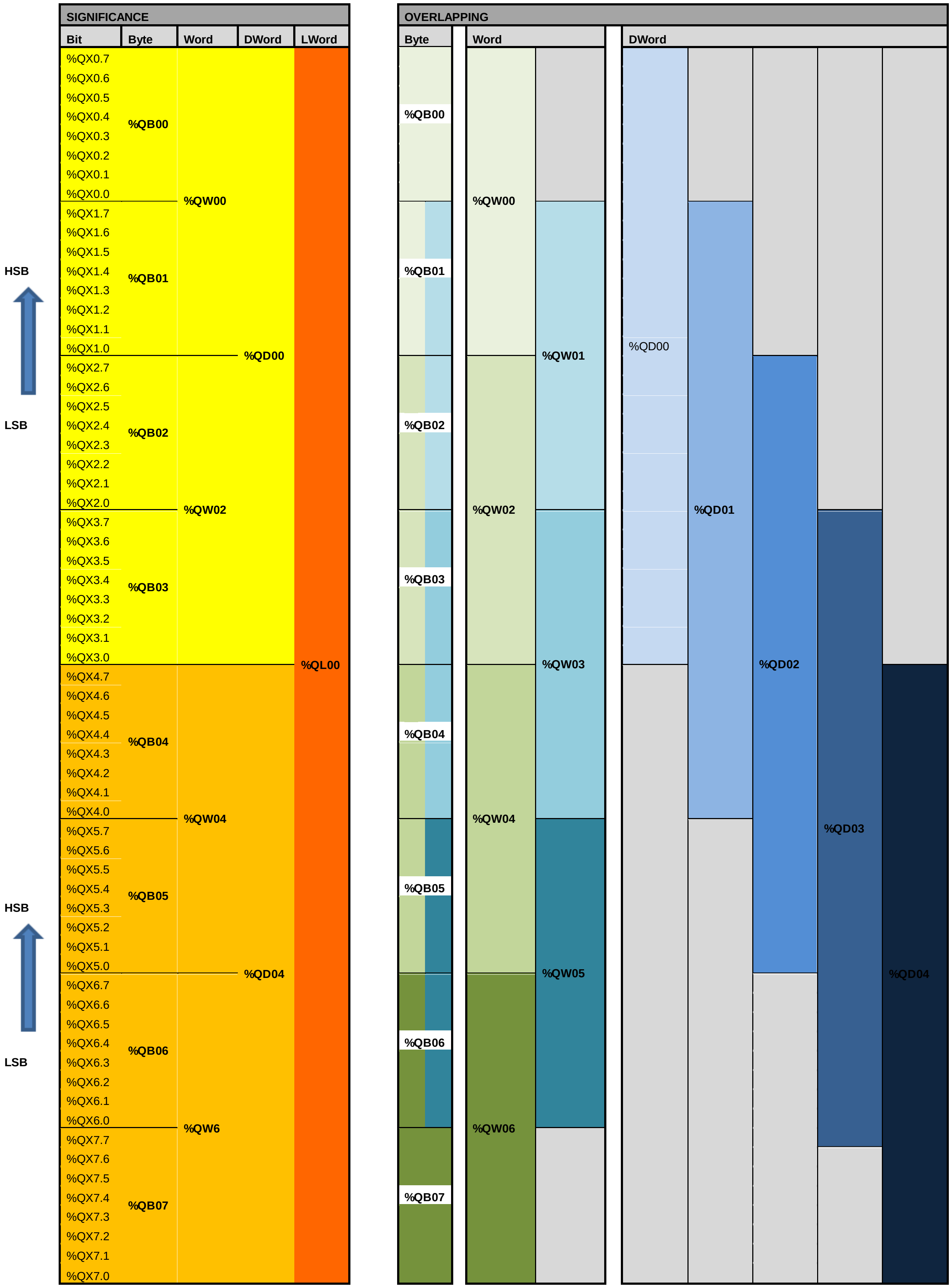


Table 5-2. Memory Organization and Access

The Table 5-2 shows the organization and memory access, illustrating the significance of bytes and the disposition of other variable types, including overlapping.

Project Profiles

A project profile in the MasterTool IEC XE is a group of rules, common features and patterns used in an industrial automation solution development, a profile which implies in the application implementation form. With the diversity of application types supported by the Nexto Series Runtime System, following a profile is a way to simplify the programming complexity. The applications can be created according the following profiles:

- Single
- Basic
- Normal
- Expert
- Custom
- Machine Profile

For each profile defined for the RTS, MasterTool IEC XE can provide numerous compatible templates. When the user selects a template as a model in project creation, the new application will be developed as a specific profile, adopting rules, characteristics and standards defined by the profile associated with the template. Each project profile defines standard names for the tasks and programs, which are pre-created by the project templates. The developer is required to follow the nomenclature strictly for the tasks, but can follow or not the suggested names for the respective programs.

To ensure the project compatibility to a certain profile during the development, two approaches are used:

- MasterTool IEC XE only allows the creation of projects based on a template, selecting at the time that profile is chosen.
- During code building, MasterTool IEC XE verify all the rules set for the profile selected in the project.

The following sections detail the characteristics or standards 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 stress 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 main task, MainTask. In case that the option *Enable I/O Task Update per Task* can’t be used, present as of MasterTool IEC XE version 2.01.

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 MaisTask.

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 5-3. Single Profile Tasks

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 5-4. 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	TimeInterruptTask00	01	Cyclic	20 ms	-

Table 5-4. Basic Profile Tasks

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 it 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 5-5. 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	TimeInterruptTask00	01	Cyclic	20 ms	-

Table 5-5. Normal Profile Tasks

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 5-6, 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 Experienced 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	TimeInterruptTask00	01	Cyclic	20 ms	-
ExternInterruptTask01	ExternInterruptPrg01	11	External	-	IO_EVT_1
TimeInterruptTask01	TimeInterruptPrg01	09	Cyclic	30 ms	-
FreeTask	FreePrg	31	Continuous	-	-

Table 5-6. Expert Profile Tasks

Custom

The Custom project profile allows the developer to explore all the potential of the Runtime System implemented in the Nexto Series central processing units. No functionality is disabled; no priority, task and programs association or nomenclature 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. However, the user can generate the same elements available for the Expert profile.

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	TimeInterruptTask00	01	Cyclic	20 ms	-
ExternInterruptTask01	ExternInterruptPrg01	11	External	-	IO_EVT_1
TimeInterruptTask01	TimeInterruptPrg01	09	Cyclic	30 ms	-
FreeTask	FreePrg	31	Continuous	-	-

Table 5-7. Custom Profile Tasks

Machine Profile

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

This profile is characterized by allowing shorter intervals in the MainTask, allowing faster execution of user code. This 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.

This profile may further include an interruption task, called TimeInterruptTask00, with a higher priority than the MainTask, and hence, can interrupt its execution at any time.

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

Table 5-8. Machine Profile Tasks

General Table

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

Table 5-9. 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.

Maximum Number of Tasks

The maximum number of tasks 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.

Table 5-10 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	NX3004/NX3005						NX3010						NX3020						NX3030					
		S	B	N	E	P	M				E	C	M	S	B	N	E	C	M	S	B	N	E	C	M
Configuration Task (Task WHSB)	**Cyclic**	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0
User Tasks	**Cyclic**	1	1	15	15	15	2	1	1	15	15	15	2	1	1	23	23	23	2	1	1	31	31	31	2
	Triggered by Event	0	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0	23	0	0	0	0	0	31	0
	Triggered by External Event	0	1	1	14	15	0	0	1	1	14	15	0	0	1	1	22	23	0	0	1	0	30	31	0
	Freewheeling	0	1	0	1	15	0	0	1	0	1	15	0	0	1	0	1	23	0	0	1	0	1	31	0
	Triggered by State	0	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0	23	0	0	0	0	0	31	0
NETs – Client or Server Instances	**Cyclic**	4						4						8						16					
COM (n) – Master or Slave Instances	**Cyclic**	1						1						1						1					
TOTAL		16						16						24						32					

Table 5-10. 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.

CPU Configuration

The Nexto CPU configuration is based on the action of structuring the diagnostics area, the retentive and persistent memory area and hot swap mode, among other parameters.

The user must double-click on the Nexto CPU, in the device tree, as shown on Figure 5-1, and configure the field as described in the CPU Configuration.

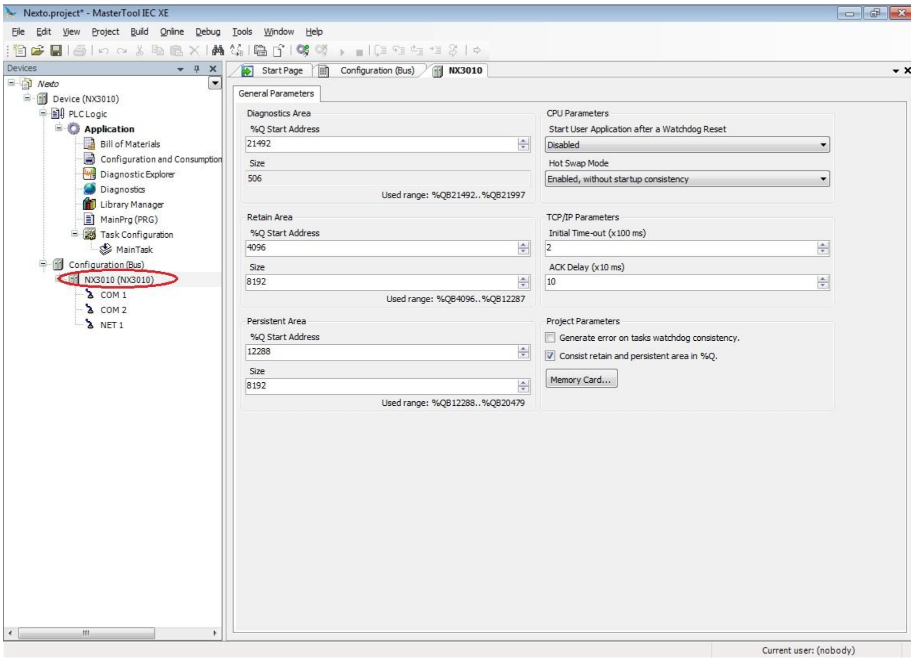


Figure 5-1. CPU Configuration

Besides that, for the communication between the CPU and the MasterTool IEC XE to be possible, the Ethernet NET 1interface must be configured, as described in the NET 1. Double-clicking on the CPU NET 1 icon, in the devices tree, a new tab will appear for the configuration of the communication network where the module is connected.

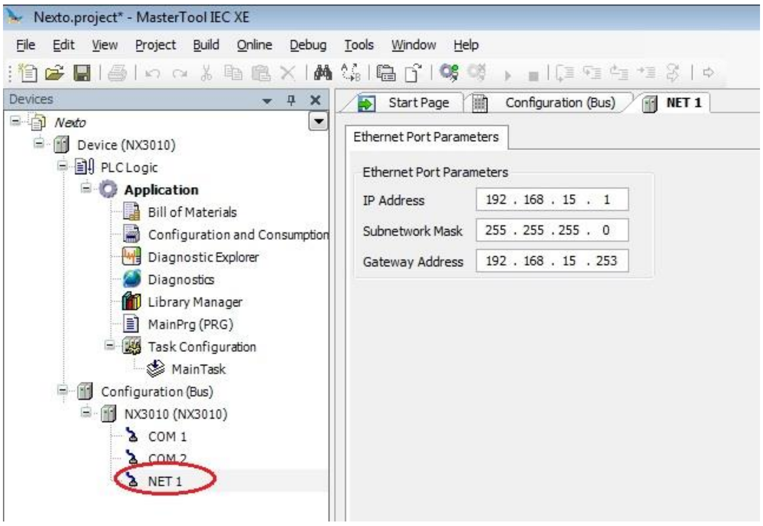


Figure 5-2. Configuring the CPU Communication Port

In case the CPU with the configured IP is not found in the network or the active CPU has a different IP, a message will appear on the screen during the Login, requesting to the user the possibility of changing the previous IP by the configured (Yes option) or No and quit sending the project.

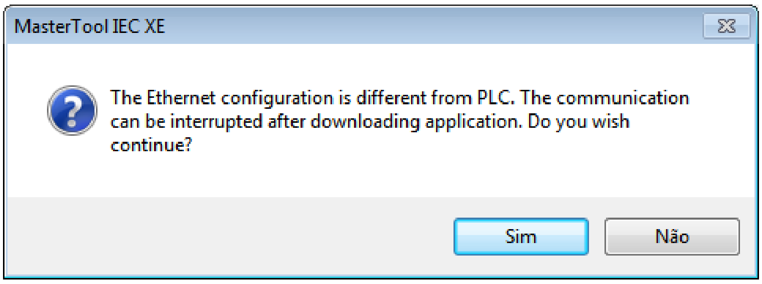


Figure 5-3. IP Configuration Warning

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.

Inserting a Protocol Instance

The Nexto Series CPUs, as described in the General Features chapter, offer protocols as the MODBUS. The desired protocol instance must be added and configured in the communication interface as shown in the Protocols Configuration chapter.

Two cases of MODBUS protocol insertion are described below: one in the serial interface and the other in the Ethernet interface.

MODBUS RTU

The first step for the MODBUS RTU configuring, in slave mode, is to include the instance in the desired COM (COM 1 in this case) by clicking with the right button on the COM and select "Add Device...", as shown on Figure 5-4:

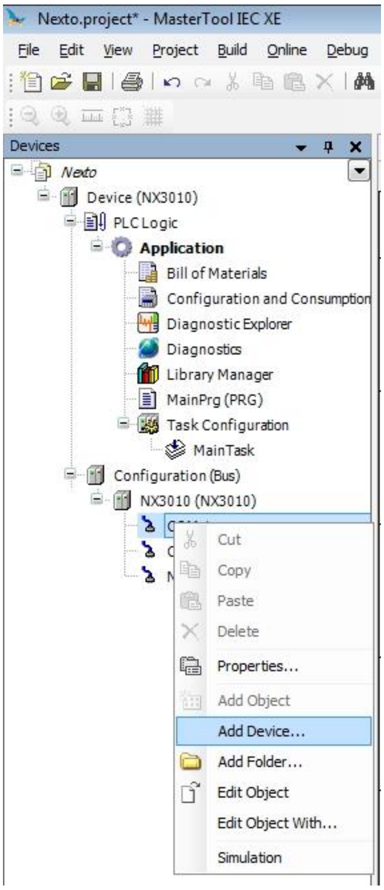


Figure 5-4. Adding an Instance

After that, the available protocols for the user will appear on the screen. Define the protocol configuration mode selecting "MODBUS Symbol RTU Slave", for symbolic mapping setting or "MODBUS RTU Slave", for direct addressing (%Q) and click on Add Device, as depicted on Figure 5-5:

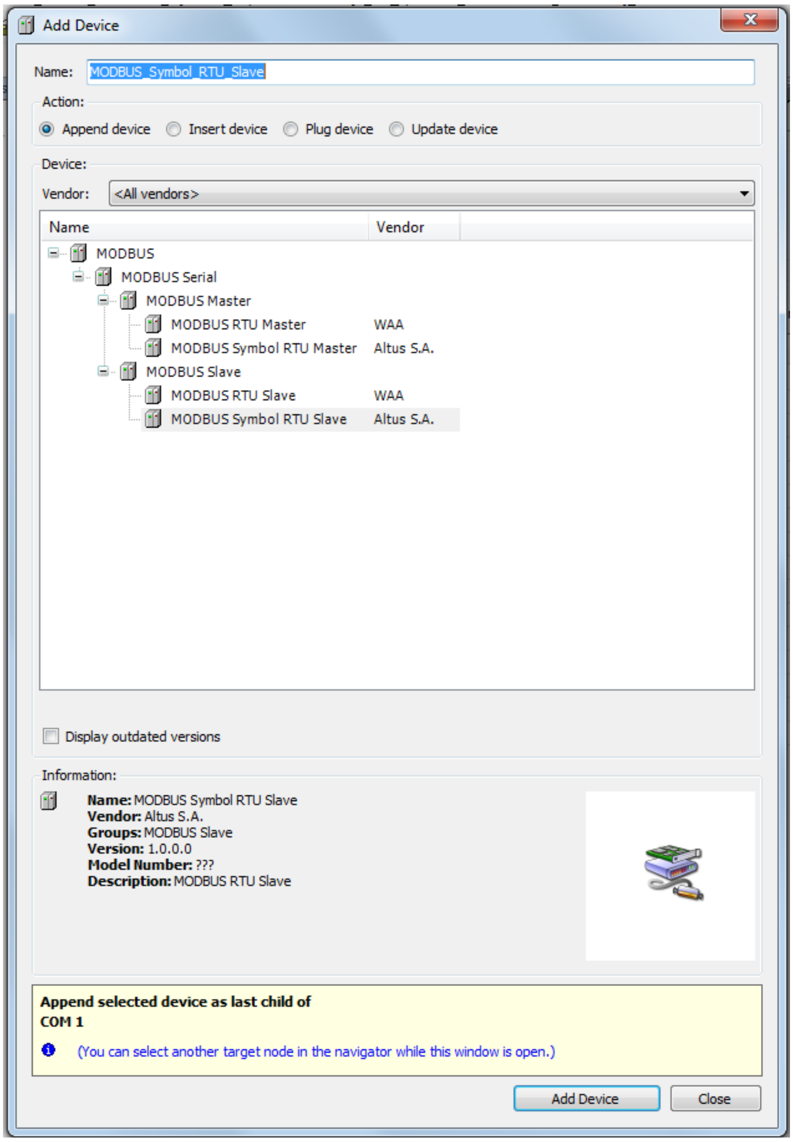


Figure 5-5. Selecting the Protocol

MODBUS Ethernet

The first step to configure the MODBUS Ethernet, in client mode, 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 5-6:

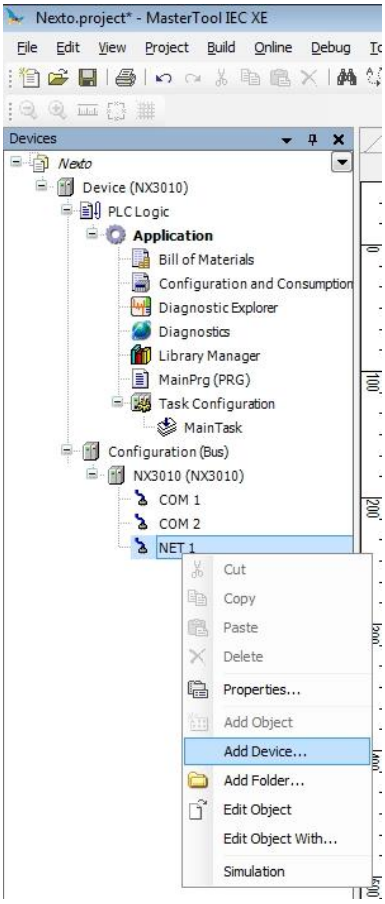


Figure 5-6. Adding the Instance

After that, the available protocols for the user will appear on the screen. Define the protocol configuration mode selecting “MODBUS Symbol Client”, for symbolic mapping setting or “MODBUS Client”, for direct addressing (%Q) and click on Add Device, as depicted on Figure 5-7:

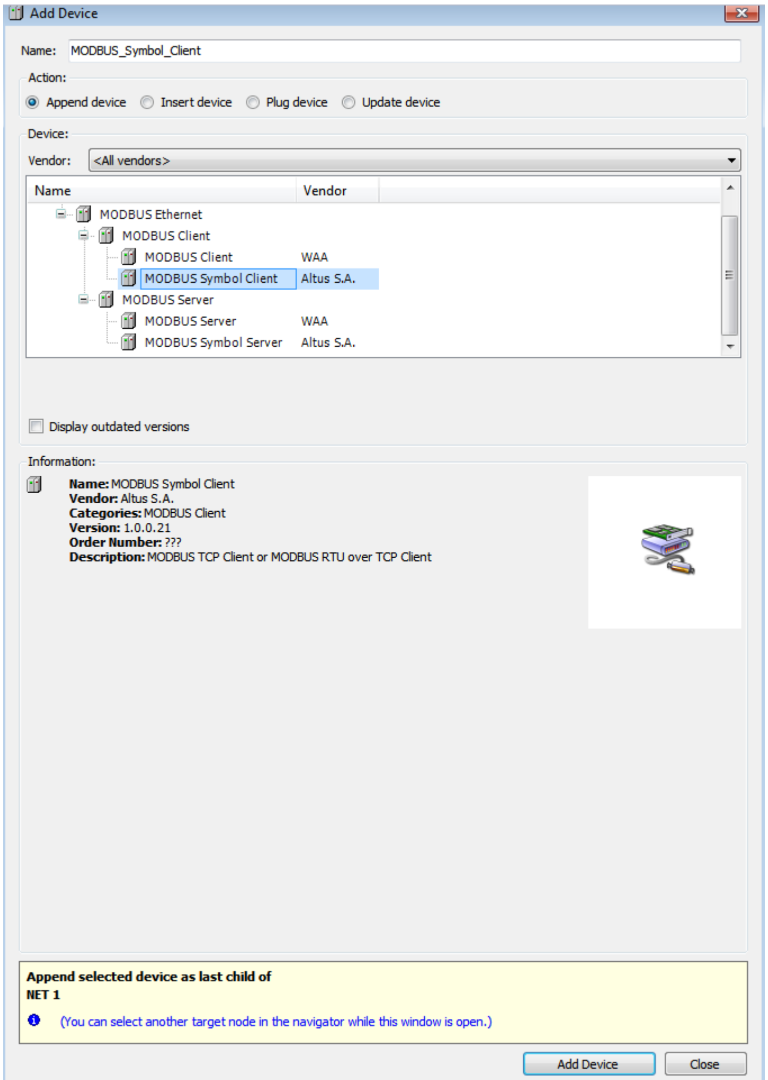


Figure 5-7. Selecting the Protocol

Finding the Network

As there is the possibility of more CPUs being connected to the network, the user must find all communication units and select the desired one.

Initially, the option Device must be accessed, in the device tree, double-clicking on it. In the tab “Communications Settings”, select the Gateway and click on “Scan network”. On the "Communication Settings" tab select the Gateway or in case there is no Gateway or the user wants to add a new gateway, click the "Add Gateway", setting your IP in the window that opens. For mapping devices present in the network, click on "Scan Network ".

Then, the user must wait until the MasterTool IEC XE software searches and show the available CPUs in the network.

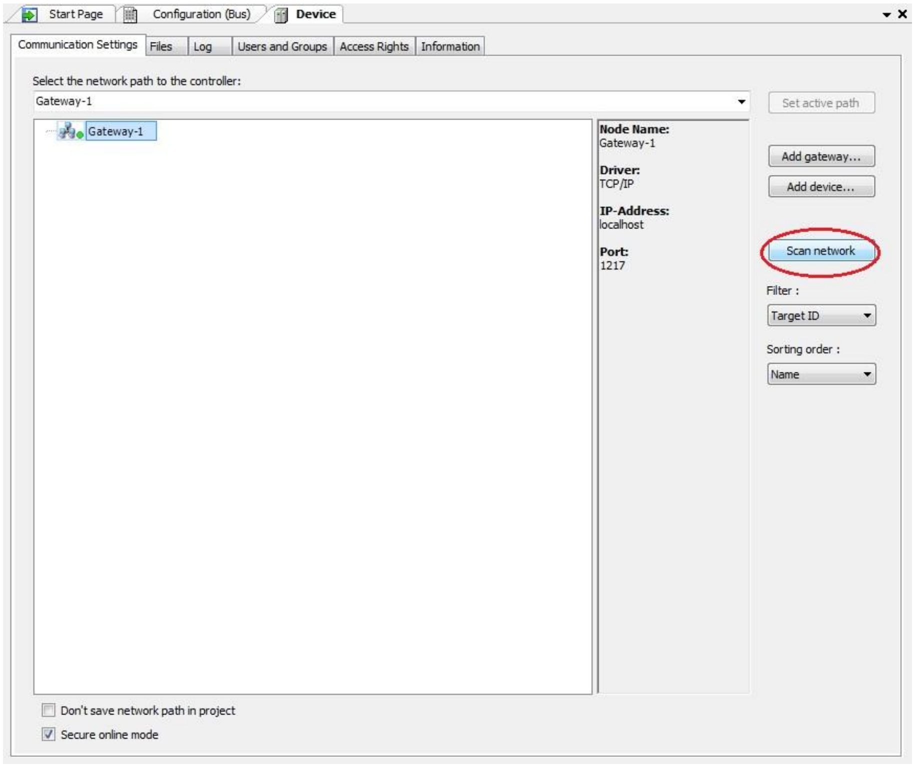


Figure 5-8. Finding the CPU

Following, the desired CPU must be selected and the option "Set active path" clicked, to activate the CPU and to inform the configuration software the CPU should communicate and send the project.

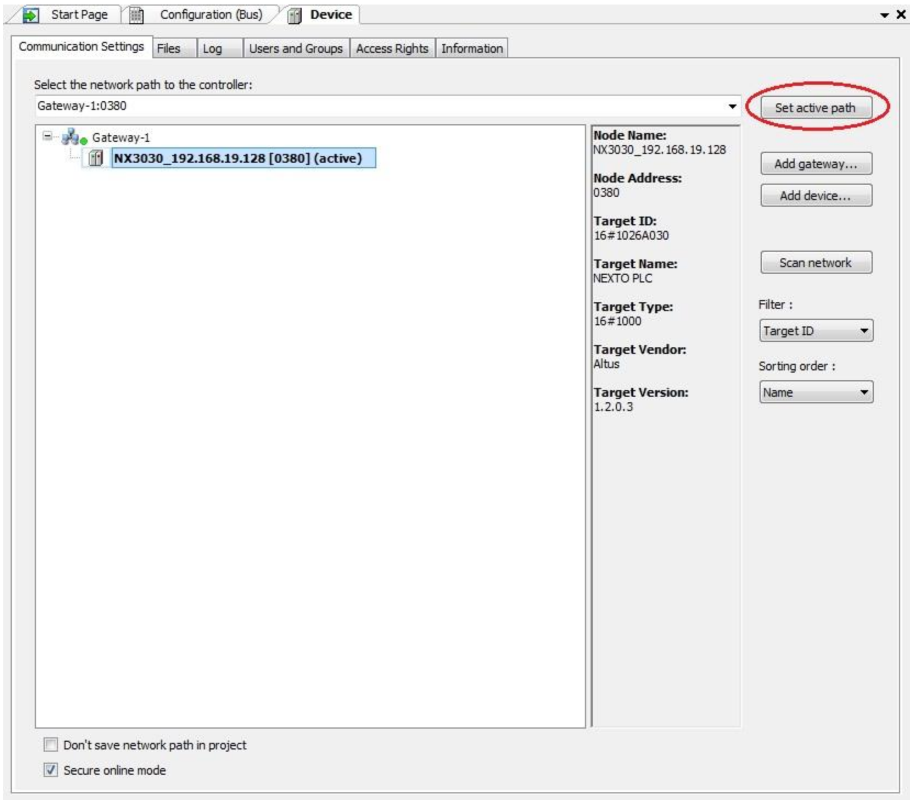


Figure 5-9. Activating the CPU

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

Login

After the application has been compiled and the errors found corrected, the project must be sent to the CPU. For that to be possible, the operation Login in the MasterTool IEC XE software must be executed. This operation may take a few seconds, depending on the size of the generated file. To execute the Login, go to Online menu and click on the option “Login”, as shown on Figure 5-10.

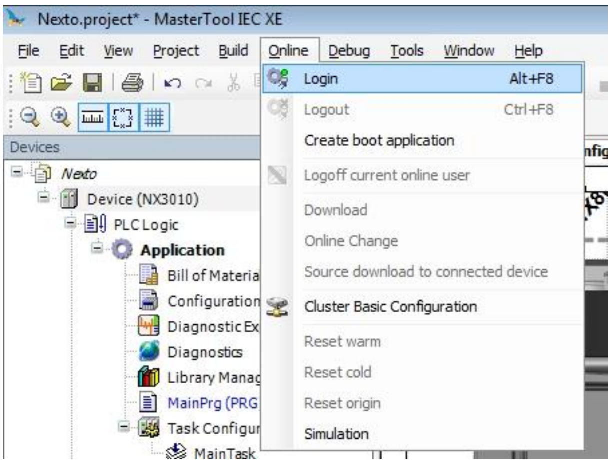


Figure 5-10. Sending the Project to the CPU

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.

Figure 5-11 shows the message the MasterTool IEC XE presents in case the new project, which is being sent, is different from the project already existent inside the CPU. The available options are the following:

- 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

ATTENTION: Before version 2.01 of MasterTool IEC XE, when the Login with online change was executed, the application was not saved in the program memory. It was necessary to run the "Create Boot Application" in the Communication menu without logging out, for the application to be recorded in program memory. From version 2.01 this operation came to be carried out automatically without the need to run the command.

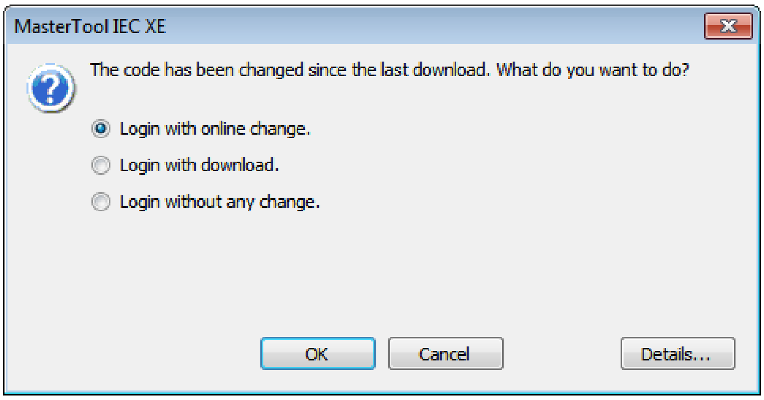


Figure 5-11. CPU Project Updating

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).

Figure 5-12 shows a message the MasterTool IEC XE shows when only changes in the application variables were done; turning impossible the new project sending and updating on a CPU new cycle, which is in run mode (see Run Mode). Therefore, the MasterTool IEC XE requests whether the login must be executed as download and the CPU stopped (see Stop Mode) or the operation must be canceled.

PS.: The button "Details..." presents the changes made in the application.

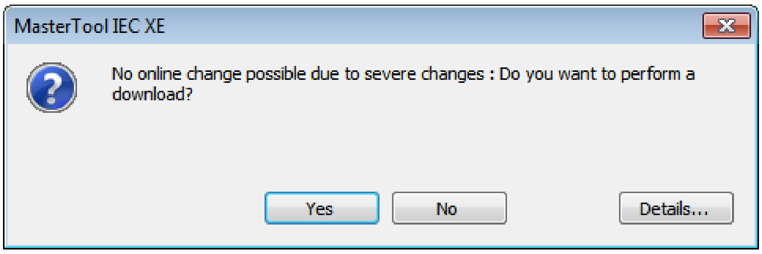


Figure 5-12. Variable Changes

At the first time an application is sent to the CPU, the message shown on Figure 5-13 will appear on the MasterTool IEC XE screen.

Figure 5-13. First Application Sending

Run Mode

Right after the project has been sent to the CPU, the application will not be immediately executed (only if an online – online change transmission were made). For that to happen, the command "Start" must be selected. This function allows the user to control the execution of the application sent to the

CPU. Besides, it allows initial values to be pre-configured, in order to turn possible the CPU updating on the first cycle.

To select such functionality, the option “Start”, from the Debug menu, must be clicked, as shown on Figure 5-14.

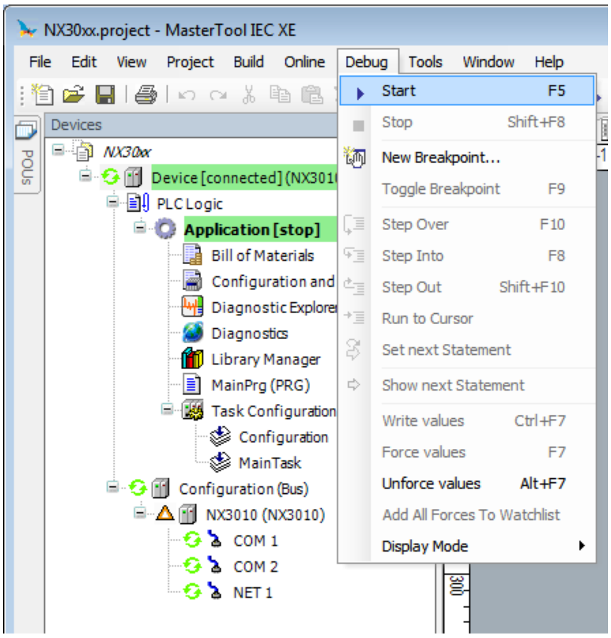


Figure 5-14. Starting the Application

Figure 5-15 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 forced and visualized by the user. In case the variables are forced through the F7 command on the keyboard, the CPU will indicate this condition on the graphic display. For further details see Graphic Display chapter.

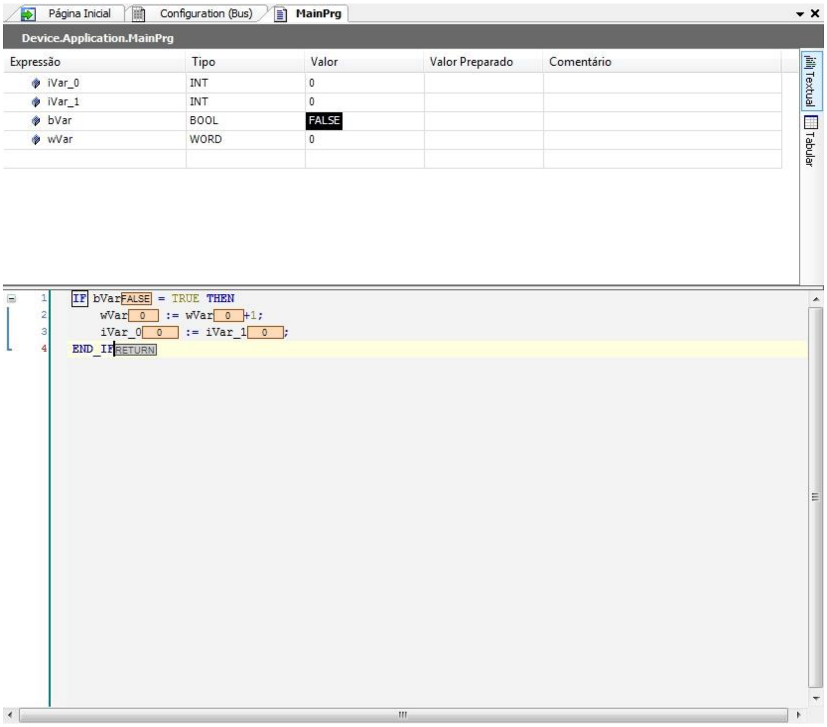


Figure 5-15. Program Being Executed

In case the CPU is initialized with an application already internally stored, it automatically goes to Run Mode, without the need for a MasterTool IEC XE command.

Stop Mode

For CPU execution interruption, without losing the connection with the MasterTool IEC XE software, the user must select the “Stop” option, available at the menu Debug, as shown on Figure 5-16.

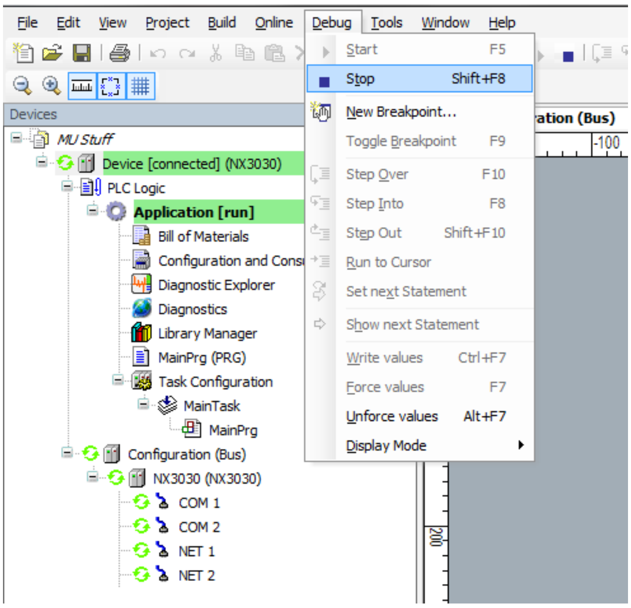


Figure 5-16. Stopping the Application

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

Writing and Forcing Variables

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

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

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

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

When a forced writing is done into a redundant variable of the Active PLC, the MainTask execution time will be impacted, in both Active and Stand-by PLC. This occurs because the two half-clusters will exchange in each cycle information about the forced variables. Therefore, when forcing variables in a redundant system, the user should consider the time added to the task execution time.

The Table 5-11 exemplifies the medium execution time added to the MainTask with a number of forced variables:

	Active PLC			Stand-by PLC		
Execution Time	**50 ms**	**100 ms**	**200 ms**	**50 ms**	**100 ms**	**200 ms**
Increase with 10 forcings	2.4 %	2.2 %	1.7 %	4.0 %	3.4 %	2.0 %
Increase with 50 forcings	12.0 %	9.2 %	6.0 %	18.0 %	12.0 %	8.0 %
Increase with 128 forcings	26.0 %	21.0 %	16.0 %	56.0 %	34.0 %	22.5 %

Table 5-11. The Influence of the Variables Forcing in a Redundant PLC

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.

Logout

In case the user option is to finalize the communication with the CPU, the command "Logout" must be used, placed in the Online menu, as shown on Figure 5-17.

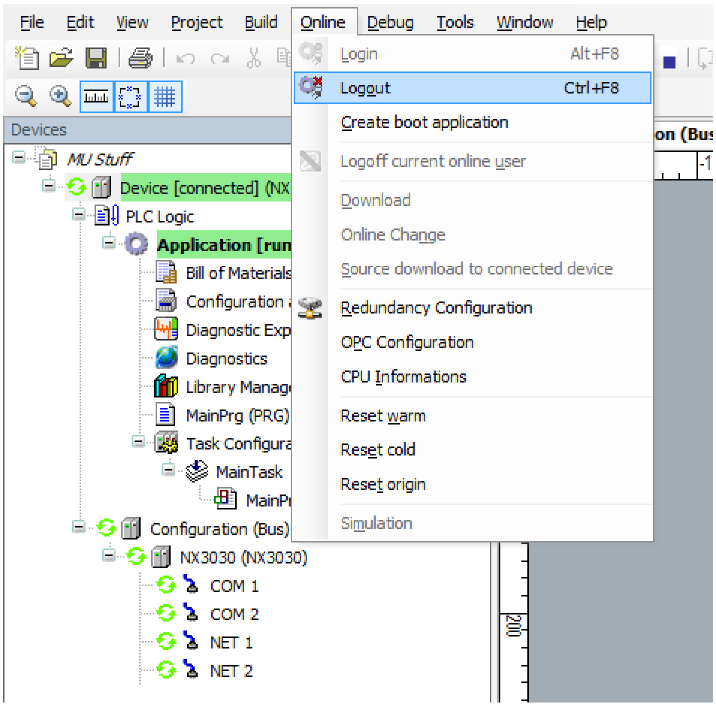


Figure 5-17. Interrupting the Communication with the CPU

Project Upload

Nexto Series CPUs allow the project storing in the product memory, which can be uploaded and reused through the MasterTool IEC XE software.

To store a project in the product memory, the CPU must be connected (Login) and the option to send the source download implicitly at program download must be selected.

To upload the project previously stored, the options shown on Figure 5-18 must be selected.

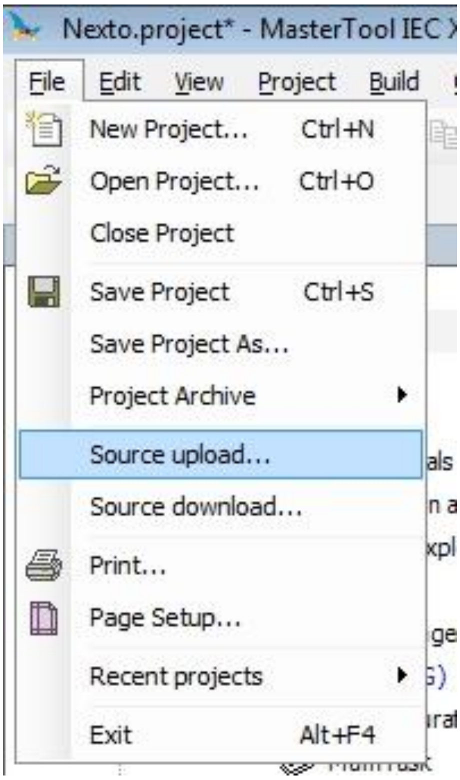


Figure 5-18. Project Upload Option

After, the desired CPU has to be selected and the OK button clicked as shown on Figure 5-19.

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.

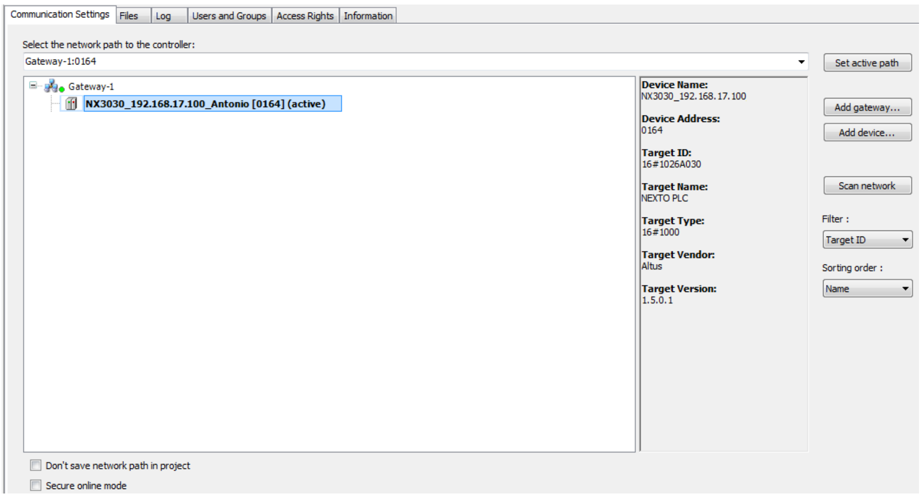


Figure 5-19. Selecting the CPU

ATTENTION: The memory size area to store a project in the Nexto CPUs is defined on Table 2-5.

ATTENTION:
The upload recovers the last project stored in the controller as described in the previous paragraphs. In case only the download for execution of a specific applicative occurs, it will not be possible to be recovered through upload.

CPU Operating States

Run

When a CPU is in Run mode it indicates that all application tasks are in execution.

Stop

When a CPU is in Stop mode it indicates that the application tasks are stopped. The variable values in the tasks are kept with the current value and output variables assume defined values by the user.

When a CPU switches for the Stop mode from the submitting of an application, the variables in the application tasks will be lost with the exception of the persistent type variables. The output variables will assume the defined value by the user and then the outputs value will switch for the safe state. As the new application is loaded, the output variables will assume again the defined value by the user.

Breakpoint

When a debugging mark is reached in a task, it is interrupted. All the active tasks in the application will not be interrupted, they will continue their execution. In this mode it is possible to go through a program in the Online mode. A step by step can be executed and the debugging interruptions positions depend on the editor.

For further information about the use breakpoints, please consult the MasterTool IEC XE Utilization Manual - MU299609.

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.

Reset Warm

This command puts the CPU in Stop mode and starts all the application tasks variables, except the persistent and retentive type variables. The variables started with a specific value will assume exactly this value, the other variables will assume the start standard value.

Reset Cold

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

Reset Origin

This command removes all the application tasks variables, including the persistent type variables and deletes the CPU application.

Notes:

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

Command: In order to execute a command from any type of Reset, is necessary to be in Online mode on CPU.

6. Redundancy with NX3030 CPU

Introduction

This chapter describes the Nexto Series CPUs redundancy which can only be used with the NX3030 CPU.

Nexto' s redundancy is of the hot-standby type, thus, the controllers are doubled. One controller is normally in active state and controlling a process, while the other is normally in stand-by state, keeping the synchronism with the active controller. In case of a failure in the active controller damaging its process control, the stand-by controller switches automatically to Active, within a very short time, in order not to disturb the process and cause any discontinuities in its outputs.

The hot-standby redundancy is a method used to increase failure tolerance and, consequently, increase the availability of automation systems. The basic idea is to ensure that no simple failure in duplicated components causes the process control interruption.

The hot-standby redundancy is applied on:

- Oil exploration platforms;
- Energy generation and distribution plants;
- Security interlock (Instrumented Security Systems);
- Continuous processes such as chemical plants, oil refinery, paper production, etc.

In the Nexto Series CPUs hot-standby redundancy, as it has already been described, the controllers are doubled. Besides, the field buses (PROFIBUS-DP) can also, optionally, be doubled, as well as the Ethernet supervisory networks and the Ethernet HSDN (High Speed Deterministic Network) control networks. By choosing the networks duplication, the availability becomes even higher.

The Nexto Series CPUs hot-standby redundancy is not applied to I/O modules. In case the I/O module redundancy is desired, it can be treated by the user in the application level. For instance, the user can duplicate or even triplicate an analog input module and create a vote scheme to define which input will be considered in an application specific time.

Figure 6-1 shows a typical example of redundant architecture using the NX3030 CPU.

The redundant CPU central part is formed by two identical racks, called PLCA and PLCB, and a redundancy control panel (PX2612). In the redundancy context, each rack (PLCA or PLCB) is called half-cluster, while the group formed by these two racks is called cluster.

In this example, a PROFIBUS network, Ethernet supervision network and Ethernet HSDN control network duplication can also be observed.

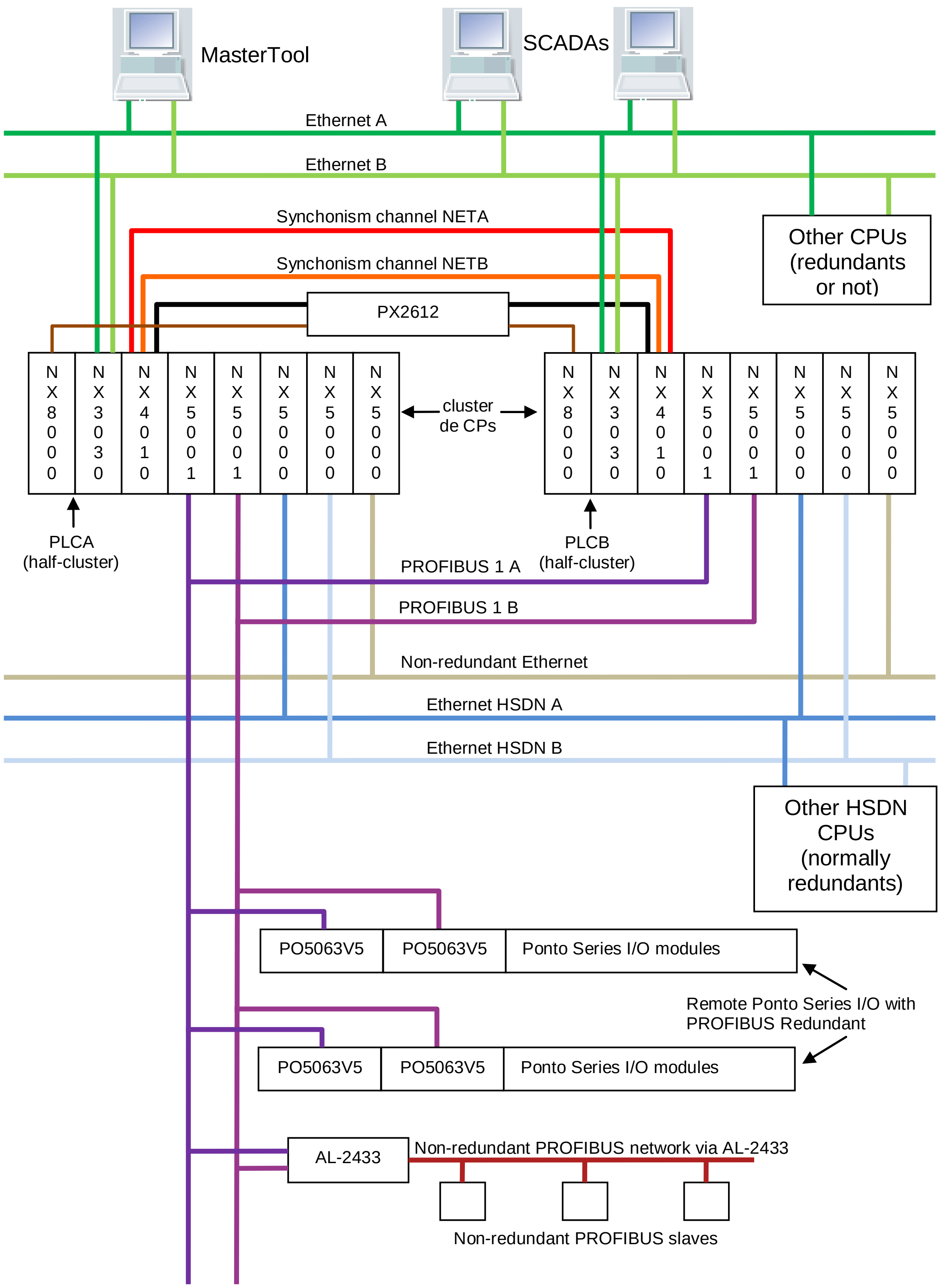


Figure 6-1. Example of redundant architecture with NX3030 CPU

Technical Description and Configuration

Minimum Configuration of a Redundant CPU (Not Using PX2612 Panel)

A redundant CPU is composed, at least, by:

- Two identical half-clusters

Each half-cluster consists of at least the following modules:

- The rack itself where the modules are inserted, which can be one of the following:
  - NX9000, with 8 positions
  - NX9001, with 12 positions
  - NX9002, with 16 positions
  - NX9003, with 24 positions
- The power supply NX8000, at rack positions 0 and 1;
- The NX3030 CPU, at rack positions 2 and 3;
- The module NX4010, at rack positions 4 and 5.

Figure 6-2 shows an example of a redundant CPU minimum configuration, using the smallest rack (NX9001, with 12 positions). In this case, it can be observer that the three modules inserted in the rack have double width (occupy two rack positions).

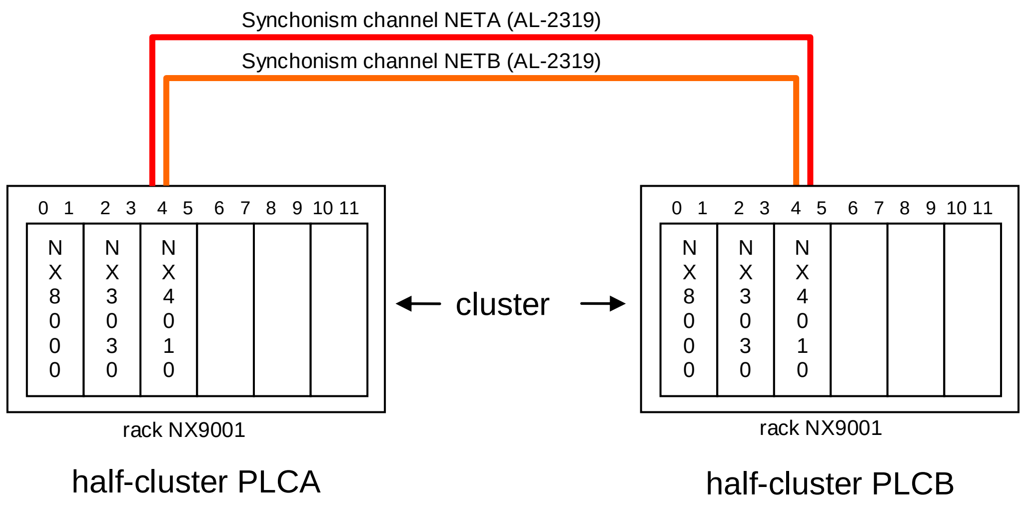


Figure 6-2. Minimum configuration of a redundant CPU in rack NX9001

Typical Configurations of a Redundant CPU

A minimum configuration, as the one shown on Figure 6-2, may already work as "redundant processing unit". It would be possible to use the serial and Ethernet communication channels from NX3030 CPU, for instance, for MODBUS TCP communication with a SCADA system, and MODBUS RTU and/or MODBUS TCP communication with smart field devices or MODBUS remote I/O systems.

In typical configurations, however, additional modules are inserted in the PLCA and PLCB half-clusters, for instance, to deliver a remote PROFIBUS I/O and Ethernet additional channels.

Among the additional modules which, optionally, may be inserted in the half-clusters, are the following:

- PROFIBUS Masters NX5001

- Ethernet Interfaces NX5000

In case is necessary, bigger racks can be used, as the NX9002 (16 positions) and NX9003 (24 positions). It must be observed that all the listed modules, so far in this chapter, have double width (occupy two positions).

Additionally, it’s also possible to use the PX2612 panel, which allows the execution of some redundant state machine transitions that, otherwise, wouldn’t be possible, in addition to the automatic half-clusters shutdown in failure situations.

NX5001 Modules Addition for PROFIBUS Networks

A redundant PLC is up to until four NX5001 modules for PROFIBUS networks usage. Each network can be single or redundant. In case the PROFIBUS “n” (being “n” a number between 1 and 4) be redundant, the two networks that belongs to this are named PROFIBUS “n” A and PROFIBUS “n” B. In case the PROFIBUS “n” be single, the network that belongs to it will be named PROFIBUS “n” A.

To create a redundant PROFIBUS network, must be inserted two NX5001 modules in each half-cluster. To create a simple PROFIBUS network, simply insert a NX5001 module in each half-cluster. Thus, it can be configured up to four simple networks, or two redundant networks, or a redundant and two simple. In other cases, fewer than four NX5001 modules will be needed in each half-cluster. More information about PROFIBUS networks is provided in the PROFIBUS Network Configuration section.

In Figure 6-1, there is only one PROFIBUS network (PROFIBUS 1), and the same is redundant (PROFIBUS 1 A and 1 B PROFIBUS). In this example, therefore, were inserted two NX5001 modules in each half-cluster.

NX5000 Modules Addition for Ethernet Networks

It’s possible to insert up to six NX5000 modules in each half-cluster, delivering six additional Ethernet channels, besides the two Ethernet channels already existent in the NX3030 CPU.

The Ethernet channels can be used in an individual way, or organized in NIC Teaming pairs, which are used to deliver redundant Ethernet channels, and are described, with more details, in the Redundant Ethernet Networks with NIC Teaming section.

A typical application for the NX5000 module can be the construction of a redundant HSDN (High Speed Deterministic Network), for the communication between several redundant CPUs. Through this network, many redundant CPUs can exchange messages in a totally segregated network, in order to guarantee determinism and a fast communication. Furthermore, configuring this network as redundant with NIC Teaming pairs, an excellent availability may be reached. In order to build such network (redundant HSDN), two NX5000 modules must be inserted in each half-cluster. Figure 6-1 shows a redundant HSDN example using two NX5000 modules in each half-cluster.

Applications where input and output modules are connected to Ethernet networks may require extra interface modules NX5000 to connect to these networks. In these cases, the network that connects the modules of inputs and outputs can be a simple or redundant network. Furthermore, the interfaces can be configured with the option of generating life failure. In this case, a network failure will cause a switch-over.

Figure 6-1 also shows an example with a NX5000 module used in the isolated form (without NIC Teaming redundancy), inserted at the right side from the other modules in each rack.

NX4010 Module

The NX4010 model, as shows Figure 6-3, was conceived in order to provide the interconnection between the two PLCA and PLCB half-clusters, and also to connect these half-clusters to the redundancy control panel PX2612. For further information regarding this module connections, see Interconnections between Half-Clusters and the Redundancy Control Panel PX2612.

Figure 6-3. NX4010 Module

NX4010 Features

Its main features are:

- Data and application synchronization between two half-clusters;
- Redundant communication interface between two half-clusters;
- Automatic switchover (active half-cluster change) in case of NX4010 and CPU communication time out;
- Possibility to switch off the other half-cluster;
- One Touch Diag TM;
- Electronic Tag on Display;
- Display and LEDs for diagnostics indication

Other features (generals, electrical, mechanic and environment) are presented in the NX4010 Redundancy Module Technical Features - CE114900.

Redundancy Control Panel PX2612

The PX2612 control panel is an optional item in a redundant system. It must be used when the ‘redundancy with panel’ option is selected during the project creation using the wizard. Figure 6-4 shows the redundancy control panel, while Figure 6-5 shows the frontal panel with details.

Through the DB9 connector called CONTROL PLC A, the connection with the CONTROL connector from PLCA NX4010 is made, using the AL-2317/A cable.

Through the DB9 connector called CONTROL PLC B, the connection with the CONTROL connector from PLCB NX4010 is made, using the AL-2317/B cable.

Furthermore, there’s a connector with 5 male terminals:

- GND: terminal for ground connection;

- RL A: 2 terminals connected to a relay NO (normally open) contacts, which can be commanded by PLCB to switch off PLCA. This relay must be closed by PLCB in order to switch off PLCA;
- RL B: 2 terminals connected to a relay NO (normally open) contacts, which can be commanded by PLCA to switch off PLCB. This relay must be closed by PLCA in order to switch off PLCB.

A CPU (PLCA or PLCB) can turn off the other CPU (PLCB or PLCA) in some exceptional situations, using the NO relays in the RLA and RLB connectors. Such situations are described in the Transition between Redundancy States section

The PX2612 has also 6 buttons for redundancy command and 6 LEDs used for redundancy state indication. Each CPU reads 3 from these 6 buttons and controls 3 LEDs.

For further information regarding these buttons and LEDs functions, see PX2612 Redundancy Command Panel Functions section.

Figure 6-4. Redundancy Control Panel PX2612

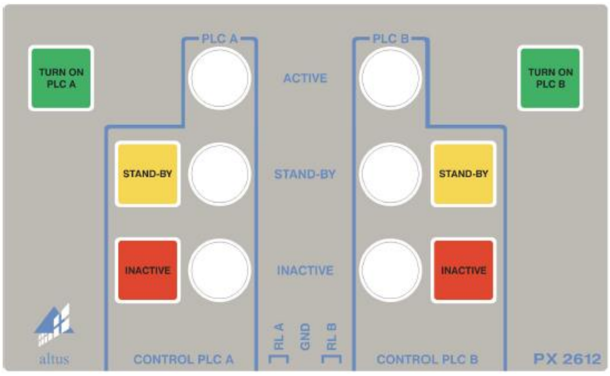


Figure 6-5. Redundancy Control Panel PX2612 Frontal View

PX2612 Features

The redundancy control panel PX2612 has the following features:

- CONTROL PLC A: connection to the module NX4010 from PLCA
- CONTROL PLC B: connection to the module NX4010 from PLCB
- RL A: relay NA terminals used to switch off PLCA
- RL B: relay NA terminals used to switch off PLCB
- GND: grounding

Other features (generals, electrical, mechanic and environment) are presented in the Redundancy Control Panel PX2612 Technical Features - CT112500.

Interconnections between Half-Clusters and the Redundancy Control Panel PX2612

Figure 6-6 shows how the connections between PLCA, PLCB and PX2612 have to be made, including the possibility to allow a CPU to switch off the other, which is necessary in exceptional situations.

Two AL-2319 cables must be used for the synchronism and redundancy channels NETA and NETB. One of these two cables interconnects the NX4010 NET 1 connector from each CPU (NETA channel). The other cable interconnects the NX4010 NET 2 connector from each CPU (NETB channel).

An AL-2317/A cable interconnects the NX4010 CONTROL connector from the PLCA to the PX2612 CONTROL PLC A.

An AL-2317/B cable interconnects the NX4010 CONTROL connector from the PLCB to the PX2612 CONTROL PLC B.

Besides this, it’s necessary to build a special power supply circuit in order to allow a CPU to switch off the other in extreme cases.

For higher reliability, two separate 24V power supplies must be used, one for PLCA supply and other for PLCB supply.

It can be observed that is necessary to use two external relays from the normally closed type (NC), with current capacity to feed the NX8000. These relays must be dimensioned for a nominal current around 2 A, however, a current inrush of around 10 A must be taken into account. Shunt diodes connected to the NC relays solenoids must be used to protect the PX2612 NO relay contacts

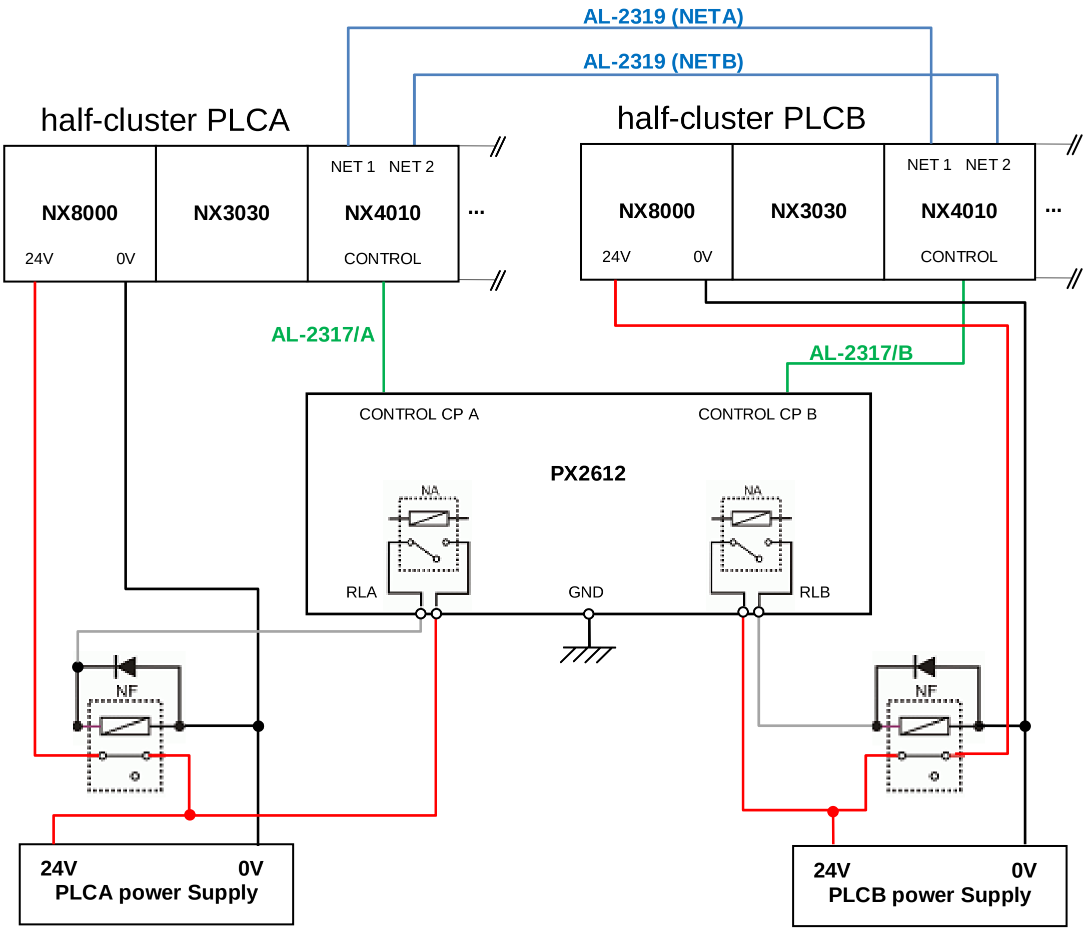


Figure 6-6. Interconnections between PLCA, PLCB and PX2612

General Features

	Redundant CPU General Features
Allowed CPUs	NX3030
Redundancy types	Hot-standby
Failure tolerances	Tolerates, at least, simple failures in doubled equipment in the half-clusters. In specific cases, it can tolerate multiple failures.
Half-cluster 5 redundancy states	- Not-configured: initial state, also considered when the CPU is off or isn't executing the MainTask; - Starting: temporary state assumed after Not-configured, where some tests will define the next state (Inactive, Active, Stand-by or back to Not-configured); - Inactive: state reached after some types of failures or for programming maintenance; - Active: controlling the user process; - Stand-by: ready to switch to Active and control the user process, in case there's such demand (e.g. Active CPU failure).
Main failures which cause switchover between the Active CPU and the Reserve CPU. **The reserve CPU switches for the Active and the Active can go to Inactive or Not-configured.**	- Supplying failure; - Power supply; - CPU (stop in the MainTask execution); - NX4010; - Failure in both synchronism channels (NETA and NETB) and the cause isn't in the Reserve CPU. In this case the Reserve CPU, besides assuming the Active state, switches the other CPU off; - Failure of some synchronism channel (NETA and NETB) and the cause is in the Active CPU; - Failure in some vital PROFIBUS network.
Commands that cause switchover between the Active CPU and the Reserve CPU.	- Commands via redundancy control panel (PX2612); - Commands received from MasterTool or from a SCADA system, through this CPU (local) or the other CPU (remote); - Commands generated by user application (e.g.: in case of other diagnostics as Ethernet communication failure) through this CPU (local) or the other CPU (remote).
Main failures which prevents a CPU to go to the reserve state or remain in it. **Such failures drive the CPU to a Not–Configured or Inactive state.**	- Supplying failure; - Power supply; - CPU (stop in the MainTask execution); - NX4010; - Failure in both synchronism channels (NETA and NETB) and the cause is in the Reserve CPU. - Failure in the synchronism service for redundancy data; - Failure in the synchronism service for the redundant forcing list; - Total failure in some vital PROFIBUS network; - Different project from the Active CPU, with project automatic synchronization enabled; - Firmware version incompatible with the Active CPU.
Commands that drive the CPU out of the reserve state	- Commands via redundancy control panel (PX2612); - Commands received from MasterTool or from a SCADA system, through this CPU (local) or the other CPU (remote). - Commands generated by user application (e.g.: in case of other diagnostics as Ethernet communication failure) through this CPU (local) or the other CPU (remote).
Switchover time	- Up to 3 cycles from the MainTask, depending on the stimulus for state change (command or failure); - In case of PROFIBUS network failure, 2 MainTask cycles + 500 ms
No discontinuities switchover (bump-less)	- A switchover doesn't cause discontinuities in the controller outputs, nor in the inner variables.
Redundancy overhead (MainTask cycle CPU consuming increased by redundancy).	- Maximum value automatically calculated by MasterTool and informed to the user, considering an empty redundant forcing list; - Typical average value of 60ms for 224kbytes of redundant data, in a system with a redundant PROFIBUS network and two redundant Ethernet HSDN networks.
CPU display	- Among other diagnostics, shows the redundancy state (Active, Stand-by, Inactive, Not-configured and Starting) together with the CPU identification (PLCA or PLCB).

Redundancy Control Panel PX2612	- Through buttons, allows commands of switchover or redundancy states transition for maintenance; - LEDs signalize the redundancy state in each half-cluster; - NO relay allows a half-cluster to switch off the other in extreme situations. A button allows the other half-cluster reactivating; - Embedded resources for buttons, LEDs and relays tests.
Redundancy diagnostics	- Indicate failures in the PLCA and in the PLCB, independent of their states (Active or Inactive); - Prevent “obscure failures”; - Allow quick maintenance, essential for high availability.
Redundancy commands	- Allow the execution of the same PX2612 control panel actions, among other commands (e.g. switchover command). - Can be executed in the local CPU, or transmitted to the other CPU (remote) via synchronism channels NETA/NETB; - Can be received through MasterTool or a SCADA system; - Can be executed through user application.
Redundancy events	- Register diagnostics and redundancy commands changes, with timestamp, allowing an investigation of the switchover causes.
SNTP (Simple Network Time Protocol)	Allow the events to have a precise timestamp adjusted to the world hour. It also synchronizes the CPU real time clock for other applications.
Commands and diagnostics synchronization	Each MainTask cycle, PLCA and PLCB exchange diagnostics and commands through synchronism channels NETA or NETB. This way, a CPU knows the other diagnostics and commands.
Redundant data synchronization	Each MainTask cycle, the Active CPU copies redundant data to the Inactive CPU through the synchronism channels NETA and NETB. Non-redundant data are not synchronized.
Redundant forcing list synchronization	Each MainTask cycle, the Active CPU copies the redundant forcing list to the Inactive CPU through the synchronism channels NETA and NETB. This list includes only forced redundant variables, this way PLCA and PLCB can have different non-redundant data groups forced, as these variables are not synchronized.
Single project for PLCA and PLCB	There’s a single project for the PLCA and PLCB, generated by MasterTool. The project is composed by the applicative project (executable code) and the archive project (source code).
CPU identification	Through MasterTool, a NX3030 CPU identifies itself as PLCA, PLCB or non-redundant CPU. This identification isn’t part of the applicative project generated by MasterTool, even though is written in a CPU using MasterTool. The CPU identification allows the feature of a single project for PLCA and PLCB.
Automatic project synchronization	If the Active CPU project becomes different from the Inactive CPU, it is copied from the first to the second. This synchronization can take several MainTask cycles. One must remember the project is composed by the applicative project (executable code) and the archive project (source code), and both are synchronized. This synchronization can be disabled in special cases in order to allow visualization of project modifications which can only be downloaded off-line in non-redundant CPUs.
On-line expansion of modules and PROFIBUS remotes	There are project modifications that can’t be done on-line in a non-redundant CPU, such as the inclusion of new modules or PROFIBUS remotes. However, using the CPU and the PROFIBUS network redundancy, it was defined a procedure to accomplish this goal, very important for systems which need high availability.
Private IP addresses for PLCA and PLCB	It’s possible to connect to a specific CPU (PLCA or PLCB) using a private IP address, to obtain half-cluster specific diagnostics, for instance. The PLCA IP address will always be associated to the PLCA NET(i) interface, while the PLCB IP address will always be associated to the PLCB NET(i) interface..
Active IP	Name of a strategy that allows the Ethernet client connect to a server from the redundant CPU using always the same IP address. This prevents the necessity of complex scripts to change the IP address when switchovers occur due to redundancy. The Active IP address will always be associated to the NET(i) interface from the Active CPU.
NIC Teaming	Name of the strategy which allows two Ethernet interfaces from a half-cluster to form a redundant pair sharing a same IP address.

	This way, redundant Ethernet network can be built easily, without the need for the clients, connected to a NIC Teaming, to implement complex scripts to switch IP addresses.
PROFIBUS Network and Vital Failures Configuration	The CPU supports 2 PROFIBUS networks, each one may be redundant or not. It's also possible to configure if each PROFIBUS network failure is considered vital (causes switchover) or not.
Single and cyclic user task	Only one user task is allowed, called MainTask. This task is cyclic.
Main POU programs: NonSkippedPrg and ActivePrg	At a redundant project creation, MasterTool generates automatically two empty POU programs, which must be filled by the user: - NonSkippedPrg: this POU is executed in both CPUs (PLCA and PLCB), independent on the redundancy state (Active or Inactive). It's used for diagnostics and special commands management. - ActivePrg: this POU is executed only in the Active CPU and is used for the final user's process control.

Table 6-1. General features of a redundant CPU

Purchase Data

The minimum configuration for a redundant CPU implies on the purchase of the following modules:

- Two racks, which must be chosen between the three available models according to the modules to be installed:
  - NX9000: 8 positions (4 double modules)
  - NX9001: 12 positions (6 double modules)
  - NX9002: 16 positions (8 double modules)
  - NX9003: 24 positions (12 double modules)
- Two NX8000
- Two NX3030
- Two NX4010
- Two AL-2319

Furthermore, it may be necessary to purchase the following additional modules:

- One PX2612
- One AL-2317/A
- One AL-2317/B
- Two modules NX5001 for each simple PROFIBUS network.
- Four modules NX5001 for each redundant PROFIBUS network.
- Two modules NX5000 for each additional simple Ethernet network.
- Four modules NX5000 for each additional redundant Ethernet network (NIC Teaming).

ATTENTION:
It can be installed up to 4 PROFIBUS modules in each half-cluster. This means that we can configure up to 4 simple PROFIBUS networks or up to 2 redundant PROFIBUS networks.

Principles of Operation

In this section, the redundant CPU functions using a NX3030 CPU is described, along with its behavior and states. It's also presented concepts and programming and configuration restrictions that will be used in the next chapters.

NX3030 CPU Identification

A NX3030 CPU has a nonvolatile identification register where it's possible for it to be identified as:

- Non-redundant: it can't be used in a redundant CPU (default configuration)
- PLCA: used in the redundant CPU PLCA
- PLCB: used in the redundant CPU PLCB

The identification register can be adjusted using the MasterTool programmer. The first thing to be done in a NX3030 CPU, before downloading the redundant project in it, is to identify it as PLCA or PLCB. In case the identification isn't executed, several redundancy features won't work correctly, as, for instance, the synchronization of the application between the PLCs.

ATTENTION: The CPU identification register is not part of the redundant CPU project, thus it isn't saved as part of this project in the computer where MasterTool is being executed. The register is saved only in the nonvolatile CPU memory.

Single Redundant Project

Due to the identification register previously described, there's a single project for the redundant CPU, identical for both PLCA and PLCB.

Configuration parameters that must be different for PLCA and PLCB (e.g. Ethernet interface IP address) appear doubled in the redundant CPU project (one for the PLCA and another for the PLCB). Each CPU will consider the correspondent one, after analyzing its identification register.

Redundant Project Structure

Redundancy Template

A redundant CPU project is created automatically from a model, called Redundancy Template.

The template starts from the minimum redundant CPU configuration, as defined in the Minimum Configuration of a Redundant CPU section. Besides this, some dialogs with the user are made for the insertion of additional modules in the half-clusters, such as PROFIBUS masters (NX5001) and Ethernet modules (NX5000).

PROFIBUS remotes must be inserted by the user, below the NX5001 PROFIBUS masters already inserted.

Furthermore, tasks and basic POUs from the program type are created, as described in the following sections.

Single and Cyclic Task MainTask

The redundant CPU project has a single task, called MainTask, which is cyclic. The user can adjust the task cycle time.

MainPrg Program

The MainTask is connected to a single POU from the program type, called MainPrg. The MainPrg program is created automatically.

The MainPrg code is the following, in ST language:

```
fbRedundancyManagement();
NonSkippedPrg();
IF fbRedundancyManagement.m_fbDiagnosticsLocal.eRedState =
REDUNDANCY_STATE.ACTIVE THEN
   ActivePrg();
END_IF
```

MainPrg call two POUs from the program type, called NonSkippedPrg and ActivePrg. NonSkippedPrg is always called, as it's executed in both CPUs. On the other hand, ActivePrg is only called when the "RedDgnLoc.RedState = Active" condition is true, in other words, when the CPU is in active state.

However, the NonSkippedPrg program is executed in both CPUs (PLCA and PLCB) independent on the redundancy state of this CPU. On the other hand, the ActivePrg is executed only in the active CPU.

Opposite to the MainPrg, which must not be modified, the user may modify the NonSkippedPrg and ActivePrg programs. Initially, when the redundant project is created from the Redundancy Template, these two programs are created "empty", but after that the user may insert his code.

ATTENTION: When the OPC option is enabled when creating the project, the NonSkippedPrg program is not created empty. For more information, refer to the OPC Usage section in Redundant Projects.

ActivePrg Program

The main goal of this program, which is executed only in the active CPU, is to control the final user process.

This program normally acts on the redundant variables, among which the direct representation variables are found %I and %Q associated to the remote I/O system. For further information see the chapter Redundant CPU Programming, MainTask Configuration - ActivePrg Program.

ATTENTION The compilation being successful or not, MasterTool informs the calculated looseness and the redundancy overhead predicted on the message window.

NonSkippedPrg Program

This program is executed in both CPUs (PLCA and PLCB) independent on the redundancy state. It's typically used for functions such as:

- To organize non-redundant diagnostics to report to a SCADA system
- To receive and execute non-redundant commands from a SCADA system
- To manage switchover conditions normally not automatically contemplated by the redundant CPU, that can vary from user to user. E.g. a user will be able to execute a switch over to the Reserve CPU if the Active CPU isn't communicating with the SCADA system, while another user may not want a switchover on this situation
- To enable or disable I/O drivers according to the redundancy state, e.g. disable a Modbus RS-485 master in the Inactive CPU
- To detect failures in I/O drivers in an inactive CPU, in order to avoid obscure failures. Some I/O drivers don't include such failures automatically detection, while others, such as the PROFIBUS, does it automatically
- Other activities which, for some reason, need to be executed either in the Active CPU and the Reserve CPU

For further information see MainTask Configuration - NonSkippedPrg Program chapter.

Redundant and Non-redundant Variables

The redundant CPU variables can be classified among redundant and non-redundant. Redundant variables are copied from the Active CPU to the Inactive CPU, at the MainTask beginning of each cycle, through the synchronism channels NETA and NETB. On the other hand, non-redundant variables aren't copied between half-clusters, thus can have different values in PLCA and PLCB.

The non-redundant variables are used to store private information of each half-cluster (PLCA or PLCB), such as module diagnostics inside the half-cluster, including the redundancy diagnostics (half-cluster diagnostics state, etc.).

The redundant variables regard the shared information connected to the process control. The variables associated to the I/O modules are typical examples of redundant variables.

Redundant and Non-redundant %I Variables

The NX3030 CPU allocates 96 kbytes of %I variables (%IB0 ... %IB98303).

The first 82 kbytes can be redundant (%IB0 ... %IB81919). The last 16 kbytes are always non-redundant (%IB81920 ... %IB98303).

The 80 kbytes area which can be redundant is allocated for inputs, which can be read from an I/O remote module (PROFIBUS, Modbus, etc.).

The 16 kbytes non-redundant area is allocated for a half-cluster "quick private diagnostics", and also for the redundancy command panel PX2612 buttons. Quick diagnostics are the ones that must be updated each MainTask cycle.

The user may configure the redundant %I variables quantity, between 0 and 81920 Kbytes, in 1kbyte multiples (o valor default é 16384 bytes - %IB0 ... %IB16383). The proper configuration of redundant %I from %IB0 is important to decrease the necessary time for redundant variables synchronization (decrease the redundancy overhead). E.g. if the real application allocates only %IB0 ... %IB1499 for redundant inputs, the redundant %I area size can be defined as 1500 bytes

The figure below illustrates the redundant and non-redundant %I direct representation variables allocation, where RI is the %I quantity really configured as redundant

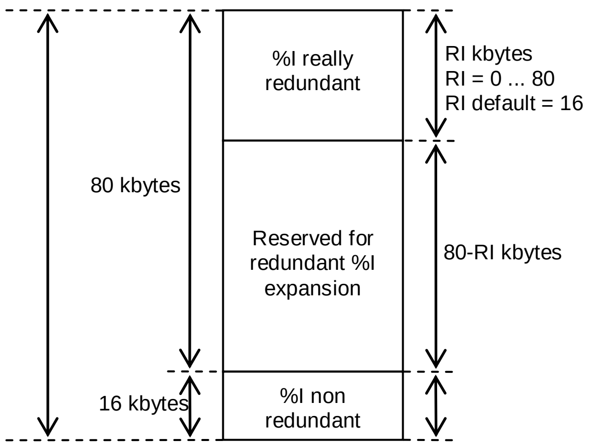


Figure 6-7. Redundant and Non-redundant %I Allocation

Redundant and Non-redundant %Q Variables

The NX3030 CPU allocates 96kbytes of %Q variables (%QB0 ... %QB98303).

The first 80 Kbytes can be redundant (%QB0 ... %QB81919). The last 16kbytes are always non-redundant (%QB81920 ... %QB98303).

The 80 Kbytes area which can be redundant is divided in two sections:

- The first Kbytes are reserved for outputs that can be redundant, and are typically allocated for an I/O remote system (PROFIBUS, Modbus, etc.). The size value is configurable and its default value is 16384. By default, this section includes %QB0 ... %QB65535.
- The next bytes are reserved for diagnostics which can be redundant, from the I/O system (I/O modules diagnostics, communication interfaces diagnostics, PROFIBUS slaves diagnostics, etc.), for instance. Different from the quick diagnostics (allocated in %I), such diagnostics allocated in %Q can take more than one MainTask cycle to be updated. By default this section includes 16 Kbytes (%QB65536 ... %QB81919).

The non-redundant area (%QB94208 ... %QB98303) is typically allocated for diagnostics and private commands of a half-cluster, and also for the redundancy command panel PX2612 LEDs and relay.

The user can reduce the redundant %Q variable quantity in each one of the sections which can be redundant…

- On the first section, the really redundant area size can be configured between 0kbytes and 65535 bytes, in 1kbyte multiples (the default value is 16384 bytes). The proper configuration of redundant %Q is important to decrease the necessary time for redundant variables synchronization (decrease the redundancy overhead). E.g. if the real application allocates only %Q0 ... %Q1499 for redundant outputs, the redundant %Q area size can be defined as 1500 bytes.
- On the second section, the really redundant area size can be configured between 0kbytes and 81919 bytes, in 1kbyte multiples (the default value is 16384 bytes). The proper configuration of redundant %Q is important to decrease the necessary time for redundant variables synchronization (decrease the redundancy overhead). E.g. if the real application allocates only %QB65536 ... %QB66999 for redundant diagnostics, the redundant %Q area size can be defined as 1464 bytes.

The figure below illustrates the redundant and non-redundant %Q direct representation variables allocation, where RQS is the %Q output quantity configured as redundant in the first section, and RQD is the %Q diagnostics quantity configured as redundant in the second section.

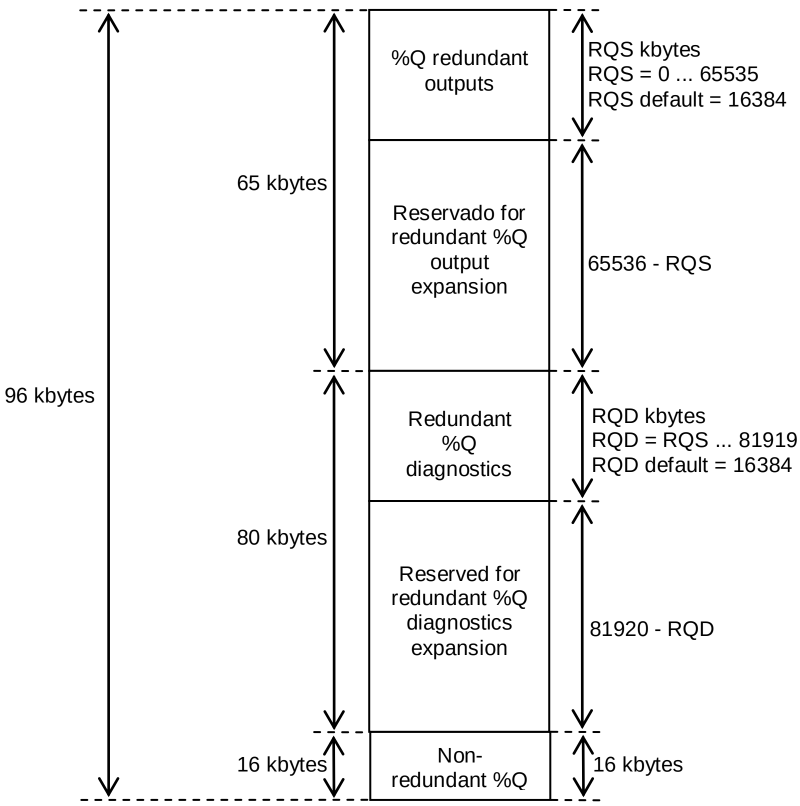


Figure 6-8. Redundant and Non-redundant %Q Allocation

Redundant and Non-redundant %M Variables

The NX3030 CPU allocates 64kbytes of %M variables (%MB0 ... %MB65535).

All the 65535 bytes can be redundant (%MB0000 ... %MB65535). By default the redundant %M variables quantity is 0.

The %M variable use must be avoided and the use of symbolic variables preferred (see Redundant and Non-redundant Symbolic Variables section).

Redundant and Non-redundant Symbolic Variables

Besides the direct representation variables (%I, %Q and %M) which are allocated automatically. The user can explicitly declare symbolic variables, inside of POUs or GVLs. The maximum size allowed for redundant symbolic variables allocation is 512kbytes.

ATTENTION: Symbolic variables must not be confused with AT variables. The AT variables are mere symbolic names attributed to direct representation variables (%I, %Q and %M), using the “AT” declaration. Thus, AT variables don’t allocate any symbolic variables memory.

Symbolic variables are redundant in the following cases:

- When declared in POUs from the “program” type created in the user application, exceptionally the NonSkippedPrg program.
- When declared in GVLs created in the user application and these GVLs marked as redundant.

Symbolic variables aren’t redundant in the following cases:

- When declared in the NonSkippedPrg program. This program has been described previously in the NonSkippedPrg section.
- When declared in POUs from the “function” type. It can be observed this POUs normally must allocate variables only on the pile (non static), which consequently don’t need to be redundant. If the user declares static variables (VAR STATIC) inside the POUs from the “function” type, this will be considered bad programming. Such static variables, in case they are created, will be considered non-redundant.
- When declared in POUs from the “function block” type. It can be observed the mere “function block” declaration doesn’t allocate memory (what allocates memory is to turn a function block into as instance).

It must be observed that the function blocks instances, declared inside POUs from the program type or inside GVLs, behave as symbolic variables, in other words, allocate redundant memory. In the same way symbolic variables, when function block instances, are declared in the NonSkippedPrg program or when the GVL isn’t marked as redundant, such instances are non-redundant

Multiple Mapping

If the user desires to map the redundant command variables in more than one communication port (COMx or NETx) it’s necessary the implementation of a control by the user within his own application.

The control logic to be implemented must write in the redundant command variables based on the variables (commands) values from each communication port (COMx or NETx). Besides that, the control logic must restart the communication ports command variables, as the redundancy control just restarts its own command variables.

The following is an example of this implementation:

```
VAR
    var_StandBy_command_Ethernet_relation : BOOL;
    var_StandBy_command_Serial_relation : BOOL;
    var_Inactive_command_Ethernet_relation : BOOL;
    var_Inactive_command_Serial_relation : BOOL;
    var_TurnOn_command_Ethernet_relation : BOOL;
    var_Turn_command_Serial_relation : BOOL;
END_VAR

//Logic to put the local PLC in StandBy
IF var_StandBy_command_Ethernet_relation = TRUE THEN
    DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
    var_StandBy_command_Ethernet_relation:=FALSE;
END_IF
IF var_StandBy_command_Serial_relation = TRUE THEN
    DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal:=TRUE;
    var_StandBy_command_Serial_relation:=FALSE;
END_IF
// Logic to put the local PLC in Inactive
IF var_Inactive_command_Ethernet_relation = TRUE THEN
    DG_NX4010.tRedundancy.RedCmdLoc.bInactiveLocal:= TRUE;
    var_Inactive_command_Ethernet_relation:=FALSE;
END_IF
IF var_Inactive_command_Serial_relation = TRUE THEN
    DG_NX4010.tRedundancy.RedCmdLoc.bInactiveLocal:=TRUE;
    var_Inactive_command_Serial_relation:=FALSE;
END_IF
//Logic to switch on the local PLC switched off by the PX2612
IF var_TurnOn_command_Ethernet_relation = TRUE THEN
```

```
    DG_NX4010.tRedundancy.RedCmdLoc.bTurnOnLocal:= TRUE;
    var_TurnOn_command_Ethernet_relation:=FALSE;
END_IF
IF var_Turn_command_Serial_relation = TRUE THEN
    DG_NX4010.tRedundancy.RedCmdLoc.bTurnOnLocal:=TRUE;
    var_Turn_command_Serial_relation:=FALSE;
END_IF
```

Above there’s an example in ST language, where the redundancy command can be executed through two variables from different communication ports. On the same example, three different commands were executed (StandBy, Inactive and Turn-on).

Where:

var_StandBy_ command_Ethernet_relation: Bool type variable attributed to an Ethernet communication Coil which will execute the command to put the local Half-Cluster in Stand-By.

var_StandBy_command_Serial_relation: Bool type variable attributed to a Serial communication Coil which will execute the command to put the local Half-Cluster in Stand-By.

DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal: this command executes an action similar to the button STAND-BY from the PX2612, in the local PLC.

var_Inactive_command_Ethernet_relation: Bool type variable attributed to an Ethernet communication Coil which will execute the command to put the local Half-Cluster in Inactive.

var_Inactive_command_Serial_relation: Bool type variable attributed to a Serial communication Coil which will execute the command to put the local Half-Cluster in Inactive.

DG_NX4010.tRedundancy.RedCmdLoc.bInactiveLocal: this command executes an action similar to the button INACTIVE from the PX2612, in the local PLC.

var_TurnOn_command_Ethernet_relation: Bool type variable attributed to an Ethernet communication Coil which will execute the command to reactivate the local Half-Cluster after switched off by the PX2612 relay.

var_Turn_command_Serial_relation: Bool type variable attributed to a Serial communication Coil which will execute the command to reactivate the local Half-Cluster after switched off by the PX2612 relay.

DG_NX4010.tRedundancy.RedCmdLoc.bTurnOnLocal: this command executes an action similar to the button STAND-BY from the PX2612, in the local PLC.

Diagnostics, Commands and User Data Structure

Each CPU has several data structure related to redundancy. The following structure is AT variables mapped over %Q variables:

- RedDgnLoc: has diagnostics from the CPU (local) related to the redundancy, as the CPU redundancy state, for instance;
- RedDgnRem: it’s a copy from the other CPU RedDgnLoc, received through NETA/NETB synchronism channels. This way, this CPU (local) has access to the other CPU (remote) diagnostics;
- RedCmdLoc: has commands which must be applied on this CPU (when called Local) or on the other CPU (when called Remote). E.g. the StandbyLocal field from this data structure corresponds to a command which must be executed in this CPU (local), while the StandbyRemote field corresponds to a command which must be executed in the other CPU (remote);
- RedCmdRem: it’s a copy from the other CPU RedCmdLoc, received through NETA/NETB synchronism channels. This way, this CPU (local) can execute commands received from the other CPU (remote);

- RedUsrLoc: has 128bytes of data filled freely by the user (e.g. communication diagnostics with a SCADA system). These 128bytes of data can be interchanged with the other CPU (remote);
- RedUsrRem: it's a copy from the other CPU RedUsrLoc, received through NETA/NETB;

On Maintenance section, the following sub-sections offer more details regarding these data structures:

- Redundancy Diagnostics Structure
- Redundancy Commands
- User Information Exchanged between PLCA and PLCB

Cyclic Synchronization Services through NETA and NETB

This section describes the three synchronization services which occur cyclically in a redundant CPU between PLCA and PLCB, through NETA and NETB synchronism channels.

These services are executed at the beginning of each MainTask cycle, and in the sequence which they appear below:

- First, the Diagnostics Exchange and Commands service is executed;
- Second, the Redundant Data Synchronization service is executed;
- Third, the Redundant Forcing List Synchronization service is executed.

Diagnostics and Commands Exchange

This service is responsible by the interchange of the following data structures, in each MainTask cycle:

- To copy RedDgnLoc from PLCA to PLCB RedDgnRem;
- To copy RedCmdLoc from PLCA to PLCB RedCmdRem;
- To copy RedUsrLoc from PLCA to PLCB RedUsrRem;
- To copy RedDgnLoc from PLCB to PLCA RedDgnRem;
- To copy RedCmdLoc from PLCB to PLCA RedCmdRem;
- To copy RedUsrLoc from PLCB to PLCA RedUsrRem.

The service will be executed using only one synchronism channel (NETA or NETB). This way the service can be completed even if one channel has problems.

Redundant Data Synchronization

This service is responsible for the redundant variables transferring, from the Active CPU to the Inactive CPU. As previously described, there are symbolic redundant variables and also redundant direct representation variables (%I, %M and %Q).

For this service to be executed, several conditions must be satisfied:

- The previous synchronization service in this MainTask cycle (Diagnostics and Commands Exchange) must be completed with success;
- In case this CPU is in Active state, the other must be in Non-Active state. On the other hand, in case this CPU is in Non-Active state, the other must be in Active state;
- Both projects (2 CPUs) must be identical, except when the project automatic synchronization is disabled (see Project Synchronization Disabling section);
- At least one synchronism channel (NETA and/or NETB) must be operational. If both synchronism channels (NETA and NETB) are operational, the communication is distributed between both (load balances) in order to reduce the synchronization time. In case only one channel is operational, the synchronism will continue to be executed only by this channel, keeping the redundant data synchronization.

Redundant Forcing List Synchronization

This service is responsible for the redundant forcing list transferring, from the Active CPU to the Inactive CPU.

For this service to be executed, several conditions must be satisfied:

- Both synchronization services previous to this cycle (Diagnostics and Commands Exchange) must be completed with success;
- In case this CPU is in Active state, the other must be in Non-Active state. On the other hand, in case this CPU is in Non-Active state, the other must be in Active state;
- Both projects (2 CPUs) must be identical, except when the project automatic synchronization is disabled (see Project Synchronization Disabling section);
- At least one synchronism channel (NETA and/or NETB) must be operational. If both synchronism channels (NETA and NETB) are operational, the communication is distributed between both (load balances) in order to reduce the synchronization time. In case only one channel is operational, the synchronism will continue to be executed only by this channel, keeping the redundant data synchronization.

ATTENTION:
The redundant forcing list has only forcing over redundant variables. On each CPU (PLCA and PLCB), there can be a different forcing list related to non-redundant variables.

Sporadic Synchronization Services through NETA and NETB

The following synchronization services are executed sporadically, in other words, they are not executed in each MainTask cycle. Another system task executes these sporadic services in background.

Project Synchronization

This service is responsible for synchronizing the Active CPU and Non-Active CPU projects. This happens when the projects are different in both CPUs and the automatic projects synchronization is enabled on both CPUs.

The synchronization is always from the Active CPU to the Non-Active CPU and it’s enough that a one out of two synchronism channel (NETA or NETB) is operational for this service to be executed.

When the synchronization is enabled, the following files and services will be synchronized:

- Project application(executable code);
- Project archive (source code);
- User and groups;
- Access rights;
- Trace.

The synchronization service will start within thirty seconds after one of the CPUs goes to Active state, and after its beginning, the project CRC will be checked every five seconds.

When synchronization is started the Non-Active CPU goes to Stop mode, at the Not-Configured state. After the transferring of all necessary files, the Non-Active CPU goes to Run, at Starting state. In case the transfer fails, the CPU goes back to Not-Configured state.

The time the synchronization will take to be fully executed depends on the project size. In average, a transfer rate between the synchronism channels is approximately 500 Kbytes/s.

In case the synchronization is interrupted (communication loss between synchronism channels) during the files transferring from the Active CPU to the Non-Active CPU, the procedure is aborted and restarted when the communication is restored. Only after the conclusion of the whole procedure the Non-Active CPU goes to Run mode.

Besides keeping the projects synchronized, the Project Synchronization will also avoid the Non-Active CPU to assume superior states in relation to Starting in case the CRC is different or some Online Change is to be executed in the Active CPU.

ATTENTION: A project synchronization will have the same effect as a download in the Non-Active CPU. This service isn’t executed if the automatic Project Synchronization Disabling, as it’s described on Project section. No synchronization service between CPUA and CPUB works in case the synchronism channels cables are inverted. E.g. to connect the NETA channel in the NETB channel from CPUB and the NETB from the CPUA in the NETA in the CPUB.

ATTENTION: In the update from the version 1.20 to later versions of MasterTool IEC XE, was done a modification in the communication protocol between the synchronism channels. Therefore, is not possible to sync data between two PLCs when one of the applications has been created in a version prior to 1.21 and another application has been created in an equal or higher version. To be able to perform the synchronization, you should perform the actions described at section Not Loading the Application at Startup in the PLC with the oldest project. Doing this, the application will not be loaded, but, when this PLC goes to Non-Configured state during the system initialization, the applications will be synchronized automatically.

ATTENTION: Before version 2.01 of MasterTool IEC XE, when sending the source code to the active CPU, the Stand-by CPU went for Non-Configured state to sync it. However, to complete the synchronization operation, the CPU remained in the state Non-Configured, being necessary to pass the CPU to Stand-by status via STAND-BY button on the PX2612 or equivalent command. Starting with version 2.0.1 the CPU that is in Stand-by will change your state to Not-Configured during the synchronization process, but will return automatically when the sources are the same between the two Half-Clusters

Project Synchronization Disabling

On Sporadic Synchronization Services through NETA and NETB section, applicative project and archive project synchronization services were described. These services normally must be enabled, and are useful when the project modifications can be downloaded on-line in the Active CPU and the Stand-by CPU afterwards, automatically, through the synchronism channels NETA/NETB.

However, there are project modifications which can’t be downloaded on-line in any CPU, e.g. the inclusion of modules in a PROFIBUS remote, or the inclusion of a new PROFIBUS remote. In these cases, using the CPU and PROFIBUS network redundancy, such modifications can be made without interrupting the process control. A procedure to accomplish this objective is described in the Exploring the Redundancy for Off-Line downloading of Modifications without Interruption of the Process control section.

In this procedure it’s necessary to disable temporarily the project synchronizations, allowing, for a while, one CPU to operate with a project new version, while the other CPU still operates with the old project version.

A NX3030 CPU has a register for Project Synchronization Disabling, nonvolatile, which allows the disabling of the project application and project archive synchronization services. This register can be adjusted using MasterTool. It’s enough to disable the project synchronization in one of the two CPUs to guarantee it doesn’t work anymore.

To disable the Project synchronization, the user must, firstly, connect into desired PLC with the software MasterTool (see chapter MasterTool Connection with a NX3030 CPU from a Redundant PLC).

Next, in the Online / Cluster Basic Configuration menu, the combo-box “Project Synchronization” must be opened, allowing the selection of the two following options:

- Enable
- Disable

The option “Disable” must be selected and the combo-box correspondent “Write” button pressed. A message informs if the operation is successful or not.

The disabling configuration of project synchronism isn’t part of the redundant project developed in the MasterTool. Such configuration is only in a non-volatile memory area in the CPU, which can be read or written using MasterTool. MasterTool doesn’t save this configuration in any file.

This configuration is copied on each cycle of MainTask, from the non-volatile memory to the DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable. The user can verify this diagnostics in the PLC to see if the command succeeded, since the PLC is in Run mode (DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable must be 1). In case the PLC isn’t in Run mode, it’s possible to verify configuration straight on the NX3030 CPU display in the PLC (see Redundancy Diagnostics on the NX3030 CPU Graphic Display section).

The DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable diagnostic can also be observed also in the remote PCL through the DG_NX4010.tRedundancy.RedDgnRem.sGeneral_Diag.bProjectSyncDisable (since the Non-Active PLC is in Run mode). A PLC (Active or Non-Active) stops the project synchronization service every time any of the following bits are true:

- DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag. bProjectSyncDisable
  - This PLC, local bit. This PLC is with the project synchronization disabled.
- DG_NX4010.tRedundancy.RedDgnRem.sGeneral_Diag. bProjectSyncDisable
  - The other PLC, remote bit. The remote PLC is with the project synchronization disabled.

ATTENTION: The Project Synchronization Disabling register isn’t part from the redundant CPU project, thus it’s not saved as part of it in the computer where MasterTool is being executed. The register is saved only in the nonvolatile CPU memory.

PROFIBUS Network Configuration

It’s possible to install up to four PROFIBUS Master NX5001 modules in each half-cluster. So, we can define up to two redundant PROFIBUS networks, called PROFIBUS 1 and PROFIBUS 2, or up to four simple PROFIBUS networks, called PROFIBUS 1, PROFIBUS 2, PROFIBUS 3 and PROFIBUS 4, or even one redundant network and two simple ones, named PROFIBUS 1, PROFIBUS 2 and PROFIBUS 3.

PROFIBUS Redundancy

Each of the PROFIBUS networks can be redundant or not redundant. For example, if PROFIBUS 1 network is redundant, it will be divided into PROFIBUS 1 A and PROFIBUS 1 B. If it’s not redundant, there’s going to exist only PROFIBUS 1 A. The same applies to the PROFIBUS 2.

Figure 6-1 shows an example with a single PROFIBUS network (PROFIBUS 1), which is redundant (PROFIBUS 1 A and PROFIBUS 1 B).

Only a few remote types can be connected straight to this redundant PROFIBUS network:

- PO5063V5: PROFIBUS slave DP-V0 for Ponto Series remotes;

- PO5065: PROFIBUS slave DP-V1 with Hart, for Ponto Series remotes;
- AL-3416: PROFIBUS slave DP-V0 for AL-2004 CPU.
- NX5210: PROFIBUS slave DP-V0 for Nexto Series remotes.

Figure 6-1 also shows the possibility to connect non-redundant remotes to this type of redundant PROFIBUS network, through the AL-2433 module (ProfiSwitch). Such non-redundant PROFIBUS remotes can be from any brand or model.

PROFIBUS Failure Modes Vital and Not-Vital

Each one of the PROFIBUS networks can be configured in two different modes:

- Vital failure: in case this network fails completely, this failure can determine a redundancy state transition in the redundant CPU (switchover). In case a redundant PROFIBUS network, a complete failure implies in the failure of both composing networks.
- Not-Vital failure: even if this network fails completely, this failure won’t determine a redundancy state transition in the redundant CPU (switchover).

Redundant Ethernet Networks with NIC Teaming

Figure 6-1 shows two redundant Ethernet networks examples, with NIC Teaming.

In the first case, the NX3030 CPU connects to the supervision network (SCADA), also used for configuration through MasterTool. Both NX3030 CPU Ethernet ports (NET 1 and NET 2) form a NIC Teaming redundant pair, interconnected in two different switches (Ethernet A and Ethernet B). In some point, these two switches must be interconnected, for the two NIC Teaming ports connection and for an even higher availability (against double failures).

In the second case, two NX5000 modules also form a NIC Teaming redundant pair, interconnected in two different switches (Ethernet HSDN A and Ethernet HSDN B). In some point, these two switches must be interconnected, for the two NIC Teaming ports connection and for an even higher availability (against double failures).

Such Ethernet architectures turn possible an excellent availability, against Ethernet port failures, in cables and switches.

ATTENTION: If two modules, or Ethernet interfaces, form a NIC Teaming redundant pair, the configuration and device inclusion will be only possible in the first interface. The second interface will have his configuration parameters blocked for edition.

A cluster of two Ethernet ports forming a NIC Teaming pair has a single IP address, related to the port pair. This way, a client as SCADA or MasterTool, connected to a CPU server, doesn’t need to worry in IP address changing in case there’s a failure in any NIC Teaming pair port.

Each of the Ethernet interfaces that form the NIC Teaming pair have a unique diagnostics structure to point to failures which eventually might appear in any port of a NIC Teaming pair.

For further details regarding NIC Teaming configuration and diagnostics, see the following sections:

- Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2)
- NX5000 Modules Configuration

IP Change Methods

A redundant cluster from Nexto Series has four methods for IP change in the Ethernet ports of the NX5000 modules in each half-cluster and one method for IP change in the NET1 and NET2 ports of the NX3030 CPU. These methods define the ports’ behavior, regarding its IP, according to the current state of the half-cluster (Active or Non-Active) and with the half-cluster (PLCA or PLCB).

The methods are: Fixed IP, Exchange IP, Active IP and Multiple IP.

Overall, it can be listed up to four IPs, according to the IP change method.

Fixed IP

It's the simplest method for IP addressing and can be configured in the Ethernet interfaces in the NX5000 Ethernet modules. In this method, it's only listed the IP addresses from the PLCA and from PLCB. Apart from the redundancy state, PLC Active or Non-Active, the PLCA will always answer by the configured IP, as also will PLCB.

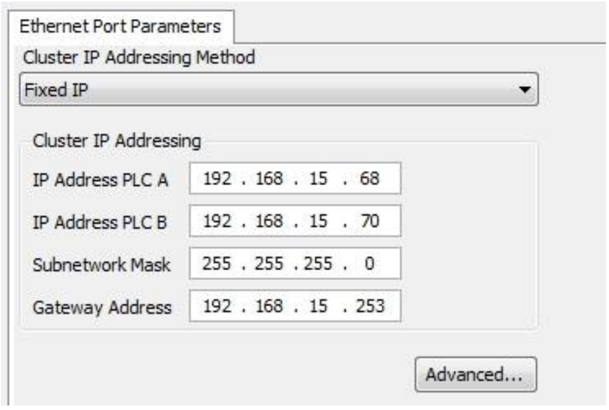


Figure 6-9. Fixed IP method

Parameters that must be configured in the Fixed IP method:

- IP Address PLC A: PLCA communication address
- IP Address PLC B: PLCB communication address
- Subnetwork Mask
- Gateway Address

Exchange IP

The Exchange IP can be configured in the Ethernet interfaces in the NX5000 Ethernet module. In this method, the half-cluster IP depends on the PLC state (Active or Non-Active). On every switchover the IP change occurs between the half-clusters allowing them to assume the IP address from the new redundancy state.

PS: for this addressing method, the Ethernet ports from both PLCs (PLCA and PLCB) assume the same IP address while they both are in the Non-Active state, generating a network address conflict. Considering this situation uncommon, where no PLC is controlling the system, this turns out to be a big problem and has to be considered.

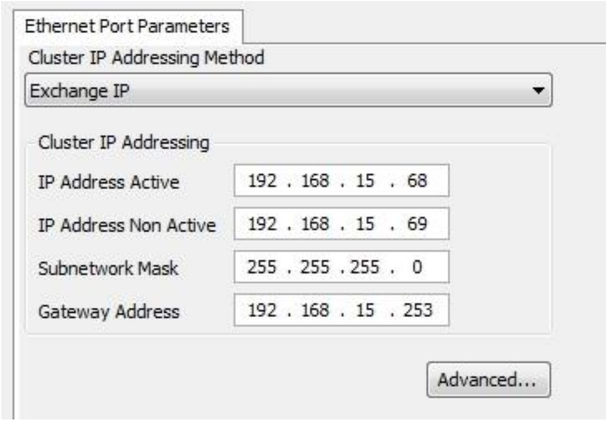


Figure 6-10. IP Automatic Change

Parameters that must be configured in the Exchange IP method:

- IP Address Active: PLCA communication address
- IP Address Non Active: PLCB communication address
- Subnetwork Mask
- Gateway Address

Active IP

This method is used in the redundant NX3030 CPU NETs and is also possible to be configured in the NX5000 modules. In this method there's an IP for the Active half-cluster and two more IPs, one for the PLCA and another for the PLCB. In the redundant NX3030 CPU NETs, the Active IP address is added to the interface of the Active PLC, and it can use either the Active IP address or the PLCX IP address in order to establish communication with the PLC. On the other hand, in the NX5000 Ethernet modules the Active IP address substitutes the Non-Active PLCX IP address, when the PLC is in Active mode.

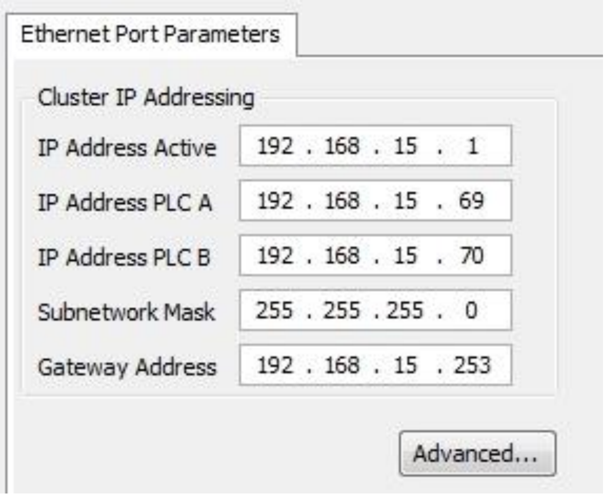


Figure 6-11. Active IP method – Redundant NX3030

Parameters that must be configured in the Active IP method for the NETs of a redundant NX3030 CPU:

- IP Address Active: IP address added to the interface when the PLC is in Active state;
- IP Address PLC A: PLCA communication address, apart from its current state;
- IP Address PLC B: PLCB communication address, apart from its current state;
- Subnetwork Mask;
- Gateway Address.

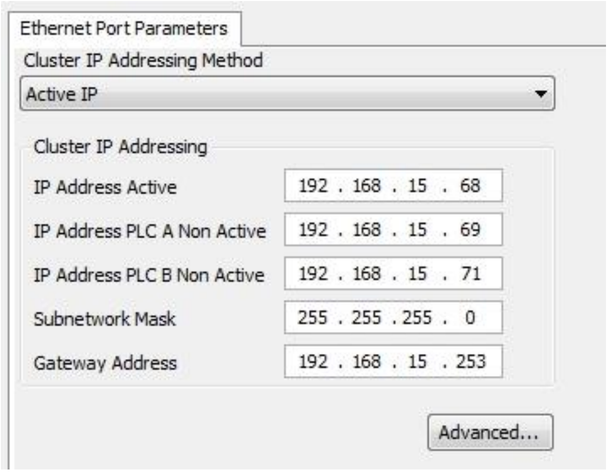


Figure 6-12. Active IP method – NX5000

Parameters that must be configured in the Active IP method for the NX5000 Ethernet modules:

- IP Address Active: Active PLC communication address. Replaces the IP address from the Non-Active PLCX;
- IP Address PLC A Non Active: PLCA communication address, when in Non-Active state;
- IP Address PLC B Non Active: PLCA communication address, when in Non-Active state;
- Subnetwork Mask;
- Gateway Address.

Multiple IP

The Multiple IP method can be configured in the Ethernet interfaces from the NX5000 Ethernet modules. In this method there's an IP for each half-cluster and for each state of the PLC. The PLCA assumes an IP address when it's Active and another when it's Non-Active. The same happens for the PLCB regarding its state (Active or Non-Active).

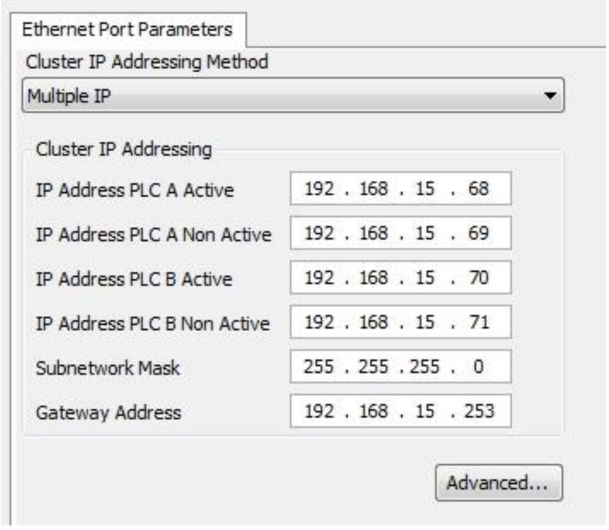


Figure 6-13. Multiple IPs method

Parameters that must be configured in the Multiple IP method:

- IP Address PLC A Active: PLCA communication address, when in Active state.
- IP Address PLC A Non Active: PLCA communication address, when in Non-Active state.
- IP Address PLC B Active: PLCB communication address, when in Active state.
- IP Address PLC B Non Active: PLCB communication address, when in Non-Active state.
- Subnetwork Mask.
- Gateway Address.

NIC Teaming and Active IP Combined Use

In case a determined port pair form a NIC Teaming in a redundant CPU, these ports can implement, at the same time, the strategies NIC Teaming and Active IP.

E.g. if the NX3030 CPU NET 1 and NET 2 ports form a NIC Teaming pair, then:

- IP Address PLC A: IP address of the NET 1 + NET 2 ports in the PLCA NX3030 CPU
- IP Address PLC B: IP address of the NET 1 + NET 2 ports in the PLCB NX3030 CPU
- IP Address Active: IP address of the NET 1 + NET 2 ports in the NX3030 CPU in the Active CPU

This way, the excellent availability from the NIC Teaming strategy is associated with the practicality of the Active IP strategy, which doesn't need scripts in SCADA systems or in other clients connected to the Active CPU server.

Ethernet Interfaces Use with Vital Fault Indication

The Ethernet ports of NX3030 and NX5000 modules can be configured to generate vital failures. This option is important for applications in which the modules of inputs and outputs are distributed over Ethernet network. In this case, if a failure occurs on the Ethernet port, this will generate a switchover. This behavior is applicable only to Ethernet ports where there is at least a communication driver that manages fault.

The communication drivers that generate vital failure are MODBUS client and MODBUS Symbol Client (all references to MODBUS Client in the following sections apply to both cases). The MODBUS Server drivers, MODBUS Symbol Server and EtherCAT Master do not generate vital failure. Thus, if an Ethernet port has a MODBUS Client driver configured and a failure occurs in the Ethernet port, a switchover will be generated if vital fault option is enabled. If the driver configured on the Ethernet port is a MODBUS Server, even if there is failure in the door, it will not generate a vital failure that causes a switchover.

To a fault be considered a vital failure in an Ethernet port on a MODBUS Client, all servers configured in the driver must be faulty. That is, if there is more of a MODBUS Client driver configured in the same Ethernet port, is considered vital fails when all servers of both Clients are faulty.

When the Ethernet port is configured to operate with NIC Teaming, the vital failure will be considered only when the two pair of doors fails.

Failure in Ethernet Interface

A switchover can be generated due to failure in the Ethernet interface, such as a loss of link. The link loss may be caused, for example, by a cable breakage or failure of a switch on the Ethernet network. Accordingly, it is necessary that, in addition to being configured to generate vital failure, there is a MODBUS Client instance configured on the Ethernet interface.

When the interval of MainTask is greater than or equal to 100 ms after the fault is detected the switchover will occur in up to two cycles of MainTask. When the interval of MainTask is less than 100 ms switchover will occur within 100 ms plus the time of MainTask after detection of failure.

Failure in Connected MODBUS Server

The time to detect the fault in a remote MODBUS Server depends on the time-out settings configured on each MODBUS Client. When a fault is detected in all Servers, the bAllDevicesCommFailure diagnostic (see Modbus Diagnostics used at Redundancy section used in) changes its state to TRUE. When this happens, the switchover will happen 3 seconds after this transition.

OPC Communication Use with Redundant Projects

The OPC protocol can be configured to communicate with redundant clusters over SCADA systems. When this option is selected in the creation of a redundant project, the Symbol Configuration object is added to the project. In this object are set system variables that will be sent to the SCADA system. This communication option is enabled in the CPU of the Ethernet ports NX3030. For further information related to the configuration of an OPC communication with redundant projects, refer to the Configuration with the PLC on the OPC Server with Connection Redundancy section of this Manual.

Redundant CPU States

In a redundant system, a CPU (PLCA or PLCB) may assume the following states:

- Active
- Stand-by
- Inactive
- Not-Configured
- Starting

ATTENTION: Frequently this manual will use the designation “Non-Active” for each state different from Active, in other words, to design any one from the other 4 states (Stand-by, Inactive, Not-Configured and Starting). An Active CPU is the one that is in Active state and a Non-Active CPU is the one that isn’t in Active state.

In the following sections these five states are briefly described. Further details regarding the redundant CPU states are described in the Transition between Redundancy States section, when the state machine and the transition causes are also described.

Not-Configured State

This is the initial redundancy state. The CPU is found in this redundancy state:

- By convention, while the CPU is OFF
- Before starting the MainTask
- Before the Starting state is switched
- In case there’s a restart through a command as reset warm, reset cold or reset origin

In case the MainTask is being executed in the Not-Configured state, the following tasks are executed:

- The PROFIBUS masters are disabled
- The cyclic synchronization services are executed (see Cyclic Synchronization Services through NETA and NETB section), if the condition for its execution are true
- The sporadic synchronization services can also be executed (see Sporadic Synchronization Services through NETA and NETB section)

The CPU will be blocked in the Not-Configured state if the other CPU is in Active state, and this CPU project is different from the Active CPU project (except if the project automatic synchronization is disabled – see Project Synchronization Disabling). In case this situation doesn’t occur, a transition from the Not-Configured state for the Starting state happens as soon as a configuration request arrives.

Sometimes, the CPU goes to Not-Configured state when has already received an automatic configuration request, when the new request for Starting state changing is not necessary. This happens at the CPU energizing, for instance.

In other situations, the user must request manually this configuration, e.g. pressing a button on the PX2612 redundancy command panel. Manually configuration requests typically aren't necessary when a user maintenance is needed before going out from the Not-Configured state, e.g. if the CPU hasn't reached the Not-Configured state due to some failure.

After getting out from the Not-Configured state, the PLC can go back to this state, due to events such as:

- Restarting (reset warm, cold or origin)
- PLC switch off
- Different projects between this PLC and the Active PLC

Starting State

Different from all other 4 states which can last indefinitely, the Starting state is temporary, taking only a few seconds. This state is always reached from the Not-Configured state, through a configuration request.

At the beginning of the Starting state, several actions, tests and verifications are executed, in order to decide which will be the next state:

- PROFIBUS masters are enabled in a passive state. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen;
- Verify if the CPU identification is correct (must be PLCA or PLCB);
- Verify if there are problems in the configuration parameters extracted from MasterTool project;
- Verify the NX4010 module integrity;
- The cyclic synchronization services are executed (see Cyclic Synchronization Services through NETA and NETB section), if the conditions for its execution are true;
- Verify the firmware compatibility version between both CPUs;
- Verify if the projects from both CPUs are equal, if the project automatic synchronization is enabled (see Project Synchronization Disabling section).
- In case the other CPU is in Active state, verify the possibility to establish a passive PROFIBUS communication with it. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen;
- In case the other CPU is in unknown state due to failures in NETA and NETB, verify the possibility of establishing a passive PROFIBUS communication with it.

Depending on the results of these verifications and tests, the CPU can go from the Starting state to any from the other four states.

Active State

In this state, the CPU controls the automated process, using the ActivePrg program, executed only in this state. The Active CPU also updates the PROFIBUS remote I/O system, putting its PROFIBUS masters in active state. The active state is used to establish communication with the PROFIBUS remotes (slaves).

The Active CPU also verifies its internal diagnostic and user switchover requests to determine if a switchover is necessary. The CPU goes out from the Active state only if it knows the other CPU is in Stand-by mode, and able to assume as Active.

However, there are some situations where the Active CPU could go out from the Active state even with no certainty that the other CPU is in Stand-by state (e.g. if the CPU is switched off).

Stand-By State

In this state the CPU is ready to be switched to the Active state, in case there's a request for that, as a failure in the Active CPU.

The Stand-by CPU also verifies its own diagnostics and can be switched to the Not-Configured or Inactive state, in case some failures occur.

PROFIBUS masters are enabled in the passive state. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen. Total failure in PROFIBUS networks configured as vitals cause a switching to the Inactive state. A total failure in a PROFIBUS network damages both composing networks (redundant PROFIBUS network) and the single composing network (non-redundant PROFIBUS network).

If the Ethernet interfaces are enabled with vital failure option, clients are enabled in passive state. Total failures in Ethernet networks configured as vital cause a switch to the Inactive status. A total failure in an Ethernet network reaches the two networks that comprise (enabled Communication Redundancy option) or the only network that compose (Redundancy option disabled Communication).

Inactive State

This state is normally reached after some failure types, or due to a manual request before a programmed maintenance.

PROFIBUS masters are enabled in the passive state. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen.

Before switching to another state, first the diagnosed failures must be corrected or the programmed maintenance executed, if those have driven the CPU to Inactive state. After, a transition for the Not-Configured state must be done, requesting a configuration. Then, a switch to the Starting state must be executed. After the Starting state, the CPU can:

- Return to the Inactive state, if determine failure types remain
- Return to the Not-Configured state, in case of other failure types
- Go to Stand-by state, if the other CPU is in Active state
- Go to Active state, if the other CPU isn't in Active state

PX2612 Redundancy Command Panel Functions

The PX2612 redundancy command panel is shown on Figure 6-4, while Figure 6-5 shows its frontal view with more details. Besides this, Figure 6-6 shows how this panel must be connected to the PLCA and PLCB half-clusters.

The PX2612 is divided in two sections: one controlled by PLCA and another by PLCB. These controllers are possible through cables AL-2317/A for PLCA and AL-2317/B for PLCB, and allow each CPU to read three buttons, write on three LEDs and a NO relay contact.

Observing the frontal view on Figure 6-5:

- PLCA executes the STAND-BY and INACTIVE buttons reading in PLC A sector.
- PLCA executes the TURN ON PLC B button reading.
- PLCA executes the writing on the three LEDs (ACTIVE, STAND-BY and INACTIVE) from the PLC A sector.
- PLCA executes the writing on the RL B relay, used to switch off PLCB.
- PLCB executes the STAND-BY and INACTIVE buttons reading in the PLC B sector.
- PLCB executes the TURN ON PLC A button reading.
- PLCB executes the writing on the three LEDs (ACTIVE, STAND-BY and INACTIVE) from the PLC B sector.
- PLCB executes the writing on the RL A relay, used to switch off PLCA.

PX2612 Buttons

This section describes the functions of the PX2612 buttons.

The STAND-BY button has the following functions:

- To request a switching from the Active state to the Stand-by state, useful when maintenance in the Active CPU is needed. After the Active CPU is switched to Stand-by (and consequently the Stand-by CPU is switched to Active), it’s possible to switched from Stand-by to Inactive using the INACTIVE button, and then execute the programmed maintenance in the inactive state.
- To request a configuration which causes a switching from the Not-Configured to the Starting state, typically after the failures that caused the transition to the Not-Configured state are repaired. After the Starting state, normally the CPU is supposed to go to the Stand-by state (or Active, if the other CPU isn’t in the Active state).
- To request a switching from the Inactive state to the Not-Configured state requesting a configuration already. This occurs typically after the failures which caused the transition to the Inactive state were corrected. After the Not-Configured state, the configuration must take it to the Starting state. After the Starting state, normally the CPU is supposed to go to the Stand-by state (or Active, if the other CPU isn’t in the Active state).

The INACTIVE button requests a switching from the Stand-by state to the Inactive state, which can be useful to execute a programmed maintenance in the Stand-by CPU. After this maintenance, the STAND-BY button may be used to make it go back to the Stand-by state, passing by the Not-Configured and Starting state (see previous description of the STAND-BY button).

The TURN ON PLCx (x = B for PLCA, or x = A for PLCB) button is used to cause a reactivating in the other CPU, in case the local CPU has switched off. As it is described in the Transition between Redundancy States section, there are exceptional situations when a CPU switches off the other at assuming the Active state, in order to avoid the possibility of both CPUs to assume the Active state simultaneously.

ATTENTION:
For a button to be considered, it must be pressed for at least 1 second. Furthermore, during this second, only this button must be pressed (the other 2 buttons must be released).

ATTENTION:
There are alternative ways to generate the same effects of the STAND-BY, INACTIVE and TURN ON PLCx buttons. Commands generated by the local CPU or the remote CPU can be used, as described, preliminary, in the Diagnostics, Commands and User Data Structure section. A more detailed description of these commands can be found in the Redundancy Commands section.

PX2612 LEDs

The PX2612 LEDs are used to inform the redundancy state, as shown on the following Table 6-2:

Redundancy state	LED ACTIVE	LED STAND-BY	LED INACTIVE
Not-Configured	off	off	off
Starting	on	on	on
Active	on	off	off
Active (recent)	blinking	off	off
Active (switching off the other CPU)	on	blinking	off
Active (recent and switching off the other CPU)	blinking	blinking	off
Stand-by	off	on	off
Inactive	off	off	on

Table 6-2. PX2612 LEDs

Each LED can be off, on or blinking. In case it's blinking, it remains on for 0.5 seconds and off for the same time.

Note that there are four different animations for the Active state, due to the following features:

- At the first 2 seconds in Active state the LED ACTIVE led blinks and remains on afterwards. This animation was created because in the first instants of the Active state, the CPU won't accept commands to get out from this state. For further details regarding this Active CPU behavior, see Transition between Redundancy States and First Instants in Active State sections.
- In case this CPU is switching off the other CPU through its PC2612 relay, the LED STAND-BY blinks. It remains off otherwise.

PX2612 Relays

The PX2612 has two NO relays. The PLCA can control the RL B, to command the PLCB switching off. The PLCB can control the RL A, to command the PLCA switching off.

Such switching off situations happen in exceptional situations, described in the Transition between Redundancy States section.

Transition between Redundancy States

The following figure shows the redundancy state machine, illustrating all the possible transitions between redundancy states.

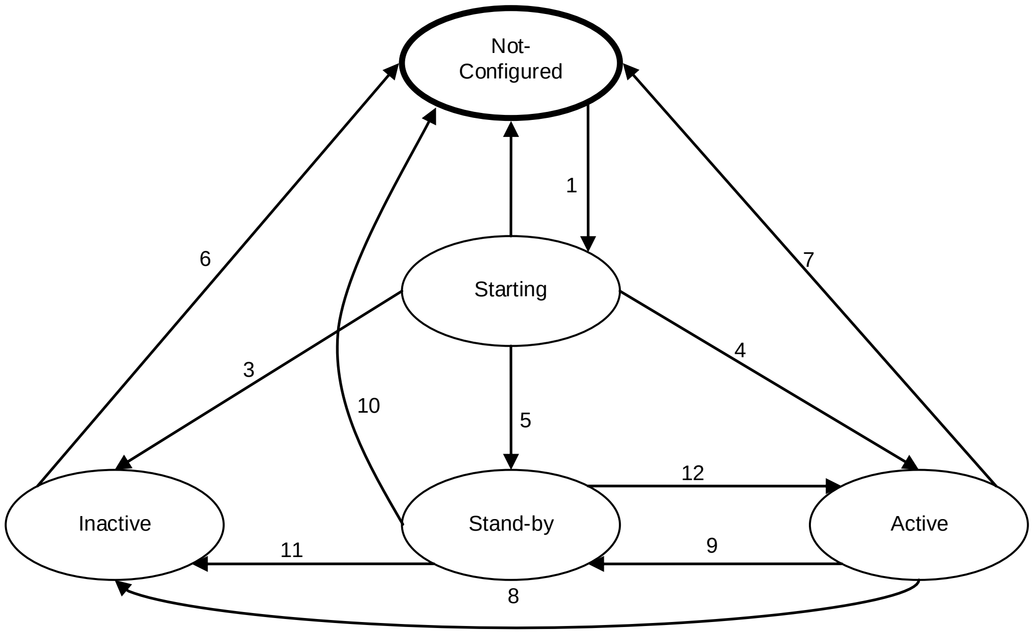


Figure 6-14. Redundancy State Machine

The following sub-sections describe all these transitions, and the causes which can trigger them. In order to interpret correctly this state machine functioning, some rules and sequences must be established:

- Transitions which originate from the same state must be analyzed in the sequence established by their number. E.g. the transitions 2, 3, 4 and 5 are originated from the Starting state. In this example, the transition 2 is first analyzed, then 3, 4 and, finally, 5. In case the transition 2 is triggered, the transitions 3, 4 and 5 won't be analyzed
- Inside a specific sub-section describing a transition, several conditions can trigger it. These conditions must be analyzed in the sequence they appear in the sub-section. Any condition that goes true can cause a transition. If a condition causes a transition, the next conditions don't need to be analyzed

- Transitions can only be triggered if the CPU is on and the MainTask is executing. Otherwise the CPU is assumed to be in the Not-Configured state
- In several cases, transitions caused by the PX2612 panel buttons are mentioned. It must be recalled there are alternatives for these buttons, which are internal commands from one CPU or the other (via NETA / NETB). Such commands were mentioned preliminary in the Diagnostics, Commands and User Data Structure section and are better described in the Redundancy Commands section. In the following sub-sections, to simplify, these commands are not mentioned, but one must remember they can cause the same transitions as the PX2612 button.

Transition 1 – Not-Configured to Starting

ATTENTION: The conditions of this sub-section must not be analyzed in case the other CPU is in Active state and the projects are different. This CPU must remain in the Not-Configured state while its project is different from the other CPU project, if the other is in Active state. This note isn't valid if the project automatic synchronization is disabled (see Project Synchronization Disabling section), as in this case differences between the CPUs projects are allowed.

- A configuration request is already existent at the beginning of the Not-Configured state. This occurs in the moment the CPU is switched on, and also other situations, described in the next sub-sections.
- The STAND-BY button was pressed during the Not-Configured state. This causes a manual configuration request. The user typically presses STAND-BY after fail repairing which had driven this CPU to the Not-Configured state.

Transition 2 – Starting to Not-Configured

- This CPU was turned off or restarted (warm reset, cold or origin) or its CPU went to Stop mode
- The identification register of this CPU is invalid (different than PLCA or PLCB)
- There are logic configuration errors in the project received from MasterTool IEC XE
- The other CPU is in the Active state and the firmware version in this CPU is incompatible with firmware version in it
- The other CPU is in Active state and the project in this CPU is different from the project in it. Besides going to the Not-Configured state, a configuration request is made. This way, after the projects are synchronized, the CPU goes out automatically from the Not-Configured state to the Starting state. This condition isn't analyzed if the project automatic synchronization is disabled (Project Synchronization Disabling section)

Transition 3 – Starting to Inactive

- NX4010 module not detected in the bus, or its microprocessor failure
- A synchronism channel (NETA or NETB) is in failure and this CPU knows this failure was caused by hardware components or internal software (internal failures of NETA or NETB)
- The Other CPU is in Active state. However, it's not possible to synchronize the redundant data or the redundant forcing list
- The other CPU state cannot be discovered through NETA / NETB, but this CPU can monitor the traffic in some configured PROFIBUS networks in vital fail mode. This way, it looks like the other CPU is controlling the process, even though NETA / NETB aren't working to confirm it
- Link loss occurred to an Ethernet Interface configured as Vital Failure

Transition 4 – Starting to Active

- The other CPU is in Non-Active state. Before the transition is possible, this condition must remain true for some time, higher to PLCB than PLCA. This way, at the moment PLCA and PLCB are simultaneously turned on; PLCA has priority to take over in Active state

- The other CPU state can’t be discovered through NETA / NETB, and besides that this CPU can’t monitor traffic in any PROFIBUS network configured as vital fail mode, or those networks weren’t created. Therefore, it really looks the other CPU if off or out of execution. For safety reasons, besides switching to Active, this CPU turns the other off using its PX2612 relay. This condition must be kept for a while before the transition is executed

Transition 5 – Starting to Stand-by

- The other PLC is in Active state. The redundant data synchronization and the redundant forcing list synchronization services are working correctly

Transition 6 – Inactive to Not-Configured

- This PLC was switched off or restarted (warm reset, cold reset or origin reset) or its CPU went to Stop mode
- The STAND-BY button was pressed on the PX2612. Besides going to the Not-Configured state, a configuration request is made. This way, the CPU goes out automatically from the Not-Configured state for the Starting state. The user typically presses this button after repairing the failure which has driven the CPU to the Inactive state
- This PLC has its synchronization disabled and the project is different from the Active PLC, at the STAND-BY button pressing, the PLC goes from Inactive to Not-Configured

Transition 7 – Active to Not-Configured

- This PLC was switched off or restarted (warm reset, cold reset or origin reset) or its CPU went to Stop mode

Transition 8 – Active to Inactive

- NX4010 module not detected in the bus, or its microprocessor failure. This CPU knows the other CPU was in Stand-by state before this failure happened. This condition isn’t analyzed in the first 2 seconds in Active state
- This PLC has lost communication with another PLC through NETA and NETB due to an internal failure but knows the other PLC was in Stand-by mode just before the failure occurred. This condition isn’t analyzed in the first 2 seconds in Active state
- This CPU can’t control all PROFIBUS networks configured in vital fail mode and knows the other CPU is in Stand-by state. This condition isn’t analyzed in the first 2 seconds in Active state.
- This CPU detected a total failure in Ethernet networks configured in vital failure mode, and knows that the other CPU is in Stand-by state

Transition 9 – Active to Stand-by

- Both PLCs, for some reason, are in Active state and this conflict must be solved. The PLCA switches to Stand-by state in case this conflict remains. The PLCB does the same after a delay smaller than PLCA. This way, in this case, PLCA has priority to remain in Active state
- The STAND-BY button was pressed and this CPU knows the other CPU is in Stand-by state. This condition isn’t analyzed in the first 2 seconds in Active state

Transition 10 – Stand-by to Not-Configured

- This PLC was switched off or restarted (warm reset, cold reset or origin reset)
- The other PLC is in Active state and it’s known this PLC project is different from the Active PLC. Besides going to the Not-Configured state, a configuration request is made. This way, after the projects synchronization, the PLC goes automatically from the Not-Configured state to the Starting state. This condition isn’t analyzed if the project automatic synchronization is disabled (Project Synchronization Disabling section)
- The other PLC is in Active state and firmware version of this PLC is incompatible with the firmware version of the Active PLC

Transition 11 – Stand-by to Inactive

- NX4010 module not detected in the bus, or its microprocessor failure
- The INACTIVE button was pressed on the PX2612. This is made typically in order to execute a programmed maintenance in the Non-Active CPU. Any programmed maintenance must be avoided in the Stand-by CPU, thus is recommended to switch to Inactive mode
- The other CPU is in Active state. However the redundant data synchronization or the redundant forcing list synchronization services haven't worked in last four cycles of the MainTask or the diagnostics synchronization service haven't worked in the last two cycles of the MainTask
- The other PLC is in Active state. However, this PLC can't monitor traffic in every PROFIBUS network configured as vital fail mode
- The other CPU is in Active state, However, this CPU detected failure in Ethernet ports configured as Vital Failure mode

Transition 12 – Stand-by to Active

- The other CPU state is unknown due to NETA and NETB failures. In this case, besides going to Active state, for safety reasons, this CPU switches off the other CPU using the PX2612 relay. When the Redundancy does not use PX2612 panel and there PROFIBUS DP this condition is not generated the remaining CP in state reserve. In this condition the fault if it has been generated by other CP, to regain control of the process is to run the command to pass the CP to the Idle state and then the command to move the CP to the state reserve. When this sequence is performed this CP will assume the Active
- The other CPU state is known and different than Active

First Instants in Active State

In the first 2 seconds in Active state, as already described in PX2612 Redundancy Command Panel Functions section, the LED ACTIVE blinks and remains on after this time has passed.

While the LED ACTIVE blinks, several transitions which, usually, could take the CPU from the Active state, aren't analyzed (see previous sub-sections that define transitions from the Active state). E.g. during this time, it doesn't work to press the STAND-BY button to try and make the CPU go to Stand-by state.

Only two conditions allow the CPU to go out of the Active state while the LED ACTIVE blinks. They are the following

- This PLC was switched off or restarted (warm reset, cold reset or origin reset), causing a transition to Not-Configured state.
- Both PLCs, for some reason, are in Active state and this conflict must be solved. The PLCA switches to Stand-by state in case this conflict remains. The PLCB does the same after a delay smaller than PLCA. This way, in this case, PLCA has priority to remain in Active state

Furthermore, in the very first instants that a PLC assumes the Active state, some non-redundant diagnostics may not be valid, such the diagnostics of the NX5000 and NX5001 modules. The method used to ignore the diagnostics possibly invalid is described in section Reading Non-Redundant Diagnostics.

Common Failures which Cause Automatic Switchovers between Half-Clusters

In this section, the more common failures which, automatically, cause a switchover from the Active CPU to Non-Active and from Stand-by CPU to Active CPU are listed. These failures trigger a sub-group of those transitions examined in the Transition between Redundancy States section.

- Power supply fault in the Active CPU. It's important that both CPUs have redundant power supplies, in order to avoid that a power supply failure doesn't affect the Stand-by CPU
- NX8000 power supply fault in the Active CPU
- Rack bus failure (NX9001, NX9002 or NX9003) in the Active CPU
- Failures in the NX3030 CPU from the Active CPU, such as:

  - Watchdog
  - Restart (reset warm, cold or origin)
  - Stop
  - Failure in the bus interfaces in one or both synchronization channels NETA and NETB

- Failures in the NX4010 from the Active PLC, such as:
  - Not recognized module in the NX3030 CPU bus
  - Failure in the NX4010 microprocessor which prevents the NETA/NETB and the PX2612 control panel (buttons, LEDs and relay) internal diagnostics updating
  - Internal failures that affect one or both synchronization channels NETA and NETB

- Active PLC PROFIBUS network total failure, in case this network is configured in vital mode. In case the PROFIBUS network is redundant, both composing networks must fail (double failure)
- Total failure of an Ethernet network in active CPU, if this network is configured with vital failure. If the Ethernet network is redundant, both networks that compose it must be faulty (double fault)

Failures Associated to Switchovers between Half-Clusters Managed by the User

Among the described transition in the Transition between Redundancy States section, some turn possible the user to manage switchovers between half-clusters, due to failures that don't generate automatic switchovers.

There are very particularly cases which depend on the philosophy of each client. E.g.: a case where the SCADA system loses the communication with the Active CPU, but keeps communicating with the Stand-by CPU.

Some clients would rather to have a manual switchover, where the operator presses the PX2612 STAND-BY button, to the Active CPU. The switchover causes a communication retry with the new Active CPU.

An alternative solution would be to cause a switchover by sending a command from the SCADA system to the Stand-by CPU, which would transmit to the Active CPU through NETA/NETB, using the RedCmdLocal (Stand-by CPU) and RedCmdRem (Active CPU) data structures to transport a command equivalent to the PX2612 STAND-BY button.

It would be also possible the Active CPU detect its communication lost with the SCADA system itself and to activate a command in the RedCmdLocal, equivalent to the PX2612 STAND-BY button. This would be a totally automatic solution with no operator intervention that would be typically made in the ActivePrg POU.

Through data structures described in the Diagnostics, Commands and User Data Structure section, it's possible to exchange diagnostics and commands between the half-clusters through NETA and NETB. This way, the user can execute special redundancy managing for failures that normally wouldn't cause any switchover. Further details regarding these data structures are offered in the following sections:

- Redundancy Diagnostics Structure
- Redundancy Commands
- User Information Exchanged between PLCA and PLCB

Below, is exemplified how the user can manage failures and execute a switchover due to an error in the Ethernet interfaces from the Active PLC (this code should be used in the ActivePrg POU):

```
//Verify if NIC Teaming is enabled.
IF ((DG_NX3030.tDetailed.Ethernet.NET1.szIP = '0.0.0.0') OR
(DG_NX3030.tDetailed.Ethernet.NET2.szIP = '0.0.0.0')) THEN

   //NIC Teaming enabled: error in two NETs to execute a switchover.
   IF (DG_NX3030.tDetailed.Ethernet.NET1.bLinkDown AND
   DG_NX3030.tDetailed.Ethernet.NET2.bLinkDown) THEN
      //Change the local PLC to StandBy.
```

```
        DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
    END_IF
ELSE
    //NIC Teaming disabled: error in one of NETs to execute a switchover.
    IF (DG_NX3030.tDetailed.Ethernet.NET1.bLinkDown OR
    DG_NX3030.tDetailed.Ethernet.NET2.bLinkDown) THEN
        //Change the local PLC to StandBy.
        DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
    END_IF
END_IF
```

ATTENTION: When two Ethernet interfaces form a NIC Teaming pair, the inactive interface will always have the IP address 0.0.0.0. This isn't a valid IP and is no possible to configure manually an interface with this address.

Fault Tolerance

The main objective of a redundant CPU is the system availability increase. The availability is the ratio between the time while the system is working properly and the total time since the system has been implemented. For instance, if a system was implemented 10 years ago and during this time, wasn't working due to failures for a year, then its availability was only 90%. This kind of availability is usually unacceptable for critic systems, where 99.99% availability is required, or even more.

In order to reach this availability level, several strategies are necessary:

- Utilization of more reliable components (with high MTBF or Mean Time between Failures), contributing for the MTBF increase of the system as a whole
- Utilization of redundancy for, at least, the most critical components or components with smaller MTBF, in such a way that a component failure can be tolerated without stopping the system. If the redundancy is implemented through components duplication, it will be necessary that both fail for the system as a whole become unavailable
- High diagnostics coverage, especially in redundant components. The component redundancy isn't very useful for the availability increase when is not possible to discover which component failed. In this case, the first failure in one component still doesn't drop the system, but remains hidden, until the second failure occurs, dropping the system, as the first failure wasn't yet repaired. The failures can be classified between diagnosable and hidden. It's strongly recommended that all redundant components failures are diagnosable
- It's also important that non-redundant components have wide diagnostics coverage, as, frequently, the system can continue working even with a non-redundant component failure. The component may not being requested, e.g. a relay with NO contact which rarely has its coil activated, doesn't have its failure detected until the moment the system requires its closing
- Low repair time for non-redundant components. A non-redundant component failure can drop the system, and during the repair, the system will be unavailable
- Possibility of repairing or substituting a redundant component without stopping the system. If this possibility exists, a great availability increase it got. Otherwise, a stop must be programmed in order to substitute the component and the repair time is computed as unavailable time
- Low repair time for redundant components. A redundant component failure doesn't drop the system, but during its repair, a failure in its redundant pair could happen. For this reason, it's important that the failure is repaired quickly after diagnosed. The higher the repair time, the higher the probability of a second failure to occur in the redundant component during this time, what would drop the system. Therefore, the higher the repair time, the lower the system availability
- Program periodic of-line tests in components in order to detect not automatically diagnosable failures by the system. The objective is to detect hidden failures, especially in redundant components or simple components which aren't being requested (e.g. a security relay). Off-line tests, sometimes, imply in system stopping what decreases the availability. Normally, special

situations, such as process programmed maintenance, are used for that purpose. The higher the period between off-line testes, the higher the time which the failure may remain hidden, and the higher the probability of a failure to damage the system, in other words, the smaller the availability

These principles were considered in the redundant CPU project using NX3030.

The next sub-sections analyze several failure types and how they are tolerated or not, and if there are switchovers associated to the tolerated failures.

Simple Failure with Unavailability

Some components, as they aren't doubled, don't even tolerate a simple failure without causing some kind of unavailability. In a redundant CPU using CPU NX3030, this is related to the following components:

- PROFIBUS remotes (slaves) in a non-redundant PROFIBUS network
- Ethernet remotes (slaves) in a non-redundant network
- I/O Modules

The failure intolerance of a non-redundant PROFIBUS network can be solved if a redundant PROFIBUS network is used, which is advisable in systems that demand a high failure tolerance. Figure 6-1 shows an example of a redundant PROFIBUS network architecture. Likewise intolerance to failure of a non-redundant Ethernet network can be solved by using a redundant Ethernet network configuration with NIC Teaming.

Regarding the I/O module unavailability, it must be observed that it doesn't imply total system unavailability. It constitutes a partial unavailability, only in the control mesh that uses this I/O module.

Even though there's no redundancy prevision for I/O modules, the user application can manage it in special cases. E.g. the user can insert 3 analog input modules in 3 different PROFIBUS remotes, and implement a vote scheme between analog inputs triples, for a critic system. However, as mentioned, such solutions must be managed by the user. There's no automatic support for them. Such solutions, generally speaking, also imply in the field transducers and actuators redundancy.

Simple Failure without Unavailability Causing a Switchover

Some redundant components tolerate simple failures without causing unavailability, but do cause switchover:

- Racks (NX9000, NX9001, NX9002 or NX9003).
- Power Supply (NX8000).
- CPUs (NX3030)
- NX4010 modules
- NX5001 modules (PROFIBUS masters) in non-redundant PROFIBUS network configuration
- NX5000 module (Ethernet) in configurations without NIC Teaming
- PROFIBUS slave interface in a redundant remote (PO5063V5, PO5065, NX5210 or AL-3416). In this case, different from the previous, the switchover happens inside the remote, between the PROFIBUS A and B networks

ATTENTION: In case of failure of the CPU NX3030 or NX4010 module in architectures where panel PX2612 or PROFIBUS network is not used, the CPU will remain in its current state. In this case, if the failure occurs in the half-cluster active, system downtime occurs.

Double Failure without Unavailability Causing a Switchover

Some components are doubled in each half-cluster, this way, before causing a switchover, both must fail:

- NX5001 modules (PROFIBUS masters) in redundant configuration, configured in vital failure mode.
- NX5000 modules (Ethernet) in configurations with NIC Teaming (redundancy managed by the user).

Redundancy Overhead

A redundant application implies on an application processing time increase, when compared to the necessary time for a non-redundant equivalent application.

This additional time happens due to cyclic synchronization services execution, described in the Cyclic Synchronization Services through NETA and NETB section, and a smaller time for the redundancy management (state machines, etc.). The total additional time due to redundancy (redundancy overhead) is estimated by MasterTool, after the redundant CPU project compiling.

ATTENTION: MasterTool calculated overhead consider an empty redundant variables forcing list.

It's up to the user to define a cycle time for the MainTask which includes:

- The additional redundancy time estimated by MasterTool
- The necessary time to execute the main POUs (NonSkippedPrg and ActivePrg). This time usually is measured after the project development (with the redundancy additional time off)
- Some MainTask cycle looseness, for other CPU tasks execution (operational system, I/O PROFIBUS drivers, Modbus, etc.). This looseness percentage can vary according to the requested performance from these other tasks. E.g. if the MODBUS communication with the SCADA system needs to allocate too much processing to reach a satisfying performance, this looseness must be increased

ATTENTION: Depending on the memory alignment, the number of bytes used in the redundancy overhead calculus might be higher than the total amount of bytes declared in the variables.

Redundant CPU Programming

Wizard for a New Redundant Project Creation

In order to create a new redundant project, the File/New Project command must be used and the Standard MasterTool Project selected.

Initially, the user must inform the desired name for the project and the directory where he desire to save it, as shown on Figure 6-15:

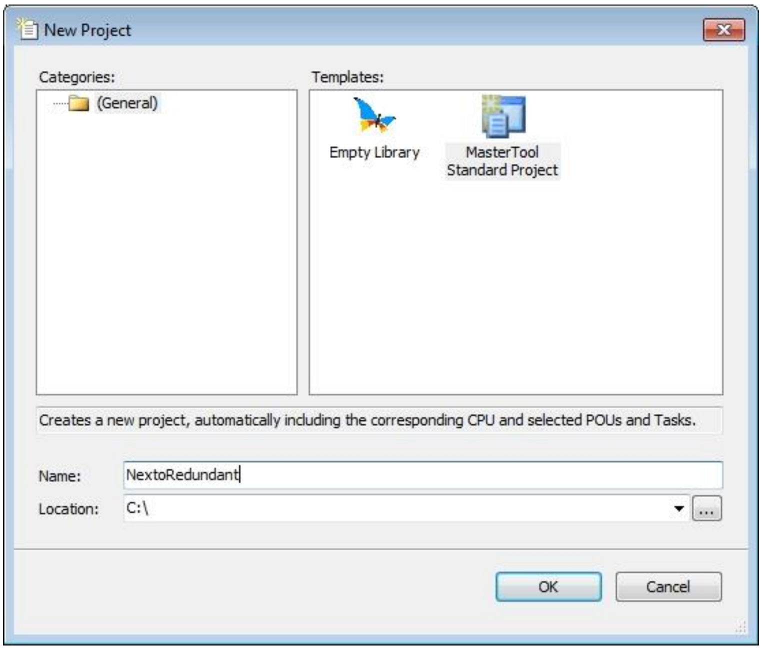


Figure 6-15. New Project

Next, the Wizard which generates the redundancy project run some questions for the user, regarding the desired configuration that must be answered successively.

The first point to be defined is the initial configuration for the half-cluster hardware:

- Select the CPU model: As the redundancy is implemented only in NX3030, it must be selected by the user
- Select the rack model: There are three rack available models and the choice depends on the module quantity used in the redundancy. For MasterTool is important the rack size according to the configured networks quantity (next wizard item)
- Select the power supply model
- Select the redundancy configuration. For a redundant project is needed to choose With Redundancy option
- Select the operation mode of redundancy. In this case the option in operation are with panel of redundancy or without (PX2612)
- Select if the OPC communication option will work or don’t
- Select if will be used redundancy with bus expansion

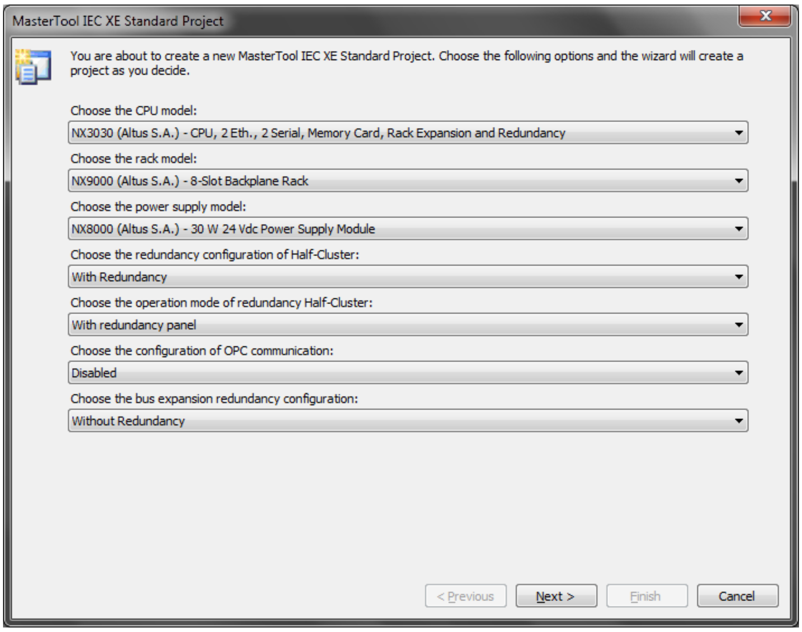


Figure 6-16. Hardware initial configuration

After, the user must define the communication networks used in the redundant application:

- Select the number of PROFIBUS networks: By the Wizard, can be created up to four PROFIBUS networks, and they can be single or redundant. It is important stress that this architecture proposed by the Wizard is typical. After that, can be created more PROFIBUS networks, respecting the maximum limit of four PROFIBUS Master modules, NX5001, in each half-cluster
- Choose the type of PROFIBUS networks:
    - There's none (no NX5001 module allocated)
    - Single (allocates one NX5001 module)
    - Redundant (allocates two NX5001 modules)
- Choose the type of Ethernet network of the CPU
    - Single Network with Failure Mode Disabled (do not generates switchover in failure case)
    - Single Network with Failure Mode Enabled (generated switchover in failure case)
    - Redundant Network with Failure Mode Disabled (operates in conjunction with the other interface and do not generates switchover in failure case)
    - Redundant Network with Failure Mode Enabled (operates in conjunction with the other interface and generates switchover in failure case)
- Choose the amount of Ethernet networks: In this case the Wizard allows the user to create up to four single networks, or up to three redundant networks, or none. It's important to stress that this is only the architecture proposed by the Wizard. After that, MasterTool allows the creation up to six networks total (three redundant maximum), always respecting the maximum limit of six Ethernet modules, NX5000, in each half-cluster.
- Select the Ethernet network type:
    - There's none (no NX5000 module allocated)
    - Single Network with Failure Mode Disabled (allocates one NX5000 and do not generates switchover in failure case)
    - Single Network with Failure Mode Enabled (allocates one NX5000 and generates switchover in failure case)
    - Redundant Network with Failure Mode Disabled (allocates two NX5000 and do not generates switchover in failure case)
    - Redundant Network with Failure Mode Enabled (allocates two NX5000 and generates switchover in failure case)

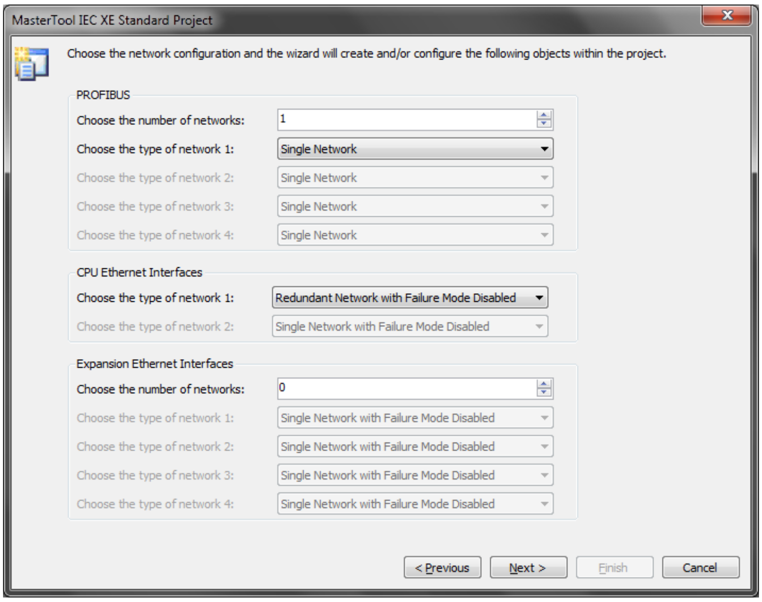


Figure 6-17. Communication networks configuration

Then the project profile and the standard language must be selected for the program creation:

- Select the project profile: It's only possible to use the simple project profile for the redundancy; hence the selection option is disabled
- Select the standard language for all programs: The language selected by the user is the standard for all programs, but any other can be used for a specific POU

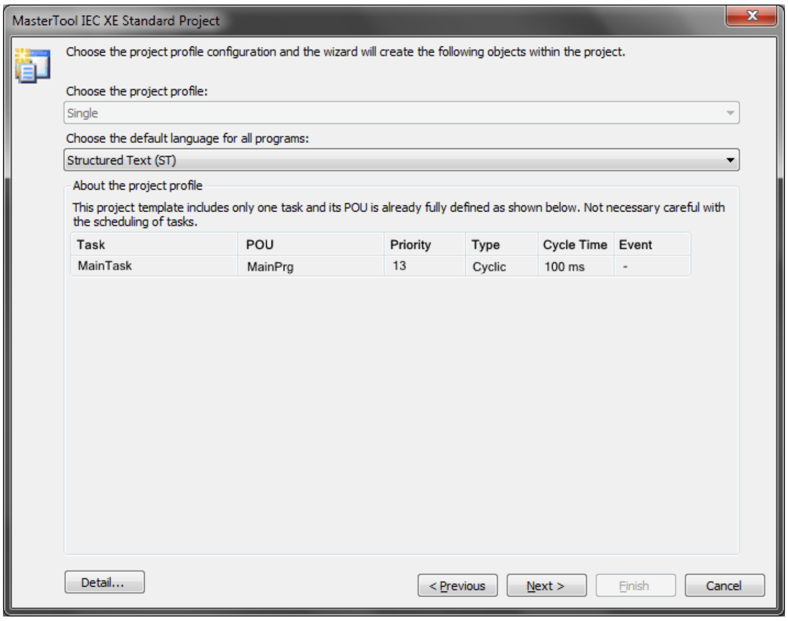


Figure 6-18. Project profile and standard language

To finish, the user must select the program language common and associated to the redundancy:

- Program associated to the MainTask (MainPrg): It must be, obligatory, in ST language, as MasterTool disables the other options
- Programs associated to the main redundancy tasks

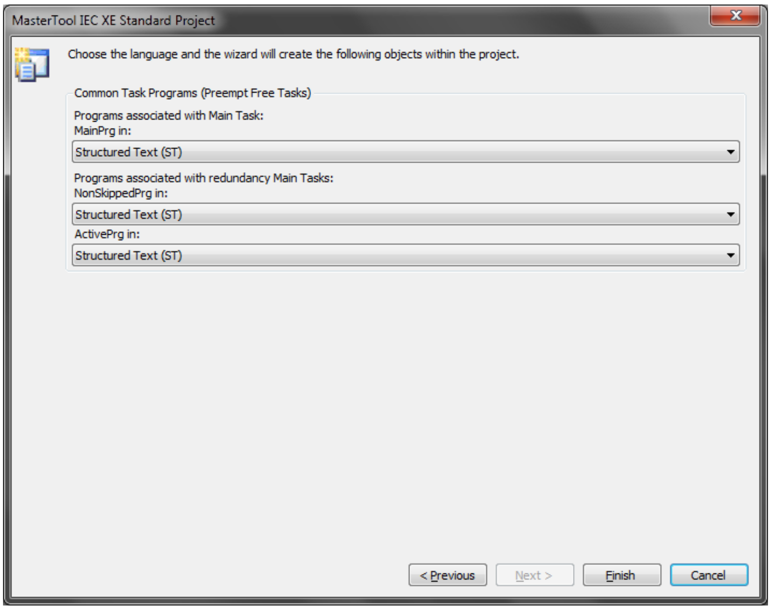


Figure 6-19. Specific programs language

ATTENTION: The ActivePrg and NonSkippedPrg POUs are created automatically, empty, in language selected on the previous questions. Other POUs which are created manually by the user can be used in any available language, except in redundant POUs which can't be written in SFC language as it uses the IEC timer as background. For further information see Limitations on a Redundant PLC Programming.

ATTENTION: The MainPrg POU will always be automatically generated in ST language, and cannot be changed by the user. This POU calls the ActivePrg (only in the Active PLC) and NonSkippedPrg (in both PLCs) POUs.

After receiving the answers for the previous questions, the Wizard generates the main project, defining a half-cluster with the following initial hardware configuration:

- Selected rack
- Power supply NX8000 (positions 0 and 1)
- NX3030 CPU (positions 2 and 3)
- NX4010 modules (positions 4 and 5) and Panel PX2612 if selected
- After the NX4010 module, NX5001 are inserted to implement PROFIBUS network with the features previously inserted by the user
- After the NX5001 modules, NX5000 are inserted to implement Ethernet network with the features previously inserted by the user

Half-Clusters Configuration

The Wizard is always used to generate the first version of a redundant project. This guarantees the initial version is generated quick and correctly.

However, it's possible that some modifications are necessary in a half-cluster, such as the insertion of new NX5001 and NX5000 modules that can be executed changing the half-cluster configuration screen. The following chapters present how to insert and configure the modules NX5000, NX5001 and NX4010.

Some rules and precautions must be followed for a redundant project, as described in the following sub-sections.

Fixed Configuration in the 0 to 5 Rack Positions

In the 0 to 5 positions of the selected rack, the following modules must be always installed:

- Power supply NX8000 (positions 0 and 1)
- NX3030 CPU (positions 2 and 3)
- NX4010 module (positions 4 and 5)

These modules must not be removed from the original project generated by the Wizard.

Any different configuration in these positions results in an error displayed by MasterTool at the project compilation.

Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2)

IP Address Configuration

Figure 6-20 presents the CPU NX3030 NET 1 port configuration (the screen for NET 2 port configuration has a subgroup of these parameters). In order to open this screen, a double click must be executed on NET 1 or NET 2, below the CPU NX3030 in the device tree.

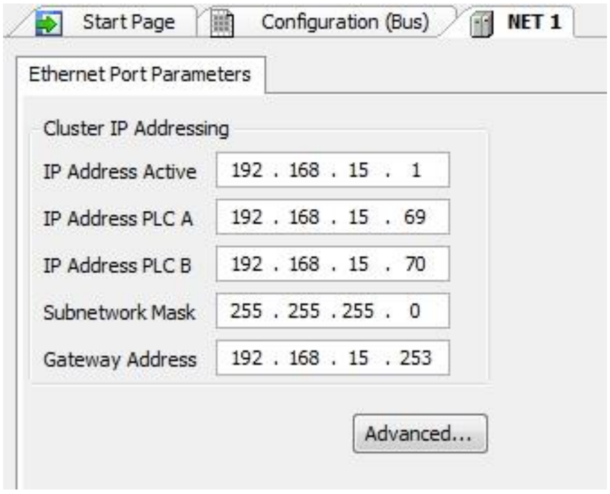


Figure 6-20. Ethernet NET 1 port parameters

Next the basic parameters of the NET 1 and NET 2 interfaces must be edited. The address has to be set according to the IP Active Change method, as described in Active IP.

ATTENTION: The NET 1 and NET 2 interfaces IP addresses, as the Gateway Address, must belong to the same subnet.

ATTENTION:
The NET 2 configuration screen has the same structure as the NET 1 configuration screen, but it doesn't have the checkbox "Redundancy", neither the NIC Teaming configuration parameters.

NIC Teaming between NET 1 and NET 2

The advanced option on the NET 1 configuration screen opens a new configuration screen, which defines if NET 1 will be redundant. In case the checkbox for Redundancy of Communication is marked, the NET 1 and NET 2 interfaces form a redundant pair with NIC Teaming, as described in the Redundant Ethernet Networks with NIC Teaming section.

Automatically, other parameters are enabled and must be configured:

- Redundancy Test Period (ms): Period to transfer the communication test frame between the two NETs. It can be configured with values between 100 and 9900
- Retries of Redundancy Test: Maximum number of times the NET, which has sent the frame, will wait for an answer. It can be configured with values between 1 and 100
- Switching Period (s): Maximum time the Active NET will wait for any package. It can be configured with values between 1 and 25

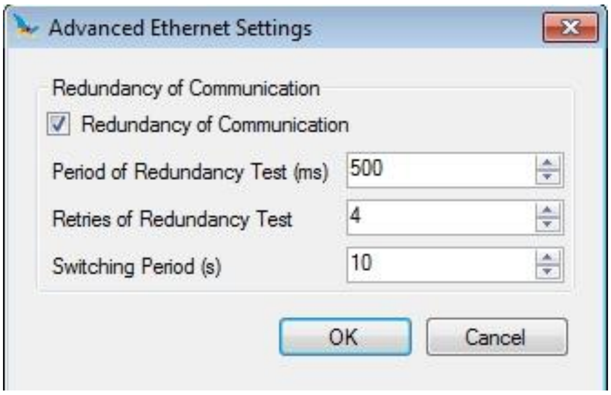


Figure 6-21. Ethernet advanced configuration

In case the answer time for the Redundancy Test reaches the Test Period times the Number of Retries and the active interface remains for a while longer than the Switching Period without receiving any package, a switchover will occur, turning active the interface that was inactive. It is important to stress that there is a delay between the failure detection and the activation of the inactive interface, due to the time necessary to interface configuration. This delay could be up to a few dozens of milliseconds.

When one of the NETs is active, it assumes the IP address configured, and the inactive NET remains with its configured IP address parameters, Subnet Mask and Gateway Address blank in the CPU diagnostics.

ATTENTION:
When a Reset Origin is performed in a CPU configured with NIC Teaming enabled for local Ethernet interfaces (NET1 and NET2), only the last active interface before the reset will be accessible. After the reset command, the accessible interface could be viewed in the CPU's Informative and Configuration Menu.

Vital failure setting in NET 1 and NET 2

The Advanced option in the setup screen of the NET 1 interfaces and NET2, opens a configuration screen where in addition to enable communication redundancy is also possible to configure if the interface will generate a switchover in case of failure as described in Ethernet Interfaces Use with Vital Fault Indication.

When configured in conjunction with the NIC Teaming redundancy, failure is considered vital failure, when a fault occurs in NET1 and NET2 interfaces.

NX5001 Modules Configuration

Insertion or Removal of NX5001 modules

NX5001 modules can be inserted or removed from the half-cluster rack. To execute this operation correctly, one must be aware of the following rules:

- The number of NX5001 modules in each half-cluster may vary between zero and four
- It can be defined up to 4 simple PROFIBUS networks or 2 redundant PROFIBUS networks, respecting the limit of 4 PROFIBUS Master NX5001 modules in each half-cluster
- When a PROFIBUS network is simple, it needs a single NX5001 module in each half-cluster. When it's redundant, it needs 2 NX5001 modules in each half-cluster
- Two NX5001 modules used to form a redundant PROFIBUS network must occupy side by side positions in the rack
- The NX5001 modules quantity in the rack must be compatible with the number of existent PROFIBUS networks and with the redundancy attribute of each network, on other words:
  - 0 x NX5001: No PROFIBUS network
  - 1 x NX5001: One simple PROFIBUS network
  - 2 x NX5001: In this case there are two options:
    - Two simple PROFIBUS network
    - One redundant PROFIBUS network
  - 3 x NX5001: In this case there are two options:
    - Three simple PROFIBUS networks
    - One redundant PROFIBUS network and one simple PROFIBUS network
  - 4 x NX5001: In this case there are three options:
    - Four simple PROFIBUS networks
    - One redundant PROFIBUS network and two simple PROFIBUS networks
    - Two redundant PROFIBUS networks

After inserting or removing the NX5001 modules, the configuration of the NX5001 modules remaining in the rack must be checked.

NX5001 Modules Parameters Adjust

Each NX5001 module used in a simple PROFIBUS network, or each redundant pair of NX5001 used in a redundant PROFIBUS network, has the following parameters to be adjusted.

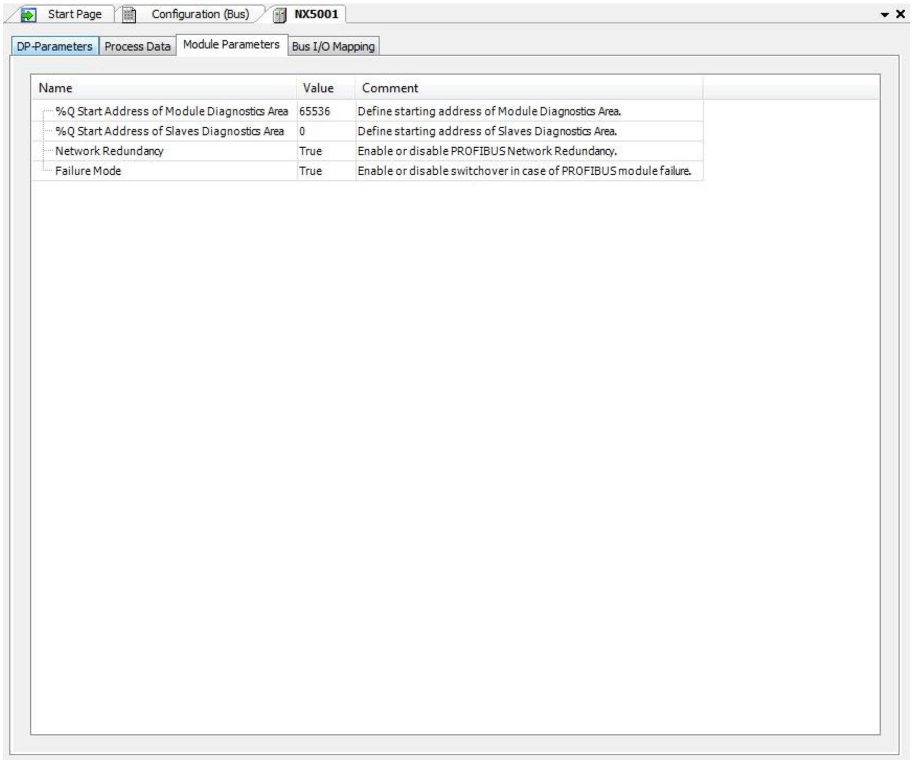


Figure 6-22. NX5001 redundancy parameters

For grouping two NX5001 modules in a redundant PROFIBUS network, a double click must be executed on an ungrouped NX5001 module which has another ungrouped NX5001 module at its right in the rack. Next the parameter “Network Redundancy”, available at the tab “Module Parameters”, must be marked as TRUE, as shown on the Figure 6-22. In order to ungroup it, the same procedure must be followed, but marking the parameter as FALSE. If this parameter is marked as TRUE, the DP parameters and the NX5001 parameters at its right are blocked for edition.

ATTENTION: In case of redundant networks, only the parameters of the NX5001 to the far left on the bus must be adjusted, while the NX5001 at the right remain blocked for edition. Some network parameters are identical to the other network while others are calculated automatically from network parameters of the left NX5001.

It’s recommended for the configured address for a NX5001 master in a redundant PLC to be 2, as the master NX5001 address in the Non-Active PLC is decremented one unit, thus the NX5001 master address results 1.

Besides that, it’s important to remember:

- The addresses from 3 to 125 are usually used for PROFIBUS slaves
- The 0 addresses are frequently used for device configuration and diagnostics
- The address 1 is reserved to be taken, dynamically, by the PROFIBUS master in the Non-Active PLC (PROFIBUS master in passive mode)
- The 126 address is frequently used for slave devices when comes from the manufacturer
- The 127 address is used for broadcast frames

In the next project compilation, MasterTool check the possible errors the user may have made at inserting or removing NX5001 modules manually.

Important to note that during the execution of a project previously configured with redundant NX5001 modules, bit 0 Command (Channel Enable Interface% QXn.0 at Bus tab: I/O Mapping) is handled by the redundant application. The interfaces must remain qualified throughout the program.

Thus, a command run by the user to disable an interface will not run the way it's expected. For example, if an interface has the status of this bit changed from TRUE to FALSE on an active CPU, this will not be interpreted as a failure that would take the CPU Active for the Inactive state. In this case, the CPU will remain in active and the other CPU that will go to the inoperative state. For these reasons, this command bit should not be manipulated by the user in a redundant application.

For further information regarding PROFIBUS networks configuration, see PROFIBUS-DP NX5001 Utilization Manual.

PROFIBUS Remotes Configuration

To configure PROFIBUS remotes under a NX5001 master, the PROFIBUS-DP NX5001 Master Utilization Manual must be consulted, together with the following manuals:

- Ponto Series Utilization Manual
- PROFIBUS PO5063V1 Head Utilization Manual and Redundant PROFIBUS PO5063V5 Head
- PROFIBUS PO5064 Head Utilization Manual and Redundant PROFIBUS PO5065 Head
- HART over PROFIBUS Network Utilization Manual

For a redundant system we must pay attention to the configuration of the watchdog parameter from the PROFIBUS remote. In case that, in the remote configuration screen, the "Watchdog control" checkbox is checked, the "Time" field needs to be correctly configured. There are two options to configure the Time and we must use the bigger time between:

- $WT \geq I \times 2 + 500ms$; and
- $WT \geq I \times 3$;

Where *WT* is the watchdog time and *I* is the MainTask configured interval.

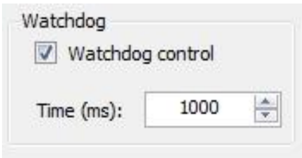


Figure 6-23. Watchdog Configuration of a PROFIBUS Remote

NX5000 Modules Configuration

NX5000 Modules Insertion or Removal

NX5000 modules can be inserted or removed from the half-cluster rack. To execute this operation correctly, one must be aware that the number of NX5000 modules in each half-cluster can vary between zero and six. Care must be taken to the fact that modules which form a redundant NIC Teaming pair must be inserted in side by side positions in the rack.

In the next project compilation, MasterTool check the possible errors the user may have committed at inserting or removing NX5000 modules manually. For instance, if the user inserted more than 6 NX5000 modules, an error occurs.

The interface of each module will be identified as NET 1, as they are identified physically on the product. In case the user adds manually NX5000 modules in the bus, the identification occurs the same way as the Wizard.

After inserting or removing the NX5000 modules, the configuration of the NX5000 modules remaining in the rack must be checked.

NX5000 Modules Configuration

For each NX5000 module in a redundant PLC, the address parameters must be adjusted as described in the IP Change Methods section, which can be accessed through a double click on the NET 1 interface, below each NX5000 module placed on the devices tree.

ATTENTION: In case two consecutive modules form a redundant NIC Teaming pair, only the basic parameters of the left NX5000 should be edited, the right NX5000's parameters edition will be blocked.

NX5000 Modules Grouping with NIC Teaming Redundancy

The NX5000 modules, as the CPU NX3030 and NX3020 NET 1 interface, present a screen of advanced configuration which defines if the module forms a redundant NIC Teaming pair with the module at its right. The configuration is made as described in the NIC Teaming between NET 1 and NET 2.

To group two NX5000 modules with a redundant pair, the following conditions must be true:

- Both NX5000 modules must be inserted in close positions in the rack.

At doing this the right module has its parameters edition blocked and the left module parameters turn to be the same to both modules.

Unmarking the checkbox "Redundancy of Communication" at the left module causes the modules' separation, making them behaves as individual modules without NIC Teaming redundancy again.

Failure Vital Setting

The NX5000 modules as well as the NET 1 and NET 2 interfaces allow you to configure if the interface will generate a switchover in case of failure, as described in Ethernet Interfaces Use with Vital Fault Indication When configured in conjunction with the NIC Teaming redundancy vital failure will be considered when failure occurs in both modules of the redundant pair.

NX4010 Redundancy Configuration

The configuration regarding the %I, %Q and %M redundant variables can be accessed through a double click on the NX4010 module, following the selection of the tab "Redundancy Parameters".

To understand these parameters the sections Redundant and Non-redundant %I Variables, Redundant and Non-redundant %I Variables and Redundant and Non-redundant %I Variables must be read.

The following parameters must be configured:

Configuration	Description	Default	Options
Redundancy %M memory offset	**Memory (%M)**		
	Redundant %M memory initial address	0	0 (disabled)
Redundancy %M memory length	Redundant %M memory size	0	0 to 65536
Redundancy %I memory offset	**Memory (%I)**		
	Redundant %I memory initial address	0	0 (disabled)
Redundancy %I memory length	Redundant %I memory size	16384	0 to 81920
Redundancy %Q memory offset reserved for I/O drivers	**Memory (%Q)**		
	%Q redundant memory offset reserved for I/O drivers initial address	0	0 (disabled)
Redundancy %Q memory length reserved for I/O drivers	%Q redundant memory offset reserved for I/O drivers size	16384	0 to 81920
Redundancy %Q	%Q redundant memory	65536	0 to 81919

memory offset reserved for diagnostics	offset reserved for diagnostics initial address		
Redundancy %Q memory length reserved for diagnostics	%Q redundant memory offset reserved for diagnostics size	16384	0 to 81920

Table 6-3. NX4010 parameters

I/O Drivers Configuration

The configuration of I/O drivers, at first, isn't different in relation to a non-redundant CPU.

What can be observed is that some I/O drivers have commands which allow its use in a redundant CPU, but it doesn't imply in configuration differences. These commands, normally, must be executed in the NonSkippedPrg program. E.g. a MODBUS RTU master driver in a RS-485 serial network must be disabled in a non-Active CPU using the code inserted by the user in NonSkippedPrg. More information regarding administration of MODBUS driver in a redundant system can be found in the MODBUS Instances Managing in Redundant System section.

In the case of PROFIBUS network, there are also special different commands for the CPUs in Active and Non-Active states. In this case, however, the redundancy management executes such commands automatically, without any user management.

To configure PROFIBUS I/O remotes, including remotes and I/O modules, see NX5001 Modules Configuration section from this manual.

MainTask Configuration

The configuration screen associated to the only task of a redundant CPU, called MainTask, which is cyclic, can be accessed through a click on the MainTask in the Device Tree.

Two parameters must be adjusted on this screen:

- The MainTask cycle time
- Watchdog time

Furthermore, the screen shows an estimative of the necessary time to manage the redundancy, calculated by MasterTool. Such estimative is only reliable after the project is complete, with all POUs developed and redundant memory areas defined.

Several considerations must be taken in order to adjust correctly the MainTask cycle time:

- The cycle time must be sufficiently low to allow the proper process control, taken in account all control feedback times
- The cycle time must be high enough for allowing, at least, the sum of the following times:
    - The NonSkippedPrg and ActivePrg POUs maximum execution time, together
    - The necessary time to manage the redundancy (redundancy overhead)
- Besides this, the cycle time must have an additional looseness necessary for the other processes execution times (PROFIBUS communication, Ethernet communication with SCADA systems, etc.)

MasterTool has conditions of calculating the necessary time for redundancy management (redundancy overhead), after the project is finished (all developed POUs and redundant memory areas defined).

Regarding the NonSkippedPrg and ActivePrg POUs execution maximum time, they are possible to be measured after these POUs are already developed. Initially, MasterTool estimates 10ms for these two POUs maximum time, together, but the user must revise this field afterwards, when measuring using the final project.

After each compilation, MasterTool sums the redundancy overhead calculated with the parameter which informs the POU times (NonSkippedPrg and ActivePrg), and verifies is the minimum looseness parameterized is being obeyed.

E.g.:

- Parameters configured in the MainTask screen:
  - MainTask cycle time: 100 ms
  - POUs NonSkippedPrg + ActivePrg estimated time: 10 ms
  - Minimum tolerance: 30%
- calculated Overhead for redundancy: 50 ms

In this case, the total time used is 60ms (10 ms + 50 ms), which consists in 60% of the MainTask cycle (100ms). This way, the maximum looseness is 40% and the minimum looseness of 30% is being respected.

ATTENTION:
A compilation error is produced in case the minimum looseness isn't respected, if it is configured in the CPU Project Parameters..

ATTENTION:
The compilation being successful or not, MasterTool informs the calculated looseness and the redundancy overhead predicted on the message window

ActivePrg Program

In this POU the user must create the main application, responsible for its process control. This POU is called by the main POU (MainPrg), being executed only in the Active CPU.

The user can also create additional POUs (program, function or function block), and call or instance them inside the ActivePrg POU, in order to structure his program. It's possible to call functions and instance function blocks defined in libraries, too.

It must be remembered that all symbolic variables defined in the ActivePrg POU, as the instances of function blocks, are redundant variables.

Symbolic variables defined in additional POUs from the program type which are called inside the ActivePrg, are also redundant variables.

ATTENTION:
Variables from the type VAR_TEMP must not be used in the redundant program.

NonSkippedPrg Program

This POU is used for controls which must be executed in both CPUs (PLCA and PLCB), independent on the redundancy state. This POU is also called by the main POU (MainPrg).

It must be remembered that all symbolic variables defined in the NonSkippedPrg POU, as well as the function blocks instances, are non-redundant variables.

The user must create additional POUs (program, function or function block), and call or instance them inside the NonSkippedPrg POU, in order to structure his program. It's possible to call functions and instance function blocks defined in libraries, too.

ATTENTION:
It must be avoided to call additional POUs from the program type inside the NonSkippedPrg, as symbolic variables declared in this type of POU are redundant, and inside the NonSkippedPrg it's normally desirable non-redundant variables. Usually the NonSkippedPrg code is small and doesn't need to call additional POUs from the program type for its structure. If the NonSkippedPrg structure is needed, function blocks or functions must be used.

Typical examples of controls executed in the NonSkippedPrg are the following:

- To create a compact diagnostics structure (%Q) to be reported to a SCADA system, from a complete diagnostics structure, where many diagnostics are not interesting for the SCADA system. These diagnostics can be extracted from data structures as RedDgnLoc, RedDgnRem, RedUsrLoc, RedUsrRem, etc.
- To copy commands received from a SCADA system for the respective data structure RedCmdLoc fields, and interconnect these commands if necessary
- To manage switchovers controlled by the user, in case of not vital failures such as the communication with a SCADA system or with a MODBUS device
- Enable and disable some specific I/O drivers, depending on the redundancy state (Active or Non-Active). E.g. a MODBUS RTU master driver in a RS-485 bus must be disabled in the Non-Active CPU. For further information see MODBUS Instances Managing in Redundant System section

ATTENTION:
It's not recommended to use function blocks TOF_RET, TON_RET, TOF and TON in the NonSkippedPrg program. See Limitations on a Redundant PLC Programming.

Redundancy Configuration Object

This object, located in the device tree, is automatically created by the Wizard. It is used to determine which POUs and GVLs are redundant, and therefore synchronized between CPs. By default, POUs and GVLs created by the user are marked as redundant, leaving the option to the user to reverse the marking when needed.

ATTENTION:
PV, PIDControl and PidRetainGVL objects can't be individually marked. In case of need to modification, the Select All option must be marked.

GVL Diagnostics

This special GVL is created and filled automatic by the Wizard and can't be modified by the user.

System diagnostics and commands, including redundancy data structure (RedDgnLoc, RedDgnRem, RedCmdLoc, RedCmdRem), are placed within direct representation special variables %Q or %I.

The SystemAT_GVL has many sentences with the AT keyword to define symbolic names for these diagnostics and commands. This way, when the user needs to reference these variables, he can use a symbolic name instead a numeric reference.

GVLs with Redundant Symbolic Variables

The user can create other GVLs different from the previously listed, in order to declare redundant symbolic variables. For that, after the GVL creation, it's necessary to mark it in the object configuration Redundancy Configuration, in the project devices tree. By default, all GVLs created by the user are, initially, redundant.

ATTENTION:
For good practice it's recommended to avoid the AT directive use in GVLs which have redundant symbolic variables declaration to prevent variable mapping in non-redundant areas.

POUs from the Program Type with Redundant Symbolic Variables

The user can declare redundant symbolic variables in POUs from the program type, with exception of the NonSkippedPrg POU where the symbolic variables declared are considered redundant.

In order to define a new POU as redundant, it must be marked in the Redundancy Configuration object after its creation, in the project devices tree. By default, all POUs created by the user are, initially, redundant.

ATTENTION: For good practice it's recommended to avoid the AT directive use in POUs which have redundant symbolic variables declaration to prevent variable mapping in non-redundant areas.

Breakpoints Utilization in Redundant Systems

For redundant systems it's recommended to use breakpoints only in the Active half-cluster, with the other half-cluster deactivated. If not, when the application execution reaches a breakpoint, the Stand-by breakpoint will take over the Active state, switching off the Active PLC.

MODBUS Instances Managing in Redundant System

The MODBUS instances are independent from the redundancy, thus they must be managed in the application, when it's up to the user to choose which instances must be enabled/disabled when a PLC goes to Non-Active state.

The example below, inserted in a NonSkippedPrg program, executes the verification of the PLC current state and in case it's in Non-Active state, disables the MODBUS RTU instances master and slave and the MODBUS Ethernet Server instance:

```
VAR
    eRedStateLocal : REDUNDANCY_STATE;
    eRedStateLocal_old : REDUNDANCY_STATE;
END_VAR

// Local PLC current state reading
eRedStateLocal := DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;

// Has the local PLC state changed?
IF eRedStateLocal <> eRedStateLocal_old THEN
    IF eRedStateLocal = REDUNDANCY_STATE.ACTIVE THEN
        // The local PLC has entered the Active state
        Diagnostics.DG_MODBUS_RTU_Slave.tCommand.bRestart := TRUE;
        Diagnostics.DG_MODBUS_RTU_Master.tCommand.bRestart := TRUE;
        Diagnostics.DG_MODBUS_Server.tCommand.bRestart := TRUE;
    ELSE
        // The local PLC has entered the Not Active state
        Diagnostics.DG_MODBUS_RTU_Slave.tCommand.bStop := TRUE;
        Diagnostics.DG_MODBUS_RTU_Master.tCommand.bStop := TRUE;
        Diagnostics.DG_MODBUS_Server.tCommand.bStop := TRUE;
    END_IF
        // Saves the last state of the local PLC
        eRedStateLocal_old:= eRedStateLocal;
END_IF
```

Limitations on a Redundant PLC Programming

On a redundant PLC there are some limitations regarding its half-cluster programming. These limitations are treated in the subsections below.

Limitations in Redundant GVLs and POUs

In a redundant GVL or a POU from the program type the following limitations must be respected for a correct functioning of the half-clusters:

- Do not use variables from the type VAR_TEMP
- Do not mix variable types (VAR, VAR_RETAIN, VAR_PERSISTENT, etc.). Only one type must be used in each GVL or POU
- Do not mix symbolic variables declaration with ATs in the GVLs. Separate GVLs must be created where in one the AT variables will be declared and in another, the symbolic variables
- Do not store a variable address in a redundant variable (use a redundant variable as a pointer), as the variable addresses may be different in the PLCA and PLCB
- Do not use the function blocks for RTC reading and writing in redundant POUs. More details can be found on the chapter RTC Clock

Non-redundant Program Limitations (NonSkippedPrg)

In a POU from the program type which aren't redundant, the case of a NonSkippedPrg POU, the following limitations must be respected for a correct functioning of the half-clusters:

- The traditional function blocks TON and TOF can't be used as they use the IEC timer. When the Stand-by PLC goes to Active state (with the other half-cluster coming out of Active state), the IEC timer is synchronized, causing a discontinuity in the timer value. The function blocks TON_NR and TOF_NR must be used instead, available in the NextoStandard library. See Configuration – Non-Redundant Timer
- POUs from the program type written in the SFC language (Sequence Function Chart) must not be used, as they use the IEC timer for transition timing
- Do not mix symbolic variables declaration with ATs in the GVLs. Separate GVLs must be created where in one the AT variables will be declared and in another, the symbolic variables

Getting the Redundancy State of a Half-Cluster

It is possible to verify the redundancy state of a half-cluster in the Redundancy Diagnostics Structure:

```
VAR
eRedStateLocal : REDUNDANCY_STATE;
END_VAR

eRedStateLocal := DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;
```

This way, the user can control a program logic that depends on redundancy state of the PLC.

Reading Non-Redundant Diagnostics

A redundant project, besides present redundant diagnostics (Redundancy Diagnostics Structure or the diagnostics from a PROFIBUS remote), presents also non-redundant diagnostics (diagnostics from the modules NX5000, NX5001, NX3030, etc.). These non-redundant diagnostics could be invalid and must not be considered at the first instants in Active state, as they aren't synchronized with the other PLC (the diagnostic state when the remote PLC was active is unknown). Therefore, these diagnostics must be ignored during the first moments in Active state, until they have valid values. Typically the time during which the diagnostics should not be considered is 5 s.

The example below shows how to not consider the diagnostics bSlaveNotPresent and bPbusCommFail from the NX5000 PROFIBUS Master module:

Logic in NonSkippedPrg:

```
PROGRAM NonSkippedPrg
VAR

   TON_DiagEnable : TON_NR;
   bDiagEnable : BOOL;
   bIsActiveState : BOOL;
```

```
    bIsActiveState_old : BOOL;
END_VAR

bIsActiveState := (DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState
= REDUNDANCY_STATE.ACTIVE);
TON_DiagEnable(IN:= (bIsActiveState = bIsActiveState_old), PT:= T#5S, Q=>
bDiagEnable);
bIsActiveState_old := bIsActiveState;
```

Logic in ActivePrg:

```
IF NonSkippedPrg.bDiagEnable THEN
    IF DG_NX5001.tGeneral.bSlaveNotPresent OR
    DG_NX5001.tGeneral.bPbusCommFail THEN
        //Actions executed when the diagnostics are active
    END_IF
END_IF
```

Redundant CPU Program Downloading

The Redundant CPU Programming section described issues related to the development of a project for a redundant CPU with NX3030 CPU.

In this section, many methods and steps to download this project in a redundant CPU are described, considering situations such as:

- Downloading the project in a brand new NX3030 CPU or in a CPU with an unknown project
- On-line modifications downloading
- Off-line modifications downloading with the process control interruption, during a programmed process stopping
- Off-line modifications downloading without the process control interruption, using redundancy features

Initial Downloading of a Redundant Project

This section describes the necessary steps to run the first download of a redundant project in a NX3030 CPU. This is necessary, for instance, for a brand new CPU recently manufactured, or for a CPU that has an unknown project.

ATTENTION: The following steps must be executed for both half-clusters (PLCA and PLCB) which compose a redundant CPU. First all steps must be executed for one CPU and then for the other.

First Step – IP Address Discovering for MasterTool Connection

The first step is to discover the IP address from the NET 1 channel in this CPU, for MasterTool connection.

This must be done through NX3030 CPU display and button, as described in the Configuration – CPU's Informative and Configuration Menu chapter. The NETWORK menu informs the IP address which can be used for MasterTool connection.

Second Step – Verifying IP Addresses Conflict

Before executing the third step, one must be sure there's no other equipment with the same IP address connected to the network, discovered in the first step. This can be discovered, for instance, disconnecting the CPU from the network and executing a "ping" in its IP address. As the CPU is disconnected from the network, the "ping" function must fail. If not, there's equipment with the same IP address.

In case the IP address is already being used by equipment in the network, the third step must be executed, and some of the following steps too, using a crossover cable to connect MasterTool to the CPU, avoiding IP addresses conflict. In one of the following cases, at downloading the project in the CPU, the definitive IP addresses are updated in it (see Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2) section).

Third Step – Preparing MasterTool Connection (Set Active Path)

The third step consists in double-clicking on the Device (NX3030 PLC) in the Device Tree, getting in the tab "Communication Settings", clicking on the Gateway, and pressing the "Scan Network" button to list all CPUs detected by MasterTool in the network.

At this moment, a CPU whose identification has the IP address found in the first step is supposed to appear. In case the user has changed the network CPU name previously, this name will be visualized. MasterTool Connection with a NX3030 CPU from a Redundant PLC section describes with more details the possible identifications which can be observed on this list. Anyhow, all possible identification has a field showing the IP address or part of it.

For instance, the bytes between square brackets form the CPU address. The right byte inside the brackets, indicate the IP address end in hexadecimal. If the bytes form the address [0010], this means the byte with value "10" indicates that the CPU IP address end is xxx.xxx.xxx.16. Next, the CPU in the list must be clicked and the "Set active path" button pressed. This done, the selected CPU must appear stressed on the list, indicating MasterTool is prepared to connect to this CPU.

Forth Step – Identifying the NX3030 CPU and Verifying the CPU Display

The forth step consists in identifying the half-cluster as PLCA or PLCB. This is made through the Online / Cluster Basic Configuration menu:

Next, the combo-box "PLC Identification" allows selecting one out of the three following options:

- PLC A
- PLC B
- Non-Redundant

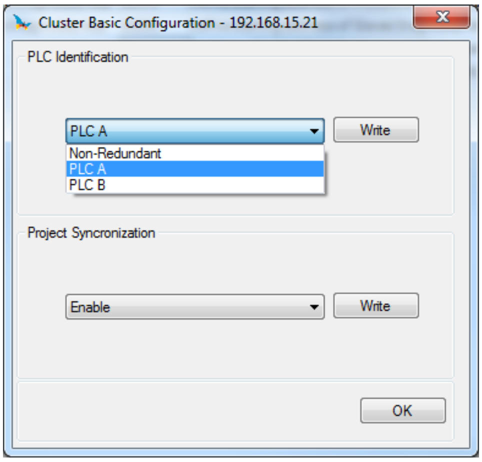


Figure 6-24. PLC Identification

In case of a redundant CPU, the user must select PLCA or PLCB. After selecting the desired option, the "Write" button correspondent to this combo-box must be pressed. MasterTool returns a message indicating command success or failure, after the Button is pressed. MasterTool returns a message warning that the CPU will be restarted and waits for the user to confirm the action. Then a message indicating command success or failure will appear. If there's success the CPU will be restarted.

ATTENTION:
The NX3030 CPU can't be in Run mode when this command is executed. Before executing this command, the user must put the CPU to Stop mode. In case the CPU is in Run mode, the command isn't executed and MasterTool warns the command has failed.

Just after executing the identification command with success, it can be observed that the selected identification appears on the Redundancy Diagnostics on the NX3030 CPU Graphic Display.

The CPU identification is also available in an internal diagnostic (DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.ePLC_ID). This diagnostic is updated from the non-volatile memory each MainTask cycle, so it's necessary for the CPU to go back to Run mode to update it. The codes returned by the diagnostics and their respective limitations are listed below:

- Non-Redundant: 0
- PLCA: 2
- PLCB: 3

The CPU identification isn't part of the redundant project developed with MasterTool. Such identification is only in a CPU non-volatile memory area, which can be modified using MasterTool.

CAUTION:
The redundancy doesn't work properly in case one of the CPUs isn't identified as PLCA and the other PLCB, when a process control interruption may occur. In case one NX3030 CPU must be replaced (e.g. after a damage), the new CPU must be previously identified with the same identification of the damaged one. The CPU display must be used to verify if both CPUs are correctly identified.

Fifth Step – Redundant Project Downloading

This step describes the redundant project downloading in the CPU. This project must be prepared according to the Redundant CPU Programming section.

A simple project (basic) can be prepared following, at least, the next sub-sections presented in this section:

- Wizard for a New Redundant Project Creation
- Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2).

Obviously, it's also possible to build a complete project and only later download it in the PLCA and PLCB, for instance, in case these CPUs hardware aren't available during the project developing in MasterTool.

The first download of a redundant project in a CPU, previously identified as PLCA or PLCB, still must be done using the IP address discovered in the first step, and selecting the third step of this procedure.

The project download is run through the Online / Login menu.

ATTENTION:
Inside the developed project using MasterTool and downloaded in the CPU in this step, were defined new IP addresses for the NET 1 interface in the PLCA and PLCB (IP_A_1 and IP_B_1), as well as a IP address for the NET 1 interface in the Active CPU (IP_Active_1) – see Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2) section.

Therefore, after this first download, the IP address discovered in the first step of this procedure usually isn't valid anymore. This IP Address change in NET 1 causes a connection loss between MasterTool and the CPU, which is showed on the screen.

For further details regarding MasterTool reconnection, see MasterTool Connection with a NX3030 CPU from a Redundant PLC section.

MasterTool Connection with a NX3030 CPU from a Redundant PLC

After executing the procedure described in the Initial Downloading of a Redundant Project section in both PLCs (PLCA and PLCB), MasterTool connection, through the NET 1 interface from NX3030 CPU can be made through one of the following addresses:

- PLCA IP address: NET1 address exclusive for PLCA
- PLCB IP address: NET1 address exclusive for PLCB

Independent from the PLC state, MasterTool can only connect to it using the PLC exclusive address, configured in PLCX IP Address. But in case the PLC is in Active state, all other services can connect to it either by the PLCX IP address or by the Active IP address.

To connect to a specific CP, at first a double-click must be done on the Device (NX3030 CPU) in the Device Tree, go into “Communication Settings” tab, click on the Gateway and press “Scan Network” button to list all PLCs detected by MasterTool in the network.

On this list it’s possible to find the following standard identifications, in case the PLC name on the network hasn’t been changed previously by the user:

- NX3030_<IP address>_PLCA: identification related to the PLCA. In this case, the field <IP address> must be the same as the PLCA IP address configured in the project;
- NX3030_<IP address>_PLCB: identification related to the PLCB. In this case, the field <IP address> must be the same as the PLCB IP address configured in the project.

Next, the PLC which MasterTool is to connect must be selected from the list and the button “Set Active Path” must be pressed. Then, at executing the command from the Online / Login menu, MasterTool connects to this PLC.

ATTENTION: MasterTool can only connect to one PLC at a time. To connect to several PLCs, multiple instances must be open in MasterTool, when care must be taken to open the correct project in each instance.

Modification Download in a Redundant Project

After both PLCs (PLCA and PLCB) from the redundant pair had its initial program already downloaded, as described in the Initial Downloading of a Redundant Project section, it’s possible to download successive changes in the project, when such changes are necessary.

MasterTool connection to the PLCs responsible for the modifications download must be executed as described in MasterTool Connection with a NX3030 CPU from a Redundant PLC section. In this section it is explained how it’s possible to connect to a specific PLC (PLCA or PLCB), to the Active PLC or to the Non-Active PLC.

Usually the modifications must be downloaded to the Active PLC and next automatically synchronized with the Non-Active PLC, through synchronism channels NETA/NETB. Therefore, MasterTool normally must use the Active PLC exclusive IP address (PLCX IP address) to connect to NET 1 channel from the NX3030 CPU in the Active PLC. In order to verify which PLC is in Active state, the same step described in Initial Downloading of a Redundant Project - Forth Step – Identifying the NX3030 CPU and Verifying the CPU Display can be followed.

ATTENTION: To download a project in the Non-Active PLC is usually useless as the project automatic synchronization (Active to Non-Active PLC) would cancel the effect of this download. However, there are special situations when the project synchronization must be disabled temporarily, being possible and useful to download a different project in the Non-Active PLC. These special situations are discussed in the Exploring the Redundancy for Off-Line downloading of Modifications without Interruption of the Process control section.

Off-Line and On-Line Modifications Download

Project modifications may be downloaded off or on-line.

Off-line downloads require the PLC, where the downloaded is supposed to be executed, stopping. On the other hand, on-line downloads allow the PLC to continue executing its application while the modification is downloaded.

Some modification types require off-line download and can't be executed on-line in the PLC where MasterTool is connected. In this case, there are two options:

- To interrupt the process control, executing the procedure described in the Off-Line Download of Modifications with Process Control Interruption section
- Use the PLC and the PROFIBUS networks redundancy in order to avoid interruption of the process control, even with the necessity to execute off-line downloads in each half-cluster (PLCA or PLCB). A procedure to reach this objective is described in the Exploring the Redundancy for Off-Line downloading of Modifications without Interruption of the Process control section

Modifications which Demand Off-Line Download and the Interruption of the Process Control

The following modifications in a project will make it impossible to be downloaded in a redundant system with no interruption of the process control:

- Modifications in the redundant memory areas (changes in the Redundancy Parameters from the module NX4010)

ATTENTION: Will not be possible to change the size of redundant memory areas without the interrupt of the process control. Thus, these areas must be carefully planned and previously configured.

Modifications which Demand Off-Line Download

The following modifications demand off-line downloads in the PLC where MasterTool is connected:

- To add or remove devices from the device tree, such as:
  - Modules in the main rack (NX5001 PROFIBUS masters, NX5000 Ethernet interfaces, etc.)
  - Remotes in PROFIBUS networks
  - I/O modules in remotes in PROFIBUS networks
  - MODBUS instances
- To modify parameters inside devices from the device tree, such as:
  - IP addresses and other Ethernet interfaces parameters
  - PROFIBUS master parameters
  - PROFIBUS remotes parameters
  - I/O modules parameters inside PROFIBUS remotes
- To modify a task's period
- Project update due to MasterTool IEC XE programmer Update.

Modifications which Allow On-Line Download

A priori, the modifications not mentioned in the sections Modifications which Demand Off-Line Download and the Interruption of the Process Control and Modifications which Demand Off-Line Download allows on-line download.

Even this way, the modifications which allow on-line download in the PLC where MasterTool is connected are listed below. These modifications are valid for variables, POUs and GVLs, redundant or not:

- To add POUs from the program type, if these POUs don't need to be associated to any task
- To remove POUs from the program type, if these POUs aren't associated to any task

- To add or remove POUs from the function or function-block type
- To modify the code of any type of POU (program, function or function block)
- To add or remove symbolic variables in any POU type (program, function or function block)
- To add or remove instances of function blocks in POUs from the program or function-block type
- To add or remove GVLs
- To add or remove symbolic variables or instances of function blocks in GVLs

On-Line Download of Modifications

In the Off-Line and On-Line Modifications Download section, modifications which demand off-line download were described, along with the ones that allow online download.

An online change must be made by connecting the MasterTool to the NET 1 channel of the active CPU, using its unique IP address. Before version 2.01 of the MasterTool IEC XE, it was necessary that the user selected the "Create Boot Application" option in the Communication menu, after sending the application for the changes to be sent to the non-volatile memory area of the CPU and could be synchronized. From version 2.01 this operation is no longer needed. After sending the application the send operation for nonvolatile memory is performed automatically.

ATTENTION: It’s important to remember that online modifications, without the option mentioned previously selected, will be lost in case of a hot reset or a CPU switch off.

ATTENTION: An online change in the declaration of retain variables of the application (adding or removing variables) followed by a drop in the power CP before "Create Boot Application" will corrupt retentive memory, because the value of the retain variables that were saved does not match the retrieved application variables in the restored memory.

When the Non-Active PLC realizes that have a different project from the Active PLC, it executes the following actions:

- Negotiates automatic project synchronization with the Active PLC
- In case it’s in the Stand-by or Starting state, it switches to the Not-Configured state and remains in it until the projects are synchronized again. After that, returns automatically to the Stand-by state
- In case it’s in the Not-Configured or Inactive state, the STAND-BY button from the PX2612 panel must be pressed or an equivalent command must be executed. This way, after the project synchronization, it goes out from the Not-Configured state and can go to Stand-by state, or go back to the Inactive state if there’s a failure

Off-Line Download of Modifications with Process Control Interruption

In the present section, it’s defined a procedure to execute an off-line download which interrupts the process control. Such situation is acceptable in specific process types and during programmed process stopping.

An off-line download from this type must be executed connecting MasterTool to the NET 1 channel of the Active PLC using the exclusive IP address from the Active PLC (CLPX Active IP). Before beginning an off-line in the Active PLC the user receives two MasterTool warnings:

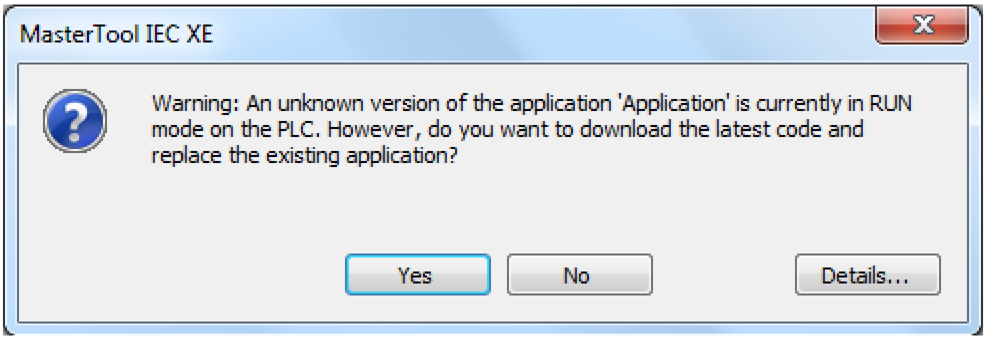


Figure 6-25. Off-Line Download Warning

By pressing *Yes*, the project is downloaded. When an off-line download is performed, the process' control is interrupted, because the project is sent to the Active PLC, which will leave the Run state, and therefore will be in the Not-Configured state.

Another important point is that if the other PLC is in Stand-by state, it must be switched to Inactive state, e.g. pressing the PX2612 INACTIVE button on this PLC. This way, the turn off of this PLC by the other PLC and its take over as Active is avoided.

ATTENTION: When the Active PLC goes out from the Run mode and goes to Not-Configured, if the other PLC was forgotten in Stand-by state, it takes over as Active and switches off the PLC which has just gone from Active to Not-Configured. In this case, thus, the off-line download can't be completed because the PLC connected to MasterTool is off.

When the off-line download finishes, it's possible to restart the PLC program execution where the application was downloaded (put in Run again). After a few seconds, this PLC takes over again the Active state.

After this PLC takes the Active state again, the other PLC can go out from the Inactive state, e.g. pressing the PX2612 STAND-BY button on it. This causes the transition of this PLC to the Not-Configured state. This PLC remains in the Not-Configured state until the automatic project synchronization finishes. Then, it goes to Starting state and back to Stand-by state afterwards.

Previous Planning for Off-Line Modifications without Process Control Interruption

The following section Previous Planning for Hot Modifications in Redundant PROFIBUS Networks describes a very important procedure which allows the off-line modifications download without interrupting the process control. Even though this procedure doesn't apply to any modification that demand off-line download, it applies to the most used modifications.

However, in order to apply this procedure, the projects must be developed with a previous planning, especially for modification that affects the PROFIBUS network. The following subsections describe such previous planning for modifications that affect the PROFIBUS network and also other modifications.

Previous Planning for Hot Modifications in Redundant PROFIBUS Networks

Among the modifications that affect a PROFIBUS network and demand an off-line download, the following are supported by the procedure which allows executing off-line downloads without interrupt the process control, if the PROFIBUS network is redundant:

- Insert a new PROFIBUS network
- Insert a new Ponto Series remote
- Insert a new I/O module in a Ponto Series remote
- Modify parameters in Ponto Series remotes or in I/O modules in Ponto Series remotes

> ATTENTION:
> It's possible to insert non-redundant remotes under a redundant PROFIBUS network, using the AL-2433 module (ProfiSwitch), as the example shown on Figure 6-1. However, such non-redundant remotes will suffer discontinuities (output deactivation) when the off-line download is executed.

Next, the planning steps that must start at the creation of a new redundant PROFIBUS network are described.

Step 1 – Plan Future Expansion of the Remotes Inserted in the PROFIBUS Network Initial Version

At first, a list must be made of the I/O modules which compose each redundant PROFIBUS remote from the Ponto Series, in the PROFIBUS network initial version. The list must have the position where each I/O module is inserted in the remote rack.

Next, the future expansion of each remote must be planned. For that, a complementary list must be made, consisting in I/O modules which might be inserted in the future. In it, the position where each I/O module might be inserted in the remote rack must be listed.

> ATTENTION:
> At the physical construction of these remotes (electric panels), it's strongly recommended to insert compatible bases with the future I/O modules in the respective positions. This way, when the I/O module insertion is necessary in this remote, there's no need for switching off the remote to insert the base. In case this detail isn't observed, it will be necessary to switch off the specific remote, as it's not possible a base hot insertion in the remote. It can be observed that the remote stopping in some cases can be tolerable, but not always.

> ATTENTION:
> The original I/O module bases must be inserted in the first remote rack positions and the future I/O modules, in the last remote rack positions.

> ATTENTION:
> It must be considered the limitations of the Ponto Series redundant remotes at constructing this list, as the PO5063V1 PROFIBUS Head Utilization Manual and PO5063V5 PROFIBUS Redundant Head, and PO5064 PROFIBUS Head Utilization Manual and PO5065 PROFIBUS Redundant Head. There are limits regarding the number of modules per remote, number of bytes per remote, current consuming per power supply, etc. These limits are verified automatically by the ProPonto. For further information, see the MT6000 MasterTool ProPonto Utilization Manual - MU299040.

Step 2 – Insert the Redundant PROFIBUS Network Initial Version in the Project

To insert the redundant PROFIBUS network initial version in the project, initially the two redundant NX5001 modules must be inserted in the rack, or use those already inserted by the redundancy wizard.

Next, each remote must be inserted in the device tree below these two NX5001, as well as the I/O modules under each remote.

Regarding the inserted I/O modules, there are two categories that must be treated differently:

- Those that are part of the PROFIBUS network initial version and will be installed immediately
- Those that will be used for future expansion

In the case of those that are part of the PROFIBUS network initial version, the module itself must be inserted in the device tree, in the planned remote correspondent position.

In the case of those that will be used for future expansion, a virtual module must be inserted in the planned correspondent position. A virtual module correspondent to a real module needs to allocate the same amount of I/O bytes than this real module. The virtual module insertion in the place of a real module avoids the real module absence diagnostics to be produced.

The following Table 6-4 shows real modules and its correspondent virtual modules:

Real Module	Correspondent Virtual Module
PO1000	PO9999 – 2 bytes input
PO1001	PO9999 – 2 bytes input
PO1002	PO9999 – 2 bytes input
PO1003	PO9999 – 2 bytes input
PO2020	PO9999 – 2 bytes output
PO2022	PO9999 – 2 bytes output

Table 6-4. Virtual Modules correspondent to the real modules

Step 3 – Allocate %I and %Q Variables Areas for the PROFIBUS Network considering Future Remote Expansion

As the NX5001, remotes and I/O modules were being inserted in the device tree in the previous step, %I and %Q variables were being allocated in three different areas:

- %I variables area for inputs
- %Q variables area for outputs
- %Q variables area for diagnostics

MasterTool executes the allocation of each one of these three variable areas in a continuous way, with no holes between them.

The initial and final address of each one of these three areas must be planned, considering the initially installed remotes in the network (see steps 1 and two), but also remotes which might be inserted in the future in this same PROFIBUS network.

At defining the initial address of each area, it's important to reserve expansion for the device which allocates addresses immediately before the beginning of this area. On the other hand, at defining the final address of each area, it's important to reserve expansion for this PROFIBUS network.

Next, an example of such planning is shown, for %I variables area for inputs:

- PROFIBUS 1 network:
  - %IB0 ... %IB499 (addresses allocated to already installed remotes)
  - %IB500 ... %IB999 (addresses allocated future remotes)
- PROFIBUS 2 network:
  - %IB1000 ... %IB1499 (addresses allocated to already installed remotes)
  - %IB1500 ... %IB1999 (addresses allocated future remotes)
- Modbus TCP server:
  - %IB2000 ... %IB2999 (addresses allocated to current mapping)
  - %IB3000 ... %IB3999 (addresses allocated to future mapping)

For the two other areas (output %Q and diagnostic %Q) similar examples could be executed.

It's possible to predict the initially allocated and future expansion areas size using the following Table 6-5 which indicates the byte quantity allocated for the 3 phases for each module:

Module	Inputs %I (bytes)	Output %Q (bytes)	Diagnostic %Q (bytes)
NX5001	4	2	86
PO5063V5	0	0	25

PO5065	0	0	25
PO9100 (one each remote)	2	2	10
PO1000	2	0	10
PO2020	0	2	10
PO9999 – 2 bytes output	0	2	10
PO9999 – 2 bytes input	2	0	10

Table 6-5. %I and %Q variables allocation for PROFIBUS network modules

Note:

Variable Allocation: Further information regarding the size and type of memory allocated for each module can be found in the PROFIBUS-DP NX5001 Master Utilization Manual.

After executing the planning for the 3 areas (initial and final address of each area), the initial addresses must be inserted in the projected started in step 2.

At first, the parameter " %Q Initial Address of Module Diagnostics Area" must be modified in the first NX5001 module, as shown on the table on the next figure. The planned initial address must be used for the diagnostic %Q variables area.

Second, the first network I/O module must be found, starting with the NX5001, which allocate %I variables for inputs. At finding it, the correspondent "Address" parameter must be altered.

Third, the first network I/O module must be found, starting with the NX5001, which allocate %Q variables for outputs. At finding it, the correspondent "Address" parameter must be altered.

ATTENTION:
At this moment it's recommended to verify the allocated address range for the 3 variable areas, verifying if the final addresses of each area are within the planned range, and if there's a good free area for expansion for new future remotes insertion.

Previous Planning for Other Hot Modifications

There are other hot modifications which, though they don't affect the PROFIBUS network, also demand off-line downloading. Next, it's presented some examples of this type of modifications supported by the procedure which allow executing modifications off-line download without interrupting the process control:

- NX5000 modules insertion (Ethernet)
- Ethernet or Serial communication I/O driver insertion
- Ethernet or Serial communication I/O driver new mapping insertion
- MainTask period modification

Some simple modifications, such as the MainTask period, don't demand any previous planning.

On the other hand, the previous examples of modifications imply the direct representation %I and %Q variables allocation for diagnostics, inputs and outputs similar to discussed in step 3 from the previous planning for hot modifications which affect the PROFIBUS network (see Step 3 – Allocate %I and %Q Variables Areas for the PROFIBUS Network considering Future Remote Expansion).

This way, at inserting the NX5000 module, or an I/O Ethernet or Serial driver, the allocation of the 3 following areas must be planned for the inserted device:

- %I variables area for inputs
- %Q variables area for outputs
- %Q variables area for diagnostics

The Step 3 – Allocate %I and %Q Variables Areas for the PROFIBUS Network considering Future Remote Expansion section shows an example of group allocation of these areas, including PROFIBUS networks and an I/O driver (Modbus TCP server).

Incompatibility of Applications

If the areas to be used in the future not be planned properly, the redundant memory areas may have to be altered, thus generating a incompatibility between the applications. This will result in only on PLC to remain in the Active state, with the other PLC remaining Inactive, without the possibility of synchronizing redundant data or application between the two PLCs.

This incompatibility is informed by the redundancy diagnostics at:
DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bApplicationIncompatible.

This diagnostic is active when the download of a new application is done to one of the PLCs, usually the Non-Active, with one of the following changes:

- Changes in the redundant memory areas, configured in the parameters of the NX4010 module
- Changes (create or remove) in the symbolic redundant variables, declared in redundant POUs or redundant GVLs

It is important to stress that, to make changes in symbolic redundant variables, the incompatibility problem will occur only if a new application download is done to one of the PLCs. In case that the modifications in symbolic redundant variables, and all the other modifications made in the project, fit into the group of Modifications which Allow On-Line Download, is possible to do an On-Line Download of Modifications with no generation of incompatibility of applications between the PLCs.

Project Update due to MasterTool IEC XE Update

The MasterTool IEC XE programming tool is under a constant enhancement process, improving its features and adding new ones. When it is necessary to update the tool in a redundant system, the used project also needs to be updated. This update is done through the Project/Project Update menu, available in the tool. After updating the project, the Off-Line download without Process Control Interruption can be done.

Updating Project from Versions Previous to 2.00 to version 2.00 or Higher

Among the MasterTool IEC XE version changes there is a special case that must be planned more carefully to avoid stopping the process. The update of a project created with a version prior to 2.00 of the MasterTool IEC XE to version 2.00 or higher causes a reconfiguration in the area allocated for the Persistent Variables object. This reconfiguration was implemented aiming at optimizing the allocation of this area. However, if this object is present and marked as redundant in a project, this reconfiguration won’t allow the data to be synchronized between the two project, always setting one of the Half-Cluster in Inoperative State.

This way, if this situation happens, the NX3030 CPU software can detect it and stop the synchronization of the Persistent Variables object data until the two Half-Clusters’ projects are the same, and, therefore, are using a project with the updated MasterTool IEC XE version. This situation won’t stop the process, but if a correct update sequence is not followed, the data of the Persistent Variables object can be restarted.

In this case, the Off-Line Download sequence below must be followed:

- Change the Half-Cluster in Active state project, unmarking the PersistentVars object inside the Redundancy Configuration object. This download must be done as an On-Line change and for this to happen another change in the project must be done, e.g. declaring a new variable inside the NonSkippedPrg POU
- After the On-Line change, it’s necessary to run the command Create Boot Application while on-line, with the PLC in Active state. This is necessary so that the application is synchronized with the Half-Cluster that passed to Non-Configured state after the download.
- Update the project from version lower than 2.00 to version 2.00 or higher through the Project\Project Update menu in MasterTool IEC XE
- Disable the Project Synchronization through the Communication\Redundancy Configuration menu

- Download the updated Project into the Half-Cluster that's in Stand-by state. A message will be displayed indicating the PersistentVars object memory area reorganization. The procedure must continue and by the end of the project download the Half-Cluster will remain in STOP with a redundancy state as Not-Configured
- Put the CPU in RUN. The Half-Cluster will change to Starting state and then to Stand-by. The Half-Cluster will synchronize its data with the one that's in Active state.
- The data from the PersistentVars object must be copied from the Active Half-Cluster to the Stand-by manually or the receipt resource must be used
- Put the Active Half-Cluster to Stand-by. With this action, the other Half-Cluster will go to Active.
- Enable the Project Synchronization through the Communication\Redundancy Configuration menu. After this, the Half-Cluster in Stand-by sate will go into Not-Configured state and will receive the project from the Half-Cluster in Active state. By the end of this process, the Half-Cluster state will go to Starting and then back to Stand-by.
- Change the Project of the Half-Cluster in Active state marking the PersistentVars object inside the Redundancy Configuration object. This download must be done as an On-Line change, and for this to happen, another change in the project must be done, e.g. removing the variable declared at the beginning of this process.
- After this, the Half-Cluster that was in Stand-by will pass to Non-Configured and will receive the Project from the Half-Cluster in Active state. By the end of this process the Half-Cluster state will change to Starting and then back to Stand-by.

Exploring the Redundancy for Off-Line downloading of Modifications without Interruption of the Process control

In the Off-Line and On-Line Modifications Download section, it was informed that some modifications demand an off-line download in the PLC.

In these cases the user has the option to interrupt the process control, according to the procedure defined in the Off-Line Download of Modifications with Process Control Interruption section. For that, usually it's necessary to previously program a process stopping, what isn't always possible at the moment the modification is needed.

Fortunately, the PLCs redundancy, and in some cases, the PROFIBUS network redundancy make possible to execute the off-line download without interrupting the process control, for most of the modifications needed. For reaching this objective it's necessary to follow carefully a procedure, which the steps are described in the following sub-sections.

Step 1 – Verify Basic Requirements Attending

For the off-line downloading with no interruption of the process control to be possible, the following basic requirements must be attended:

- The original project must have been created according to the recommendations of the Previous Planning for Off-Line Modifications without Process Control Interruption section
- The PLC must be redundant
- In case the modification affects the PROFIBUS network, it's necessary this network to be redundant. Such modifications may be:
    - New remotes insertion
    - I/O modules insertion in existent remotes, in previously reserved positions for correspondent virtual modules. For the remote not have to be switched off, there's the need of a base compatible with the new I/O module in the position reserved for it
    - Parameters modifications in remotes or existent I/O modules
- Both PLCs projects must be equalized and the Redundant Data Synchronization and Redundant Forcing List services must be working properly with no failure diagnostics. It can be stated these conditions are satisfied when there's a PLC in Active state and another in Stand-by state. In case the Non-Active PLC isn't in Stand-by state, the following diagnostics can be observed:

- DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bRedDataSync = TRUE, indicates the success of the Redundant Data Synchronization service
- DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bRedForceSync = TRUE, indicates the success of the Redundant Forcing List service
- DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.dwApplicationCRC = DG_NX4010.tRedundancy.RedDgnRem.dwApplicationCRC, indicates both PLCs projects are equal

Step 2 – Don't Download in Group Modifications which can be downloaded On-Line

Modifications which can be downloaded on-line must not be downloaded together with modifications which must be downloaded off-line without the process control interruption. When these two kinds of modifications are needed, they must always be loaded separately.

For the current procedure to be successful, it's absolutely necessary the modifications executed to don't cause any changes in the structure of the redundant variables exchanged between the Active and Non-Active PLC, through the Redundant Data Synchronization and Redundant Forcing List services. These two services must continue to working properly even while there are temporary differences between the PLCs.

The modifications that must be loaded off-line, and supported by this procedure do not affect the structure of redundant variables.

However, some modifications which can be loaded on-line can change the structure of redundant variables, e.g.:

- Insertion of symbolic variables (redundant or not) within a POU or GVL existing or in a new POU or GVL
- Removal of symbolic variables (redundant or not) within a POU or within existing GVL. The removal of a POU or GVL can also involve the removal of symbolic variables
- Change in size/structure of symbolic variables (redundant or not) in an existing POU or GVL

Step 3 – Previous Project Backup

Before editing the modifications that must be loaded off-line without interrupting the process control, for safety reasons a backup of the project previous version must be run. It may be necessary to reinstall the previous version in case an error is committed during this procedure executing.

ATTENTION: The backup recommendation for all loaded versions in the PLCs may not be followed only in this specific procedure. It must be a usual practice.

Step 4 – Cares in Editing the Off-line Downloaded Modifications

The off-line downloaded modifications through this procedure, usually, are the following:

- Insertion of new devices in the devices tree
- Property or parameter change in devices existing in the devices tree

Such devices are normally the following:

- Modules such as PROFIBUS master (NX5001) or Ethernet modules (NX5000)
- Ponto Series PROFIBUS remotes
- I/O modules inside Ponto Series PROFIBUS remotes
- Modbus communication I/O drivers
- Mapping of Modbus communication drivers

The following cares must be taken at editing these project modifications:

- If a device existed in the previous project version, and continues existing in the modified version, the %I and %Q variables allocated for it must remain the same (command, diagnostic, inputs and outputs). Care must be taken for the inserted modifications don't change such allocations
- If a device was inserted in the modified project version, the %I and %Q variables allocated for it must not be allocated in the previous project version (command, diagnostic, inputs and outputs)

Step 5 – Inactive PLC Project Synchronism Disabling

In the procedures described in the On-Line Download of Modifications and Off-Line Download of Modifications with Process Control Interruption sections, the project is automatically synchronized from the Active PLC to the Non-Active PLC.

However, during the procedure of off-line downloading without process control interruption, the project synchronism must be temporarily disabled.

The Project synchronization disabling is explained in the section Project Synchronization Disabling and must be executed in the Non-Active PLC.

Step 6 – Physical Modifications Executing

At this moment, the physical modifications can be executed, such as:

- Install a new NX5000 module. This can be done through a module hot-insertion in each half-cluster rack, then connecting it to the Ethernet network
- Install a new redundant PROFIBUS network. The NX5001 can be hot-inserted in each half-cluster rack. Then, the redundant PROFIBUS network can be connected to them
- Install a new Ponto Series redundant remote. In this case, a remote head must be installed at a time, e.g. first in the network B and then in the network A:
    - To install the head in the network B, it may be necessary to open the cable or the contacts, thus perturbing the communication with the other heads already installed in the network B. Before doing that, all the operating active heads must be placed in the network A and the operating reserve heads in the network B
    - To install the head in the network A, it may be necessary to open the cable or the contacts, thus perturbing the communication with the other heads already installed in the network A. Before doing that, all the operating active heads must be placed in the network B and the operating reserve heads in the network A
- Install an I/O module in a base previously reserved for it, in an existent remote

Step 7 – Download the Off-Line Modifications in the Non-Active PLC

At first, MasterTool must be connected to the Non-Active PLC (see MasterTool Connection with a NX3030 CPU from a Redundant PLC section).

Next, the off-line modifications must be downloaded. At doing it, the Non-Active PLC application is automatically interrupted (goes out of the Run mode).

Step 8 – Set the Non-Active PLC Back to Run Mode to make go back to Stand-by State

The off-line load being finished, the Non-Active PLC can go back to Run mode.

A few seconds later, the Non-Active PLC must assume the Stand-by state.

In case the PLC doesn't assume the Stand-by state, the following problems may have caused this effect:

- The modifications executed changed the redundant variables structure, which prevents the correct execution of the Redundant Data Synchronization service. This can be verified through DG_NX4010.tRedundancy.RedDgnLoc. sGeneral_Diag.bRedDataSync (0 = failure) diagnostics in the Non-Active PLC. In this case, the modifications must be undone, recovering the previous project backup and restarting this procedure

- Other problems may eventually prevent the transition to the Stand-by state, even though this is unexpected. In this case, the diagnostics and the redundancy log must be observed

In case the PLC has assumed the Stand-by state, it's recommendable to verify if the projects are different between the Active and the Non-Active PLC. This can be made comparing the diagnostics DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.dwApplicationCRC and DG_NX4010.tRedundancy.RedDgnRem.dwApplicationCRC in the Non-Active PLC (the CRCs must be different).

In case both projects are equal in the PLCs, it's possible that the project synchronism disabling (step 5) has not being properly executed. This can be verified through the diagnostic DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable which must be true in the Non-Active PLC. If it isn't true, the procedure must be returned to step 5.

Step 9 – Execute Switchover between Active and Stand-by PLCs

A switchover between the PLCs must be executed, e.g. pressing the STAND-BY button on the Active PLC. The Stand-by PLC, which has a new project with the modifications, takes over as Active. The Active PLC, which has the old project, takes over as Stand-by.

Step 10 – Projects Synchronism Enabling in the Active PLC

In the step 5, the project synchronism was disabled in the Non-Active PLC. It can be observed this PLC is now in Active state.

In this step, the project synchronism must be enabled again in this PLC. The screen and methodology used for it were described in the section Project Synchronization Disabling. But this time we need to select the "Enable" option from the combo-box. MasterTool must be connected to the Active PLC (see MasterTool Connection with a NX3030 CPU from a Redundant PLC section).

After enabling the project synchronism in the Active PLC, the user must verify if this command was successful, verifying if DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag. bProjectSyncDisable= 0 in the Active PLC.

As soon as the project synchronism is enabled again, the following action sequence is expected:

- The Non-Active PLC (Stand-by state), which already knows the different between both projects; goes out from the Stand-by state and goes to the Not-Configured state
- The modified project (new) is copied from the Active PLC to the other, temporarily in Not-Configured state
- As soon as the projects are synchronized again, the Not-Configured PLC goes to Starting state and then it's supposed to go back to Stand-by state

Step 11 – Optional Reorganization of PLC and PROFIBUS Networks in Active State

At the end of the procedure, for standardization or organization reasons, the user may execute a switchover for the PLCA assumes as Active, and for all remotes PROFIBUS heads in Active state are in the network A.

Maintenance

Modules Hot Swapping in a Redundant PLC

In case of failure in a module from one of the PLCs (PLCA and PLCB), the module hot swapping may be necessary, without interrupt the process control. For that, the following steps must be followed:

- Verify if the half-cluster which won't be modified is in Active or Stand-by state, allowing it to take the process control
- To put the half-cluster having its module changed in Inactive state, through the Redundancy Control Panel PX2612 or the Redundancy Commands
- Execute the necessary exchanges in the Inactive half-cluster, as indicated in the CPU Configuration – General Parameters –How to do the Hot Swap chapter
- To put the half-cluster back to Stand-by or Active state, according to necessity

MasterTool Warning Messages

When MasterTool is with a redundant project open, or when it's connected to a NX3030 CPU identified as PLCA or PLCB, some special warning messages may occur, as described in the following sub-sections.

Blocking of Redundant or Non-Redundant Project Download

MasterTool doesn't allow the download of a redundant project, unless the CPU is NX3030 and is identified as PLCA or PLCB (see NX3030 CPU Identification section).

On the other hand, MasterTool doesn't allow the download of a non-redundant project in a NX3030 CPU identified as PLCA or PLCB.

In case any of these illegal actions is tried, MasterTool warns with a correspondent message.

Warnings before Commands which may stop the Active PLC

Some commands, as the following, may stop a PLC:

- Offline load after Online / Login
- Debug / Stop
- Debug / New Breakpoint
- Online / Reset (warm, cold, origin)

In case MasterTool is logged to the Active PLC, and one of these commands is tried, before sending it to the Active PLC, MasterTool sends the following message and waits for authorization:

"If the other PLC is in Stand-by State, it will assume as Active and turn-off this PLC. If not, this won't happen, but the automated process will no longer be controlled."

Warning before Logging to the Non-Active PLC

In normal circumstances, it isn't usual MasterTool to connect to the Non-Active PLC. This way, when there's a try to execute this type of command, MasterTool sends the following warning:

"You are logging in to a Non-Active PLC, and this is not usual. Are you sure you want to execute this command?"

On the other hand, there are circumstances (not so usual) in which it's necessary to login in the Non-Active PLC, and in these cases the user must authorize the login. Such circumstances may occur, e.g.:

- For initial configurations, as described in Initial Downloading of a Redundant Project section

- For downloading off-line a different project in the Non-Active PLC, as described in the Exploring the Redundancy for Off-Line downloading of Modifications without Interruption of the Process control section
- For monitoring or forcing the non-redundant variables in the Non-Active PLC

Redundancy Diagnostics on the NX3030 CPU Graphic Display

Many diagnostics related to redundancy can be observed on the NX3030 CPU display.

CPU Redundancy State

The PLC redundancy state, described in Redundant CPU States is seen in the three initial characters on the main screen second line, as shown in the chapter Graphic Display. The display screen is presented on initialization and again a few seconds after the navigation is finished (without pressing the NX3030 CPU button).

Screens below the REDUNDANCY Menu

There’s a menu called REDUNDANCY, where below it there are several screens. The description and access of theses screens are available in the Configuration – CPU’s Informative and Configuration Menu chapter.

Redundancy Diagnostics Structure

The NX4010 module diagnostics area is mapped over direct representation %Q variables, and defined symbolic through the AT directive, in the GVL Diagnostics.

This section is divided in two parts:

- DG_NX4010.tGeneral: General diagnostics for NX4010 operation. They are described in the Redundancy Link Module Technical Features – CE114900
- DG_NX4010.tRedundancy: Redundancy specific diagnostics which are described within the chapter. This item is divided in other 6 data structures:
  - RedDgnLoc: Has redundancy diagnostics of the local PLC (connected), e.g. the PLC redundancy state. This section is described in Redundancy Diagnostics
  - RedDgnRem: It’s a copy from the other PLC RedDgnLoc, received through Synchronism channels NETA / NETB. This way the local PLC has access to the remote PLC diagnostics. This section is described in Redundancy Diagnostics
  - RedCmdLoc: Has redundancy commands generated in this PLC (local), for instance, through write commands from a SCADA system or generated in POUs in this PLC (ActivePrg or NonSkippedPrg). This section is described in Redundancy Commands
  - RedCmdRem: It’s a copy from the other PLC (remote) RedCmdLoc, received through Synchronism channels NETA / NETB. This section is described in Redundancy Commands
  - RedUsrLoc: Used to allow the user to exchange information between PLCA and PLCB. This section is described in User Information Exchanged between PLCA and PLCB
  - RedUsrRem: Used to allow the user to exchange information between PLCA and PLCB. This section is described in User Information Exchanged between PLCA and PLCB

It is important to stress that the redundancy diagnostics structures are refreshed only when occurs, with success, a new data synchronization. Therefore, until a new synchronization doesn’t occur, the diagnostics will remain with the last read value.

Furthermore, the diagnostics structures from the remote PLC are read only, that is, values written to these structures will be overwritten in the next synchronization. Thus, is not possible to use the RedCmdRem structure to execute a command in the remote PLC. Always use the structure RedCmdLoc to execute commands.

Redundancy Diagnostics

The Redundancy Diagnostics may have several uses, such as:

- They can be consulted in order to verify the existence of a problem that needs to be solved
- Every time there are variations on them, such variations are inserted as events in the "Redundancy Event Log". Consulting the history sequence of such events, a switchover cause may be discovered, for instance
- They can be referenced in the user application (ActivePrg or NonSkippedPrg). E.g. the PLC state can be tested and in case it's not active, a MODBUS RTU serial master I/O driver can be disabled, in NonSkippedPrg

ATTENTION:
The _NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync (defined next) must be tested to verify if the data structure RedDgnRem was successfully read from the remote PLC in the last MainTask cycle. In case this diagnostic value is 0 (false), this means the data structure RedDgnRem wasn't successfully read from the remote PLC, thus the RedDgnRem values may be invalid or obsolete.

As RedDgnRem is a copy from the other PLC RedDgnLoc, it can be concluded the two structures have the same format. These are divided in other four substructures:

- sGeneral_Diag: Redundancy general diagnostics
- sNETA_Diag: NETA synchronism channel diagnostics
- sNETB_Diag: NETB synchronism channel diagnostics
- sNET_Stat: Common statistics for the synchronism channels NETA and NETB, for failure and success counting in the synchronization services

The "sGeneral_Diag" substructure has the following fields for redundancy general diagnostics:

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	Bit		
%QB(n+4)	0	bConfigDone	TRUE – The configuration process, executed in the Not-Configured state, has finished.
			FALSE – The configuration process, executed in the Not-Configured state, hasn't finished yet or wasn't executed.
	1	bConfigError	TRUE – the configuration process, executed in the Not-Configured state, has finished with errors. It's a system error, normally not expected. Get in contact with ALTUS support to report it. Also inform the ConfigErrorCode diagnostic value for the ALTUS support.
			FALSE – The configuration process has finished successfully or wasn't executed.
	2	bTooManyRedAreas	TRUE – The number of redundant areas exceeded the maximum allowed. It's a system error, normally not expected. Get in contact with ALTUS support to report it.
			FALSE – The number of redundant areas is within the expected.
	3	bTemporaryBufferTooSmall	TRUE – Intermediate data structure with insufficient size. It's a system error, normally not expected. Get in contact with ALTUS support to report it.
			FALSE – Intermediate data structure is within the expected.
	4	bExchangeSync	TRUE – The Diagnostic and Commands Exchange synchronization service was executed successfully in this MainTask cycle.
			FALSE – The RedDgnRem structure has obsolete or invalid values, as it wasn't read from

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	Bit		
			the other PLC (remote) in this cycle.
	5	bRedDataSync	TRUE – The Redundant Data Synchronization service was executed successfully in this MainTask cycle.
			FALSE – The Redundant Data Synchronization service wasn't executed successfully in this MainTask cycle.
	6	bRedForceSync	TRUE – the Redundant Forcing List Synchronization service was executed successfully in this MainTask cycle.
			FALSE – the Redundant Forcing List Synchronization service wasn't executed successfully in this MainTask cycle.
	7	bApplicationIncompatible	TRUE – The application isn't compatible between the PLCs. Was done a new application download with one of the following changes: - Changes in redundant memory area; - Changes in symbolic redundant variables; Whereas this diagnostic be TRUE, one of the PLCs will stay in Inactive state until the same application be present in the two PLCs. This implies in reload the old application or download the new application to both PLCs. More information about how to proceed can be found in section Redundant CPU Program Downloading.
			FALSE – The application is compatible between the PLCs.
%QB(n+5)	0	Reserved	Reserved bit
	1	bProjectSyncDisable	TRUE – The project application and project archive will not be synchronized between the PLCs. It's a copy from the non-volatile variable used to enabling or disabling the project synchronization, as described in the Project Synchronization Disabling section. The project synchronization is disabled in the local or remote PLC. This way, it's enough to execute the disabling command in one PLC for the project synchronization to be disabled. The enabling and disabling project synchronization commands are described in the Project Synchronization Disabling section.
			FALSE – The project application and project archive will be synchronized between the PLCs.
	2	bIncompatibleFirmware	TRUE – Firmware version is incompatible between this CPU and the remote one
			FALSE – Firmware version is compatible between this CPU and the remote one
	3	bApplicationProjectDiff	TRUE – The project application between this CPU and the remote one is different.
			FALSE – The project application between this CPU and the remote one is equal.
	4	bProjectArchiveDiff	TRUE – The project archive between this CPU and the remote one is different
			FALSE – The project archive between this CPU and the remote one is equal
	5	bOnlineChangeApply	TRUE – Some alteration was done online in the application and it hasn't been synchronized yet with the stand-by PLC.
			FALSE – There wasn't alterations online in the application or these have been synchronized already with the stand-by PLC.
	6	bFailedRED	TRUE – Failure in the NX4010 module. The

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	Bit		
			NX3030 CPU can't communicate with this module through bus, or there's a failure in the NX4010 microprocessor.
			FALSE – The NX4010 module is working properly.
	7	bFailedPBUS1A	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 1 A network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 1 A network.
%QB(n+6)	0	bFailedPBUS1B	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 1 B network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via.
			FALSE – There aren't failures in the PROFIBUS 1 B network.
	1	bFailurePROFIBUS_1	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 1 network. In case the PROFIBUS 1 network is redundant, FailurePROFIBUS_1 results from a AND logic between FailedPBUS1A and FailedPBUS1B. In case the PROFIBUS 1 network isn't redundant, FailurePROFIBUS_1 is a copy from FailedPBUS1A.
			FALSE – There aren't failures in the PROFIBUS network.
	2	bFailedPBUS2A	TRUE – this PLC can't communicate in the master state (active or passive) in the PROFIBUS 2 A network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 2 A network.
	3	bFailedPBUS2B	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 2 B network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 2 B network.
	4	bFailurePROFIBUS_2	TRUE – this PLC can't communicate in the master state (active or passive) in the PROFIBUS 2 network. In case the PROFIBUS 2

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	Bit		
			network is redundant, FailurePROFIBUS_2 results from a AND logic between FailedPBUS2A and FailedPBUS2B. In case the PROFIBUS 2 network isn't redundant, FailurePROFIBUS_2 is a copy from FailedPBUS2A.
			FALSE – There aren't failures in the PROFIBUS 2 network.
	5	bPROFIBUSVitalFailureAny	TRUE – This PLC can't communicate in the master state (active or passive) with at least one of the PROFIBUS networks configured in vital failure mode.
			FALSE – There aren't failures in the PROFIBUS networks configured in vital failure.
	6	bPROFIBUSVitalFailureAll	TRUE – This PLC can't communicate in the master state (active or passive) with all the PROFIBUS networks configured in vital failure mode.
			FALSE – There aren't failures in the PROFIBUS networks configured in vital failure.
	7	bTurnOffOtherPLC_Normal	TRUE – This PLC is closing the PX2612 relay to keep the other PLC off in normal conditions and not due to PX2612 panel test.
			FALSE – The PX2612 relay is on (bTurnOffOtherPLC_TestMode) or off.
%QB(n+7)	0	bTurnOffOtherPLC_TestMode	TRUE – this PLC is closing the PX2612 relay to keep the other PLC off due to PX2612 panel test mode.
			FALSE – The PX2612 relay is on (bTurnOffOtherPLC_Normal) or off.
	1	bActiveLED	TRUE – The PX2612 LED ACTIVE is on.
			FALSE – The PX2612 LED ACTIVE is blinking (bBlinkActiveLED) or off.
	2	bBlinkActiveLED	TRUE – The PX2612 LED ACTIVE is blinking.
			FALSE – The PX2612 ACTIVE is on (bActiveLEDl) or off.
	3	bStandbyLED	TRUE – The PX2612 LED STAND-BY is on.
			FALSE – The PX2612 LED ACTIVE is blinking (bBlinkStandbyLED) or off.
	4	bBlinkStandbyLED	TRUE – The PX2612 LED STAND-BY is blinking.
			FALSE – The PX2612 LED STAND-BY is on (bStandbyLED) or off.
	5	bInactiveLED	TRUE – The PX2612 LED INACTIVE is on.
			FALSE – The PX2612 LED INACTIVE is off or blinking (bBlinkInactiveLED).
	6	bBlinkInactiveLED	TRUE – The PX2612 LED INACTIVE is blinking.
			FALSE – The PX2612 LED INACTIVE is on (bInactiveLED) or off.
	7	bRedPanelTestMode	TRUE – The PX2612 panel is in test mode.
			FALSE – The PX2612 panel is in normal mode.
%QB(n+8)	-	ePLC_ID	This diagnostics inform this PLC identification: - 0 = non-redundant - 2 = PLCA - 3 = PLCB It's a copy from the non-volatile variable used to identify the PLC, as described in the NX3030 CPU Identification section. In the Initial Downloading of a Redundant Project section MasterTool command used to write on this non-volatile variable is described.

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
%QB(n+9)	-	eRedState	informs the redundancy state of this PLC: - Not-Configured = 0 - Starting = 2 - Stand-by = 3 - Active = 4 - Inactive = 5
%QB(n+10)	-	ePreviousRedState	Previous RedState value before the data transition.
%QW(n+11)	-	wRedStateDuration	Measures for how long (milliseconds) the current redundancy state has been assumed. This time stops incrementing when reaches 65535ms.
%QW(n+13)	-	wConfigErrorCode	Error code discovered during the configuration process in the Not-Configured state. See ConfigError diagnostics described previously.
%QD(n+15)	-	dwApplicationCRC	32 bits applicative project CRC, used to detect differences between the applicative projects of the 2 PLCs.
%QD(n+19)	-	dwArchiveCRC	32 bits project archive CRC, used to detect differences between the project archive of the 2 PLCs.
%QD(n+23)	-	dwFirmwareVersion	This PLC firmware version, used to verify the compatibility between both PLC firmware.
%QD(n+27)	-	dwIECTimer	The IEC Timer synchronization is necessary for the bump-less operation of some function block as TON and TOF. Through this diagnostic the IEC Timer from the Active PLC is received and updated in the Non-Active PLC, since the Diagnostics and Commands Exchange service has been executed successfully. The counting starts at 0 and is incremented up to 4294967295. After counting overflow restarts with 0.
%QW(n+31)	-	wCycleCounter	16 bits counter, used as sequence auxiliary information in the Redundancy Event Log. In the Active PLC, it's incremented each MainTask cycle. In the Non-Active PLC, receives a copy of the value existent in the Active PLC, since the Diagnostics and Commands Exchange service has been executed successfully. The counting starts at 0 and is incremented up to 65535. After counting overflow restarts with 0.

Table 6-6. Redundancy General Diagnostics

Notes:

Diagnostics Structures Visualization: The diagnostics structures added to the project can be visualized accessing the “Library Manager” from the tree view in the MasterTool IEC XE window. With that it’s possible to visualize all data types defined in the structure.

Direct Representation Variables: The “n” represents the configured value in the NX4010 module, through MasterTool IEC XE software, as a Diagnostics Initial Address in %Q. This definition is true for all diagnostics structure.

AT Directive: The AT directive is a word reserved in the programming software which is connected to a variable address. In case of a NX4010 module the DG_NX4010 variable is related to the module diagnostics initial address.

DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync: When this diagnostic variable is with value FALSE, is not possible to define the state of some other diagnostics, such as bIncompatibleFirmware, bApplicationProjectDiff and bProjectArchiveDiff. There will not represent the correct value because they depend on the correct functioning of the communication between the two CPUs, so that information can be correctly generated.

The “sNETA_Diag” substructure has the following fields for NETA synchronism channels diagnostics:

Direct Representation Variable		AT Variable DG_NX4010.tRedundancy.RedDgnLoc.sNETA_Diag.*	Description
Variable	Bit		
%QB(n+33)	0	bGeneralFailure	TRUE – The synchronism channel has some type of failure. The 3 next diagnostics will indicate the specific failure.
			FALSE – The synchronism channel is working properly.
	1	bInternalFailure	TRUE – The detected failure has its cause within this PLC. Such failures are treated in a special way.
			FALSE – The NX4010 module is working properly.
	2	bLinkDownFailure	TRUE – The AL-2319A cable is disconnected from the NX4010 module or broken in one of the PLCs.
			FALSE – The AL-2319A cable is connected to the NX4010 module.
	3	bTimeoutFailure	TRUE – This failure is reported in case a synchronization service hasn’t been finished successfully within a specific time out and failures from the type bInternalFailure or bLinkDownFailure haven’t been found to justify that.
			FALSE – The NX4010 module is working properly.
	4 a 7	bReserved[4..7]	Reserved.

Table 6-7. NETA Interface Specific Diagnostics

The “sNETB_Diag” substructure has the following fields for NETB synchronism channels diagnostics:

Direct Representation Variable		AT Variable DG_NX4010.tRedundancy.RedDgnLoc.sNETB_Diag.*	Description
Variable	Bit		
%QB(n+34)	0	bGeneralFailure	TRUE – The synchronism channel has some type of failure. The 3 next diagnostics will indicate the specific failure.
			FALSE – The synchronism channel is working properly.
	1	bInternalFailure	TRUE – The detected failure has its cause within this PLC. Such failures are treated in a special way.
			FALSE – The NX4010 module is working properly.
	2	bLinkDownFailure	TRUE – The AL-2319A cable is disconnected from the NX4010 module or broken in one of the PLCs.
			FALSE – The AL-2319A cable is connected to the NX4010 module.
	3	bTimeoutFailure	TRUE – This failure is reported in case a synchronization service hasn’t been finished successfully within a specific time out and failures from the type bInternalFailure or bLinkDownFailure haven’t been found to justify that.
			FALSE – The NX4010 module is working properly.
	4 a 7	bReserved[4..7]	Reserved.

Table 6-8. NETB Interface Specific Diagnostics

The "sNET_Stat" substructure has service success and failure statistics. The local and remote PLCs statistics can be restarted through commands:

```
//Local PLC
DG_NX4010.tRedundancy.RedCmdLoc.bResetNETStatisticsLocal := TRUE;
//Remote PLC
DG_NX4010.tRedundancy.RedCmdLoc.bResetNETStatisticsRemote := TRUE;
```

The substructure has the following counters:

Direct Representation Variable	AT Variable DG_NX4010.tRedundancy. RedDgnLoc.sNET_Stat.*	Description
%QW(n+35)	wSuccessExchDgCmdSync	Success counting of the Diagnostics and Commands service (0 to 65535)
%QW(n+37)	wFailedExchDgCmdSync	Failure counting of the Diagnostics and Commands service (0 to 65535)
%QW(n+39)	wSuccessRedDataSync	Success counting of the Redundant Data Synchronization service (0 to 65535)
%QW(n+41)	wFailedRedDataSync	Failure counting of the Redundant Data Synchronization service (0 to 65535)
%QW(n+43)	wSuccessRedForceSync	Success counting of the Redundant Forcing List Synchronization service (0 to 65535)
%QW(n+45)	wFailedRedForceSync	Failure counting of the Redundant Forcing List Synchronization service (0 to 65535)
%QB(n+47)	byReserved[1..8]	Reserved

Table 6-9. Interface Specific Diagnostics

Note:

Counters: All counters return to zero when its limit value is exceeded.

Redundancy Commands

The structure command fields RedCmdLoc and RedCmdRem have a suffix which can be Local or Remote. E.g. there are the command fields StandbyLocal that StandbyRemote that have equivalent effect to the PX2612 panel STAND-BY button.

A command with Local suffix generated in RedCmdLoc must be executed in the local PLC (local).

On the other hand, a command with Remote suffix generated in RedCmdLoc will be executed in another PLC (remote). This works as the following:

- The remote PLC, each MainTask cycle, receives the RecCmdLoc copy from the local PLC through NETA / NETB, and this copy is called RedCmdRem in it
- The remote PLC only executes the RedCmdRem commands with the Remote suffix

Example 1: if the local PLC is in Active state, and it's desired to switch it to the Stand-by state, the DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal bit can be turned on in it.

Example 2: if the remote PLC is in Active state, and it's desired to switch it to the Stand-by state, the DG_NX4010.tRedundancy.RedCmdLoc.bStandbyRemote bit can be turned on in the local PLC. This may be useful, for instance, if the communication of a SCADA system is temporarily unavailable with the remote PLC. In this case, the command is written by the SCADA in the local PLC that retransmits to the remote PLC through NETA / NETB.

ATTENTION:
If the DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync diagnostic is indicating a Diagnostics and Commands Exchange service failure, a command with Remote suffix isn't allowed to be transmitted to the remote PLC, thus, won't be executed.

To trigger a command, the RedCmdLoc correspondent bit must be turned on. This can be done through a SCADA system, executing writing via MasterTool or even turning the bit on inside a POU as ActivePrg or NonSkippedPrg.

The user doesn't need to worry with the command bit deactivating, which is automatically done by the redundancy manager:

- In case of commands executed in the local PLC (RedCmdLoc + commands with Local suffix), the bit is turned off as soon as the command is seen and executed
- In case of commands executed in the remote PLC (RedCmdRem + commands with Remote suffix):
    - In the remote PLC, the command is executed when the redundancy manager sees an up-going edge in the command bit
    - In the local PLC where the command was generated, the bit is turned off automatically in the next MainTask cycle

ATTENTION:
There are two command bits which normally must be turned off by the user: DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal and _NX4010.tRedundancy. RedCmdLoc.bTestRelayLocal. Further details regarding these commands are described ahead in this section. In case the user forgets to turn them off, there are automatic mechanisms which are supposed to do it instead.

It's important to stress that any command activating or deactivating are registered in the Redundancy Event Log, which is important for the history analysis, e.g. to determine a switchover cause.

Next, the RedCmdLoc and RedCmdRem structure fields are defined:

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedCmdLoc.*	Description
Variable	**Bit**		
%QB(n+55)	0	bButtonTurnOnLocal	TRUE – It's a processed copy from the TURN ON PLCx button written on the PX2612 panel. This bit is activated 1 second after the button pressing and deactivated immediately at its releasing. It's important to stress that this bit will be activated when the TURN ON button on the remote PLC is pressed, as this type of command is always sent by the remote PLC.
			FALSE – The button TURN ON PLCx isn't pressed.
	1	bButtonStandbyLocal	TRUE – It's a processed copy from the STAND BY button written on the PX2612 panel. This bit is activated 1 second after the button pressing and deactivated immediately at its releasing.
			FALSE – The button STAND BY isn't pressed.
	2	bButtonInactiveLocal	TRUE – It's a processed copy from the INACTIVE button written on the PX2612 panel. This bit is activated 1 second after the button pressing and deactivated immediately at its releasing.
			FALSE – The button INACTIVE isn't pressed.
	3	bAutoConfigLocal	TRUE – This diagnostics inform an automatic configuration request, necessary to let the

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedCmdLoc.*	Description
Variable	Bit		
			Not-Configured state in some situations.
			FALSE – Automatic configuration request disabled.
	4	bTurnOnLocal	TRUE – This command produces an equivalent action to the TURN ON PLCX button on the PX2612 in the local PLC.
			FALSE – The TURN ON PLCx button on the local PLC isn't pressed.
	5	bStandbyLocal	TRUE – This command produces an equivalent action to the STAND BY button on the PX2612 in the local PLC.
			FALSE – The STAND BY button on the local PLC isn't pressed.
	6	bInactiveLocal	TRUE – This command produces an equivalent action to the INACTIVE button on the PX2612 in the local PLC.
			FALSE – The INACTIVE button on the local PLC isn't pressed.
	7	bResetNETStatisticsLocal	TRUE – This command resets the NETA / NETB statistics (see substructure SNET_Stat in RedDgnLoc and RedDgnRem). Such statistics are failure and success counters in synchronization services.
			FALSE – The reset commands for the NETA / NETB statistics in the local PLC wasn't activated.
%QB(n+56)	0	bTestModeLocal	TRUE – This command puts the PX2612 panel in test mode, allowing its components to be tested (LEDs, relays and buttons), as explained in PX2612 Panel Test section. The PX2612 test mode is only accepted when this bit is on both PLCs. Therefore, for the PX2612 to be in test mode, the PLC verifies if RedCmdLoc.TestModeLocal and RedCmdRem.TestModeLocal are both on. The RedDgnLoc.RedPanelTestMode diagnostic is turned on to inform that the PX2612 is really in test mode. Normally the user must turn off the TestModeLocal bit on both PLCs as soon as the PX2612 tests are concluded, but in case he forgets to do that, the bit will be turned off automatically 15 minutes after being turned on.
			FALSE – The command which puts the PX2612 panel in test mode is deactivated.
	1	bTestRelayLocal	TRUE – This command is used to test the PX2612 NO relay and, consequently, the external NC relay too, used to, eventually, turn off the other PLC. This command is only accepted while the PX2612 is in test mode, being automatically switched off and ignored if the PX2612 isn't in this mode. Normally, the user must turn off the TestRelayLocal bit as soon as the relay test is concluded, but if it's forgotten, the bit is turned off as soon as the test mode is finished. It's important to stress this command is only accepted in the Active PLC, to avoid the Non-Active PLC to switch it off.
			FALSE – The command used to test the PX2612 NO relay is deactivated.
	2	bStandbyRemote	TRUE – This command produces an equivalent action to the STAND BY button on the PX2612 in the remote PLC.
			FALSE – The STAND BY button on the remote PLC isn't pressed.

Direct Representation Variable		AT variable DG_NX4010.tRedundancy.RedCmdLoc.*	Description
Variable	Bit		
	3	bInactiveRemote	TRUE – This command produces an equivalent action to the INACTIVE button on the PX2612 in the remote PLC.
			FALSE – The INACTIVE button on the remote PLC isn't pressed.
	4	bResetNETStatisticsRemote	TRUE – This command produces an equivalent action to the ResetNETStatisticsLocal button on the PX2612 in the remote PLC
			FALSE – The reset commands for the NETA / NETB statistics in the remote PLC wasn't activated.
	5 a 7	bReserved[5..7]	Reserved.

Table 6-10. Redundancy Commands

User Information Exchanged between PLCA and PLCB

The Diagnostics and Commands Exchange Synchronization service, in each MainTask cycle, exchange the following data structures between both PLCs, using the NETA / NETB synchronism channels:

- Redundancy Diagnostics (RedDgnLoc and RedDgnRem), already described in the Redundancy Diagnostics Structure section
- Redundancy Commands (RedCmdLoc and RedCmdRem), already described in the Redundancy Commands section
- User Information Exchanged between PLCA and PLCB (RedUsrLoc and RedUsrRem), which are described in this section

The RedUsrLoc and RedUsrRem structures are simply a 128 bytes array, which utilization can be freely defined by the user. They allow the user to transfer, each cycle, 128 bytes of information from PLCA to PLCB, and other 128 bytes from PLCB to PLCA.

RedUsrRem is a copy from the other PLC RedUsrLoc, received through NETA / NETB. A specific PLC writes information on RedUsrLoc, which are read in the RedUsrRem of the other PLC.

These data structures have many utilities. E.g. supposing the SCADA system is connected only to the Active PLC and it's desired to visualize some information from the Non-Active PLC. The Non-Active PLC can put this information in these data structures. Among such information, for instance, might be some not mapped diagnostics in RedDgnLoc and RedDgnRem.

Modbus Diagnostics used at Redundancy

To check for failure in all MODBUS Server configured in a MODBUS Client instance, there is a specific diagnosis in each MODBUS Client instance configured. Table 6-11 displays the diagnostics for this type of failure in a MODBUS Client instance called MODBUS_Symbol_Client.

Variable DG_MODBUS_Symbol_Client.tDiag.*	Description
bAllDevicesCommFailure	TRUE – All servers configured at this Client shows error
	FALSE – There is at least um Server configured in this Client that doesn't shows error

Table 6-11. Modbus Client Diagnostic

When configured vital failure mode, this diagnostic is consulted and 3 seconds after it's state change from FALSE to TRUE, a switchover occurs if the other conditions for this event are satisfied.

Redundancy Event Log

MasterTool allows the observation of several logs for the Nexto PLC, among them the Redundancy Event Log. These messages, specific for redundancy, registers in the System Log modifications in, practically, every field of the diagnostics and redundancy commands structure data, which are the following:

- RedDgnLoc
- RedDgnRem
- RedCmdLoc
- RedCmdRem

In case of diagnostic structures, only the following fields don't generate diagnostics:

- wRedStateDuration
- wCycleCounter
- dwIECTimer
- SyncLinkStatistics NET_Stat

Each line presented in the log has the following columns:

- Timestamp: event time and date, with resolution in milliseconds
- Severity: information, warning, error or exception
- Description: text that describes the event
- Component: component that has generated the event, and in the Redundancy Event Log case, is "Redundancy Management"

The "Description" column text has the following information:

- The diagnostic or command name which has been modified
- The new value assumed by this diagnostic or command
- The RedDgnLoclsGeneral_Diag.wCycleCounter value, which can be used as sequence auxiliary information
- The RedDgnRem.sGeneral_Diag.CycleCounter, which can be used as sequence auxiliary information

An example of the Description column can be the following:

```
RedDgnLoc.sGeneral_Diag.eRedStat = Active [Local cycle = 1234, Remote
cycle = 1233].
```

To access this screen, a double click must be done on the device (NX3030) in the device tree, and then the tab "Log" must be selected. There's a filter that allows selecting only the "Redundancy Management" component, to show only the redundancy events.

ATTENTION: Some diagnostics may point to possible failures during the redundant system initialization and in the tasks first cycles. But in a correct system function these diagnostics no longer indicate errors right after the system initialization.

PX2612 Panel Test

The PX2612 panel is composed by buttons, LEDs and relays. Many of these resources are not used very often, thus are rarely tested and the defects may not be detected. It's important to run tests from time to time in order to verify if these resources are working properly, to avoid obscure failures to prevent the PX2612 use when it's necessary.

Test Mode Entry

The first step to test the PX2612 is to set it to test mode. This is done turning on the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit on both PLCs.

The PLC perceives that is in test mode when the following two bits are on:

- DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal (RedCmdLoc.TestModeLocal on in this PLC)
- DG_NX4010.tRedundancy.RedCmdRem.bTestModeLocal (RedCmdLoc.TestModeLocal on in the other PLC)

When both bits are on, the PLC turns on the DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bRedPanelTestMode diagnostic, to inform that the PX2612 is in test mode.

Test Mode Manual and Automatic Outputs

The user can finish the test mode manually; turning off the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit in both PLCs. Actually turning it off in just one PLC is enough, as the test mode demands this bit to be on in both PLCs. However, this practice is recommended.

In case the user forgets to turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit, it’s automatically turned off 15 minutes after being turned on, finishing automatically the test mode.

LEDs Testing

Thus, during the test mode, the 6 LEDs must blink, losing its normal utility, which is showing the redundancy state.

Buttons Test

At pressing a button in the test mode, a correspondent LED stops blinking, and remains on. The following Table 6-12 presents the connection between the pressed button and the LED which remains on.

Tested button	Correspondent LED
TURN ON PLC A	ACTIVE – PLC B
STAND-BY PLC A	STAND-BY PLC A
INACTIVE PLC A	INACTIVE PLC A
TURN ON PLC B	ACTIVE – PLC A
STAND-BY PLC B	STAND-BY PLC B
INACTIVE PLC B	INACTIVE PLC B

Table 6-12. Connection between buttons and LEDs in the PX2612 button test

It can be observed that normally the LED is on the pressed button side, except for the TURN ON PLCx.

Before the LED remains on, it’s necessary to hold the button for, at least, 1 second. The LED returns to blinking after it’s released.

During the test mode, the buttons don’t allow the execution of functions which would be executed out of this mode, such as to cause a redundancy state change.

Relay Test

To test the relays, it was created the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal bit. At turning on this bit, if the PLC is in test mode and in Active state, it activates the relay, which must cause the other PLC (Non-Active) switching off. Turning off the

DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal, the relay is released, allowing the other PLC reactivating.

The command has no effect in the Non-Active PLC, to prevent it turns off the Active PLC.

The user must cause a switchover between PLCs (Active x Non-Active) in order to test the relay in both PLCs.

When the PLC which was switched off is reactivated and restarted, it returns with the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal off, thus the test mode is canceled. The DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit must be turned on again in this PLC and set it to Active state before testing its relay.

When the test mode is finished, the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit is automatically turned off, in case the user has forgotten it on.

Suggested Sequence for PX2612 Test Executing

The following sequence is suggested in order to execute the PX2612 tests:

- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit in both PLCs (PLCA and PLCB).
- It must be observed if the 6 LEDs are blinking.
- Press, one at a time, the 6 buttons and verify if the correspondent LED stops blinking and remain on while the button is pressed. It must be remembered the button must remain pressed for, at least, one second before the LED stops blinking and remains on, and that the LED returns to blinking after the button is released.
- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC switching off.
- Turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC switching on.
- Wait until the Non-Active PLC is restarted and assumes the Stand-by state. The test mode is active as the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit is turned off at the restarting in Stand-by mode PLC.
- Cause a switchover between PLCs, pressing the Active PLC STAND-BY button. The normal use of the STAND-BY button is possible because the test mode isn’t active.
- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit in the new Active PLC, which has just gotten out the Stand-by state. This way, the test mode is reactivated.
- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC switching off.
- Turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC reactivating.
- Turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit in the Active PLC, to finish the test mode. It’s not necessary to do this in the Stand-by PLC, as it has just initialized, with the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit off.

7. Maintenance

Module Diagnostics

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 WEB
- 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.

One Touch Diag

The One Touch Diagnostics , or single touch, 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 off-line (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 CPU’s Informative and Configuration Menu chapter. Figure 7-1 shows the CPU switch placement:

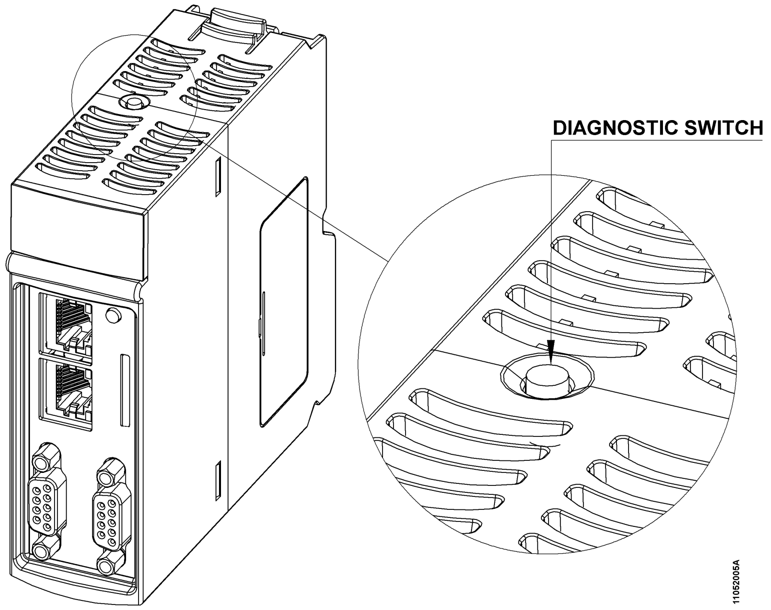


Figure 7-1. Diagnostic Switch

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 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.

Figure 7-2 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.

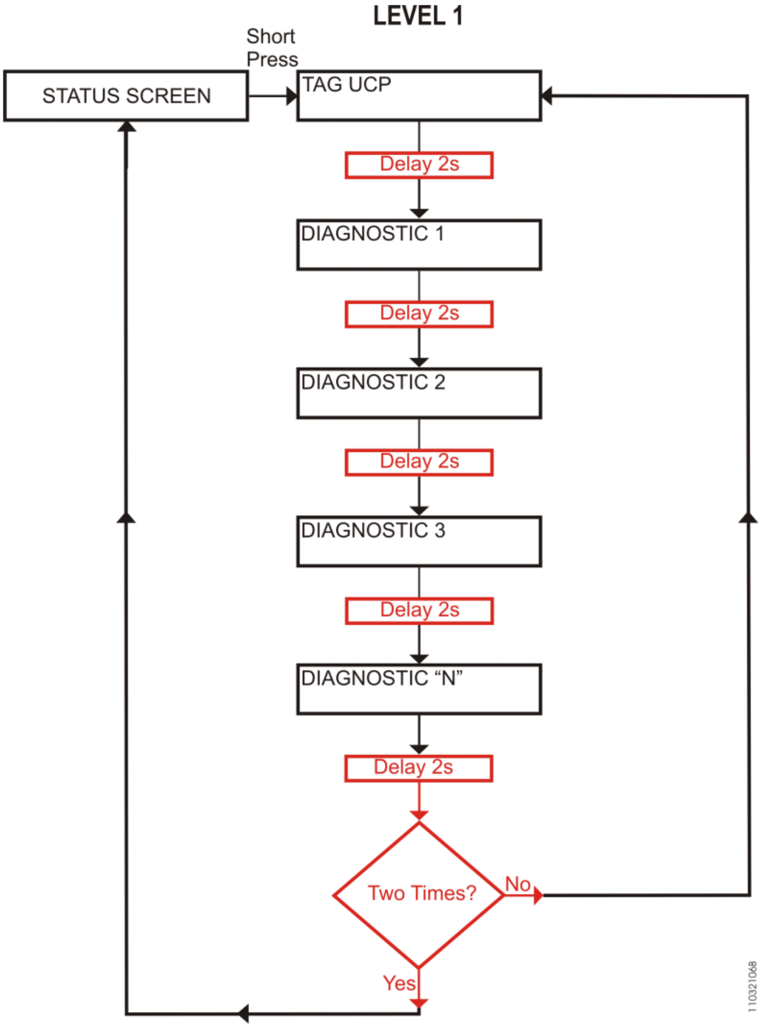


Figure 7-2. CPU Diagnostics Visualization

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.

In case a long touch is executed, the CPU goes to navigation menu, which is described in the CPU's Informative and Configuration Menu chapter. Also, it is important to observe that the One Touch Diag could be available when the module could be in Operational Mode.

Table 7-1 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 1s till 20 s
Locked Switch	20,01 s	(∞)	Diagnostics indication, see on Table 7-6

Table 7-1. 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 7-6.

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 - MU214000.

Diagnostics via LED

Nexto Series CPUs have a LED for diagnostic indication (LED DG) and a LED for watchdog event indication (LED WD). Table 7-2 and Table 7-3 show the meaning of each state and its respective descriptions:

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 7-2. Description of the Diagnostic LEDs States

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 7-3. 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 7-10.

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 4-1.

RJ45 Connector LEDs

Both LEDs placed in the RJ45 connectors (in case of NX3010, only one connector), identified by NET 1 and NET 2, 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 7-4.

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 7-4. Ethernet LEDs Meaning

Diagnostics via WEB

Besides the previously presented features, the Nexto Series brings to the user a innovating access tool to the system diagnostics and operation states, through a WEB page.

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

To access the desired CPU WEB page, it is just to use a standard navigator (Internet Explorer 7 or superior, Mozilla Firefox 3.0 or superior and Google Chrome 8 or superior) and type, on the address bar, the CPU IP address (e.g. Ex.: http://192.168.1.1). First, the CPU information is presented, according to Figure 7-3:

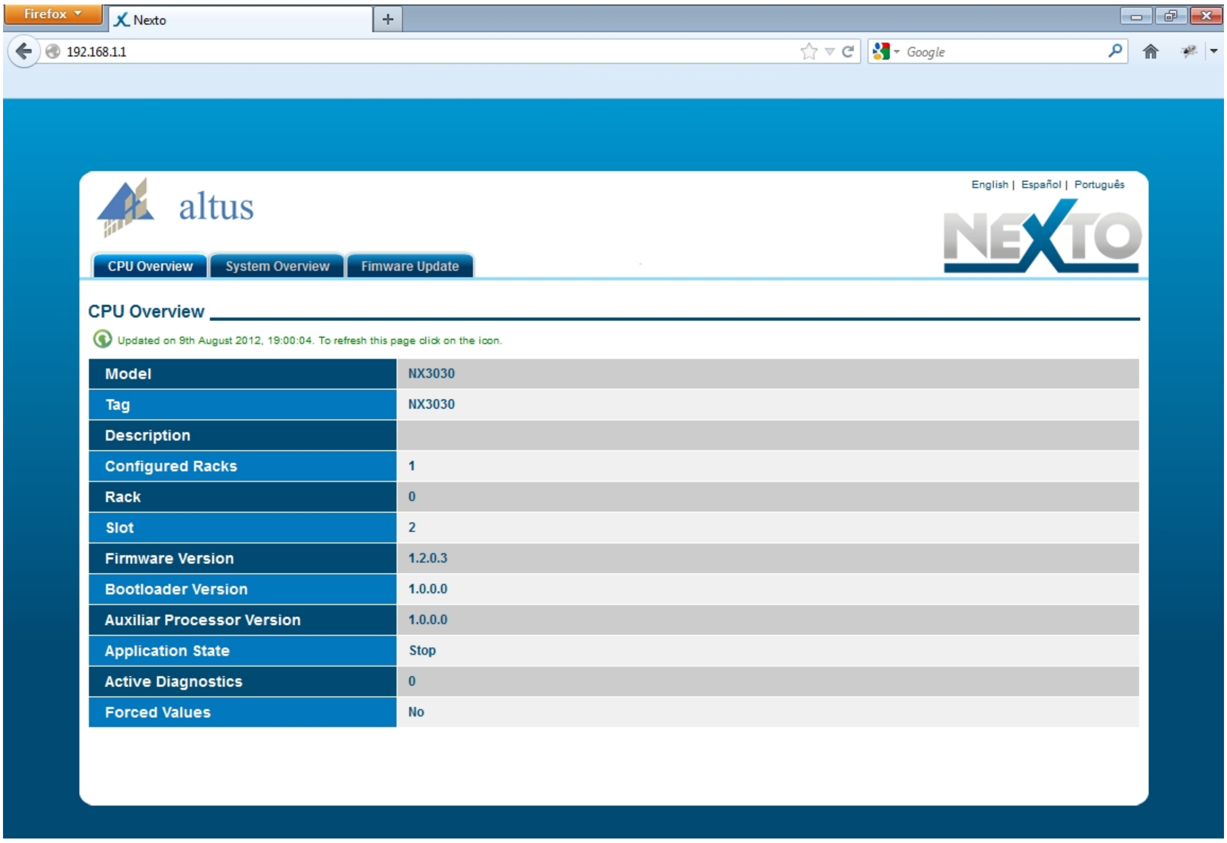


Figure 7-3. Initial Screen

There is also the “System 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 7-4. Otherwise a list with the system connected modules, Tags and active diagnostics number is presented:

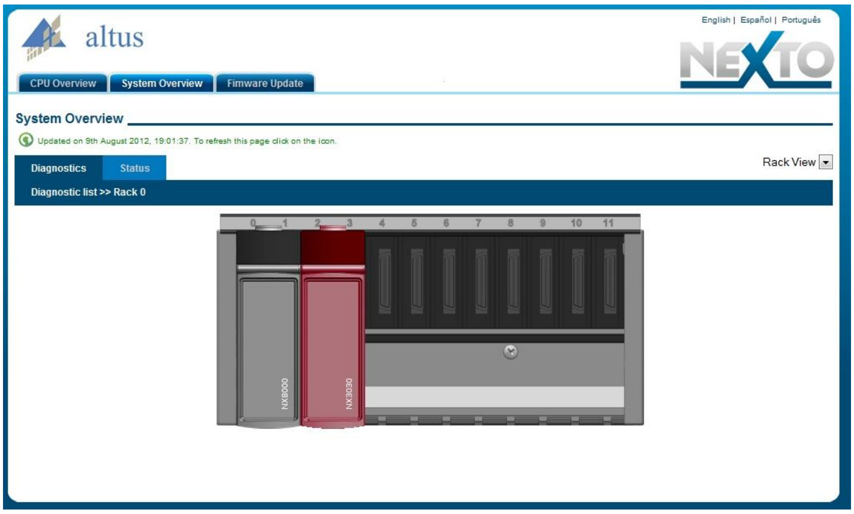


Figure 7-4. System Information

When the module with diagnostics is pressed, the module active(s) diagnostic(s) are shown, as illustrated on Figure 7-5:

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.

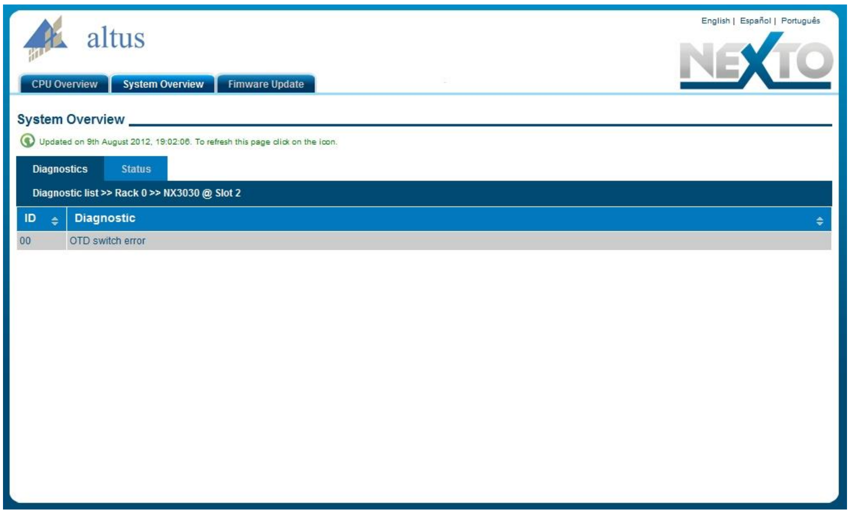


Figure 7-5. System Diagnostics

In case the Status tab is selected, the state of all detailed diagnostics is shown on the screen, as illustrated on Figure 7-6:

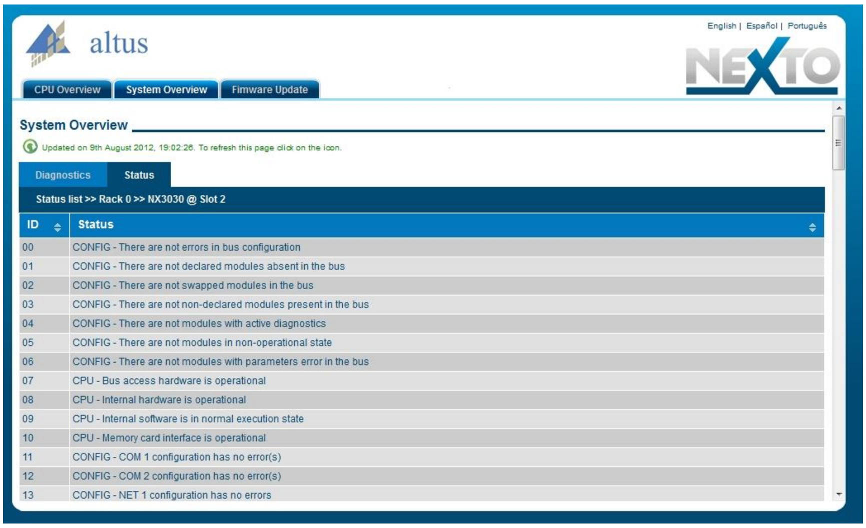


Figure 7-6. System Status

Besides, the user can choose to visualize three language options: Portuguese, English and Spanish. It is just to change the upper right menu for the desired language. The correspondent Firmware Updating tab is restricted to the user, in other words, it is for Altus internal use only.

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

Diagnostic Explorer

The Diagnostic Explorer is the inclusion of the diagnostics via WEB in the MasterTool IEC XE, in order to make the process faster and direct.

The access to this feature happens in two ways:

- Accessing the “Diagnostic Explorer” option in the device tree, placed on the MasterTool IEC XE screen left, and putting the correct IP in the field indicated on Figure 7-7. Remembering that for the diagnostics page to be shown, the user must be connected to the CPU (Login chapter)

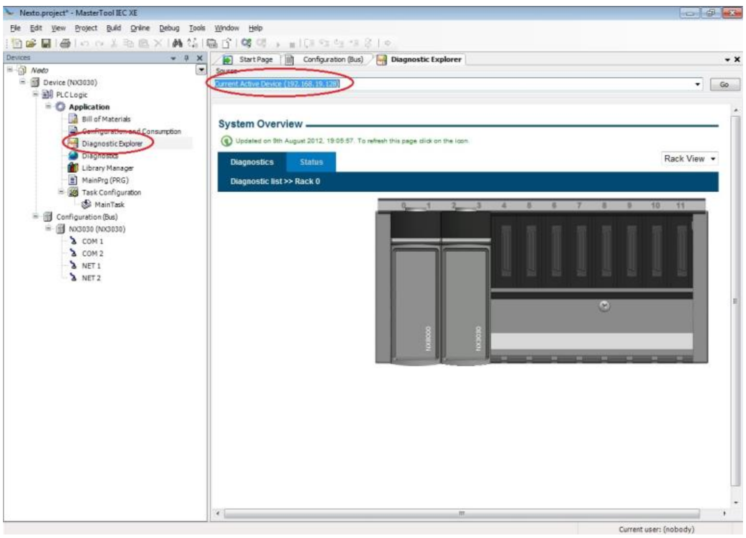


Figure 7-7. Diagnostic Explorer Screen

- Right - clicking on the module and selecting "Diagnostic", the Diagnostic Explorer is opened, directing for the module status page

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 Diagnostics created automatically by the MasterTool IEC XE.

Table 7-5 summarizes the diagnostic byte/words division:

Byte	Description
0 to 3	CPU summarized diagnostics.
4 to 558	CPU detailed diagnostics (NX3004, NX3005 and NX3010).
4 to 693	CPU detailed diagnostics (NX3020 and NX3030).

Table 7-5. CPU Diagnostics Division

Summarized Diagnostics

Table 7-6 shows the meaning of each CPU summarized diagnostic bit:

Direct Representation Variable		Diagnostics Message	DG_Modulo.tSummarized.* AT Variable	Description
Variable	**Bit**			
-	-	NO DIAG	-	There is no active diagnostic.
%QB(n)	0.0	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.
	0.1	ABSENT MODULES	bAbsentModules	TRUE – One or more declared modules are absent.
				FALSE – All declared modules were detected in the bus.
	0.2	SWAPPED MODULES	bSwappedModules	TRUE – TRUE – There are changed modules in the bus.
				FALSE – There are no changed modules in the bus.
	0.3	NON-DECLARED MODULES	bNonDeclaredModules	TRUE – One or more modules in the bus were not declared in the configuration.
				FALSE – All modules were declared.
	0.4	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.
	0.5	MODULES W/ FATAL ERROR	bModuleFatalError	TRUE – One or more modules in the bus are in fatal error.
				FALSE – all modules are working properly.
	0.6	MODULES W/ PARAM. ERROR	bModuleParameterError	TRUE – One or more modules in the bus have parameterization error.
				FALSE – All modules are parameterized.
	0.7	BUS ERROR	bWHSBBusError	TRUE – Master indication there is failure in the WHSB bus.
				FALSE – The WHSB bus is working properly.
%QB(n+1)	1.0	HARDWARE FAILURE	bHardwareFailure	TRUE – CPU hardware failure.

Direct Representation Variable		Diagnostics Message	DG_Modulo.tSummarized.* AT Variable	Description
Variable	**Bit**			
				FALSE – The hardware is working properly.
	1.1	SOFTWARE EXCEPTION	bSoftwareException	TRUE – One or more exceptions generated by the software.
				FALSE – No exceptions generated in the software.
	1.2		bReserved_10	Reserved
	1.3	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.
	1.4	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.
	1.5	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.
	1.6	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.
	1.7	NET2 CONF. ERROR	bNET2ConfigError	TRUE Some error occurred during or after the NET 2 Ethernet interface.
				FALSE – The NET 2 Ethernet interface configuration is not correct.
%QB(n+2)	2.0	INVALID DATE/TIME	bInvalidDateTime	TRUE – The date or hour are invalid.
				FALSE – The date or hour are correct.
	2.1	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.
	2.2	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.
	2.3 a 2.7		bReserved_xx	Reserved
%QB(n+3)	3.0	ABSENT RACK	bAbsentRacks	TRUE – One or more declared racks are absent.
				FALSE – There are no absent racks.
	3.1	DUPLICATED RACK	bDuplicatedRacks	TRUE – There are racks with a duplicated identification number.
				FALSE – There are no racks with a duplicated identification number.
	3.2	INVALID RACK	bInvalidRacks	TRUE – There are racks with an invalid identification number.
				FALSE – There are no racks with an invalid identification number.
	3.3	NON	bNonDeclaredRacks	TRUE – There are racks with a non-

Direct Representation Variable		Diagnostics Message	DG_Modulo.tSummarized. * AT Variable	Description
Variable	Bit			
		DECLARED RACK		declared identification number.
				FALSE – There are no racks with a non-declared identification number.
	3.4	DUPLICATED SLOT	bDuplicatedSlots	TRUE – There are some duplicated slot address.
				TRUE – There are no duplicated slot address.
	3.5 a 3.7		bReserved_xx	Reserved

Table 7-6. CPU Summarized Diagnostics Table

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.

Error in Memory Card: Memory Card diagnostics are only available for the NX3010, NX3020 and NX3030 CPUs. **Diagnostic Message:** The diagnostics messages can be visualized by the CPU graphic display using the OTD key or using the WEB, through the CPU diagnostics page.

Serial Interfaces: COM 2 interface diagnostics are only available for the NX3010, NX3020 and NX3030 CPUs.

Ethernet Interfaces: The diagnostics regarding to the NET 2 interface are only available for the NX3020 and NX3030 CPUs.

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 datatypes 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.tSummarized., where the word Module must be replaced by the CPU being used. For example, for the Configuration Mismatch diagnostic, use DG_NX3010.tSummarized.bConfigMismatch. The AT directive is a reserved word of the programmer, and some symbolic variables that use this directive indicate diagnostics.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QD(n+4)					DWORD	Target. dwCPUModel	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 7-7. Target Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%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	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 7-8. Hardware Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QW(n+21)					WORD	Exception. wExceptionCode	Exception code generated by the RTS. See Table 7-29.
%QB(n+23)					BYTE	Exception. byProcessorLoad	Level, in percentage (%), of charge in the processor.

Table 7-9. 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
		0x0054	Stack overflow
0x0013	Checksum error after the program download.	**0x0055**	Invalid disposition
0x0014	Fieldbus error.	**0x0056**	Invalid maneuver
0x0015	I/O updating error.	**0x0057**	Protected page
0x0016	Cycle time (execution) exceeded.	**0x0058**	Double failure
0x0017	Program online updating too long.	**0x0059**	Invalid OpCode
0x0018	External references not resolved.	**0x0100**	Data type misalignment
0x0019	Download rejected.	**0x0101**	Arrays limit exceeded
0x001A	Project not loaded, as the retentive variables cannot be reallocated.	**0x0102**	Division by zero
0x001B	Project not loaded and deleted.	**0x0103**	Overflow
0x001C	Out of memory stack.	**0x0104**	Cannot be continued
0x001D	Retentive memory is corrupted and cannot be mapped.	**0x0105**	Watchdog in the processor load of all IEC task detected.
0x001E	Project can be loaded but causes a drop later on.	**0x0150**	FPU: Not specified error
0x0021	Target of startup application does not match to the current target.	**0x0151**	FPU: Operand is not normal.
0x0022	Scheduled tasks error.	**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 7-10. RTS Exception codes

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
NA	%QB (n+24)	NA			BYTE	WebVisualization. byConnectedClients	Clients number connected to the WebVisualization.

Table 7-11. WebVisualization Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QB(n+25)					BYTE	RetainInfo. byCPUInitStatus	CPU Startup status: 01: Hot start 02: Warm Start 03: Cold Start PS.: These variables are restarted in all startup.
%QW(n+26)					WORD	RetainInfo. wCPUColdStartCounter	Counter of cold startups: It will be added only due hot removal of the CPU in the bus and not due to the command of Cold Reset from MasterTool IEC XE. (0 to 65535)
%QW(n+28)					WORD	RetainInfo. wCPUWarmStartCounter	Counter of hot startups: It will be added only during a sequence of startup of the system and not due the command of Hot Reset from MasterTool IEC XE. (0 to 65535)
%QW(n+30)					WORD	RetainInfo. wCPUHotStartCounter	Counter of disorders lower than the time of support to failures in the CPU power supply. (0 to 65535)
%QW(n+32)					WORD	RetainInfo. wRTSResetCounter	Counter of reset performed by the RTS (Runtime System). (0 to 65535)
%QW(n+34)					WORD	RetainInfo.wReserved_0	Reserved

Table 7-12. RetainInfo Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%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 7-13. 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.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QX(n+37).0					BIT	Thermometer. bOverTemperatureAlarm	Alarm generated due internal temperature at 85 °C or above it.
%QX(n+37).1					BIT	Thermometer. bUnderTemperatureAlarm	Alarm generated due internal temperature at 0º or under it.
%QD(n+38)					DINT	Thermometer. diTemperature	Temperature read in the internal sensor of the CPU.

Table 7-14. 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.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%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 65535)
%QW(n+53)					WORD	Serial.COM1. wTXPendingBytes	Number of characters left in the transmission buffer in COM 1. (0 to 65535)
%QW(n+55)					WORD	Serial.COM1. wBreakErrorCounter	These counters are restarted in the following conditions: - Energizing - Configuration of the COM 1 serial port - Removal of RX and TX queues PS.: When the CPU is set with parity Without Parity, the counter of errors of parity is not incremented in case it receives a different parity. In this case, it will be indicated an error of frame. The maximum value of each counter is 65535.
%QW(n+57)					WORD	Serial.COM1. wParityErrorCounter	
%QW(n+59)					WORD	Serial.COM1. wFrameErrorCounter	
%QW(n+61)					WORD	Serial.COM1. wRXOverrunCounter	
%QW(n+63)					WORD	Serial.COM1. wReserved_0	Reserved
%QW(n+65)					WORD	Serial.COM1. wReserved_1	Reserved

Table 7-15. 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 diferent parity. In this case, a frame error will be indicated.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
NA		%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
NA		%QD(n+68)			DWORD	Serial.COM2. dwRXBytes	Counter of characters received from COM 2. (0 to 4294967295)
NA		%QD(n+72)			DWORD	Serial.COM2. dwTXBytes	Counter of characters transmitted through COM 2. (0 to 4294967295)
NA		%QW(n+76)			WORD	Serial.COM2. wRXPendingBytes	Number of characters left in the reading buffer in COM 2. (0 to

				65535)
NA	%QW(n+78)	WORD	Serial.COM2. wTXPendingBytes	Number of characters left in the transmission buffer in COM 2. (0 to 65535)
NA	%QW(n+80)	WORD	Serial.COM2. wBreakErrorCounter	These counters are restarted in the following conditions: - Startup - Configuration of COM 2 serial port - Removal of RX and TX queues PS.: When the CPU is set with parity Without Parity, the counter of errors of parity is not increased in case it receives a different parity. In this case, it will be indicated a frame error. The maximum value of each counter is 65535.
NA	%QW(n+82)	WORD	Serial.COM2. wParityErrorCounter	
NA	%QW(n+84)	WORD	Serial.COM2. wFrameErrorCounter	
NA	%QW(n+86)	WORD	Serial.COM2. wRXOverrunCounter	
NA	%QW(n+88)	WORD	Serial.COM2. wReserved_0	Reserved
NA	%QW(n+90)	WORD	Serial.COM2. wReserved_1	Reserved

Table 7-16. 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.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QX(n+92).0					BIT	Ethernet.NET1. bLinkDown	Indicates the state of the link in NET 1.
%QW(n+93)					WORD	Ethernet.NET1. wProtocol	Selected protocol in NET 1: 00: Without 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	IP NET 1 Address
%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	IP NET 1 Address
%QB(n+165)					BYTE ARRAY(4)	Ethernet.NET1. abyMask	NET 1 Subnet Mask
%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 sent packages through NET 1 port. (0 to 4294967295)
%QD(n+183)	DWORD	Ethernet.NET1. dwPacketsReceived	Counter of received packages through NET 1 port. (0 to 4294967295)
%QD(n+187)	DWORD	Ethernet.NET1. dwBytesSent	Counter of sent bytes through NET 1 port. (0 to 4294967295)
%QD(n+191)	DWORD	Ethernet.NET1. dwBytesReceived	Counter of received bytes through NET 1 port. (0 to 4294967295)
%QW(n+195)	WORD	Ethernet.NET1. wTXErrors	Counter of errors of transmission through NET 1 port. (0 to 65535)
%QW(n+197)	WORD	Ethernet.NET1. wTXFIFOErrors	Counter of errors in the buffer of transmission through NET 1 port. (0 to 65535)
%QW(n+199)	WORD	Ethernet.NET1. wTXDropErrors	Counter of connection losses in the transmission through NET 1 port. (0 to 65535)
%QW(n+201)	WORD	Ethernet.NET1. wTXCollisionErrors	Counter of errors of collision in the transmission through NET 1 port . (0 to 65535)
%QW(n+203)	WORD	Ethernet.NET1. wTXCarrierErrors	Counter of errors of the carrier in the transmission through NET 1 port. (0 to 65535)
%QW(n+205)	WORD	Ethernet.NET1. wRXErrors	Counter of errors of reception through NET 1 port. (0 to 65535)
%QW(n+207)	WORD	Ethernet.NET1. wRXFIFOErrors	Counter of errors in the buffer of reception through NET 1 port. (0 to 65535)
%QW(n+209)	WORD	Ethernet.NET1. wRXDropErrors	Counter of connection losses in the reception through NET 1 port. (0 to 65535)
%QW(n+211)	WORD	Ethernet.NET1. wRXFrameErrors	Counter of errors of frame in the reception through NET 1 port. (0 to 65535)
%QW(n+213)	WORD	Ethernet.NET1. wMulticast	Counter of multicast packages through NET 1. (0 to 65535)
%QW(n+215)	WORD	Ethernet.NET1. wReserved_0	Reserved
%QW(n+217)	WORD	Ethernet.NET1. wReserved_1	Reserved

Table 7-17. Ethernet NET 1 Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
NA			%QX(n+219).0		BIT	Ethernet.NET2. bLinkDown	Indicates the link state in NET 2.
NA			%QW(n+220)		WORD	Ethernet.NET2 .wProtocol	Protocol selected in NET 2: 00: Without protocol
NA			%QX(n+220).0		BIT	Ethernet.NET2. wProtocol.bMODBUS_RTU_ETH_Client	MODBUS RTU Client via TCP
NA			%QX(n+220).1		BIT	Ethernet.NET2. wProtocol.bMODBUS_ETH_Client	MODBUS TCP Client
NA			%QX(n+220).2		BIT	Ethernet.NET2. wProtocol.bMODBUS_RTU_ETH_Server	MODBUS RTU Server via TCP
NA			%QX(n+220).3		BIT	Ethernet.NET2. wProtocol.bMODBUS_ETH_Server	MODBUS TCP Server
NA			%QB(n+222)		STRING (15)	Ethernet.NET2. szIP	IP NET 2 Address
NA			%QB(n+238)		STRING	Ethernet.NET2.	NET 2 Subnet Mask

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
					(15)	szMask	
NA			%QB(n+254)		STRING (15)	Ethernet.NET2. szGateway	NET 2 Gateway Address
NA			%QB(n+270)		STRING (17)	Ethernet.NET2. szMAC	NET 2 MAC Address
NA			%QB(n+288)		BYTE ARRAY(4)	Ethernet.NET2. abyIP	IP NET 2 Address
NA			%QB(n+292)		BYTE ARRAY(4)	Ethernet.NET2. abyMask	NET 2 Subnet Mask
NA			%QB(n+296)		BYTE ARRAY(4)	Ethernet.NET2. abyGateway	NET 2 Gateway Address
NA			%QB(n+300)		BYTE ARRAY(6)	Ethernet.NET2. abyMAC	NET 2 MAC Address
NA			%QD(n+306)		DWORD	Ethernet.NET2. dwPacketsSent	Counter of packages sent through NET 2 port. (0 to 4294967295)
NA			%QD(n+310)		DWORD	Ethernet.NET2. dwPacketsReceived	Counter of packages received through NET 2 port. (0 to 4294967295)
NA			%QD(n+314)		DWORD	Ethernet.NET2. dwBytesSent	Counter of bytes sent through NET 2 port. (0 to 4294967295)
NA			%QD(n+318)		DWORD	Ethernet.NET2. dwBytesReceived	Counter of bytes received through NET 2 port. (0 to 4294967295)
NA			%QW(n+322)		WORD	Ethernet.NET2. wTXErrors	Counter of errors of transmission through NET 2 port. (0 to 65535)
NA			%QW(n+324)		WORD	Ethernet.NET2. wTXFIFOErrors	Counter of errors in the buffer of transmission through NET 2 port. (0 to 65535)
NA			%QW(n+326)		WORD	Ethernet.NET2. wTXDropErrors	Counter of connection losses in the transmission through NET 2 port. (0 to 65535)
NA			%QW(n+328)		WORD	Ethernet.NET2. wTXCollisionErrors	Counter of errors of collision in the transmission through NET 2 port. (0 to 65535)
NA			%QW(n+330)		WORD	Ethernet.NET2. wTXCarrierErrors	Counter of errors of the carrier in the transmission through NET 2 port. (0 to 65535)
NA			%QW(n+332)		WORD	Ethernet.NET2. wRXErrors	Counter of errors of reception through NET 2 port. (0 to 65535)
NA			%QW(n+334)		WORD	Ethernet.NET2. wRXFIFOErrors	Counter of errors in the buffer of reception through NET 2 port. (0 to 65535)
NA			%QW(n+336)		WORD	Ethernet.NET2. wRXDropErrors	Counter of connection losses in the reception through NET 2 port. (0 to 65535)
NA			%QW(n+338)		WORD	Ethernet.NET2. wRXFrameErrors	Counter of errors of frame in the reception through NET 2 port. (0 to 65535)
NA			%QW(n+340)		WORD	Ethernet.NET2. wMulticast	Counter of multicast packages through NET 2 port. (0 to 65535)
NA			%QW(n+342)		WORD	Ethernet.NET2. wReserved_0	Reserved
NA			%QW(n+344)		WORD	Ethernet.NET2. wReserved_1	Reserved

Table 7-18. Ethernet NET 1 Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QB(n+219)			%QB(n+346)		BYTE	UserFiles. byMounted	Indicates if the memory used for recording user files is able to receive data.
%QD(n+220)			%QD(n+347)		DWORD	UserFiles. dwFreeSpacekB	Free memory space for user files in Kbytes.
%QD(n+224)			%QD(n+351)		DWORD	UserFiles. dwTotalSizekB	Storage capacity of the memory of user files in Kbytes.
%QB(n+228)			%QB(n+355)		BYTE	UserFiles.byReserved_0	Reserved

Table 7-19. 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.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QB(n+229)			%QB(n+356)		BYTE	UserLogs. byMounted	Status of memory in which are inserted the user logs.
%QW(n+230)			%QW(n+357)		WORD	UserLogs. wFreeSpacekB	Free space in the memory of user logs in Kbytes.
%QW(n+232)			%QW(n+359)		WORD	UserLogs. wTotalSizekB	Storage capacity of the memory of user logs in Kbytes.
%QB(n+234)			%QB(n+361)		BYTE	UserLogs. byReserved_0	Reserved

Table 7-20. UserLogs Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
NA		%QB (n+235)	%QB(n+362)		BYTE	MemoryCard. byMounted	Status of the Memory Card: 00: Memory card not mounted 01: Memory card inserted and mounted
NA		%QX(n+ 236).0	%QX(n+363).0		BIT	MemoryCard. bMemcardtoCPUEnabled	Protection level of the Memory Card: Data reading of the memory card by the authorized CPU.
NA		%QX(n+ 236).1	%QX(n+363).1		BIT	MemoryCard. bCPUtoMemcardEnabled	Data writing in the memory card by the authorized CPU.
NA		%QD (n+237)	%QD(n+364)		DWORD	MemoryCard. dwFreeSpacekB	Free space in the Memory Card in Kbytes.
NA		%QD (n+241)	%QD(n+368)		DWORD	MemoryCard. dwTotalSizekB	Storage capacity of the Memory Card in Kbytes.

Table 7-21. MemoryCard Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QB(n+245)			%QB(n+372)		BYTE	WHSB. byHotSwapAndStartupStatus	Informs the abnormal situation in the bus which caused the application stop for each mode of hot swapping. See Table 7-23 for more information
%QB(n+246)			%QB(n+373)		BYTE	WHSB.byReserved_0	Reserved

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QD(n+247)			%QD(n+374)		DWORD ARRAY (32)	WHSB. adwRackIOErrorStatus	Identification of errors in I/O modules, individually: The Array[0..31] represents 32 backplane racks, being each position made up by 32 bits. Each bit of these DWORDs represents the bus position, being the Bit-0 equivalent to position 0. So, it is performed a OR logic operation among four summarized diagnostics and the operational state of a certain position, and if the result is positive, the analogic bit will be true. Under, the considered diagnostics: Incompatible configuration (bConfigMismatch) Absent modules (bAbsentModules) Exchanged modules (bSwappedModules) Modules with fatal error (bModuleFatalError) Module is not in Operational Mode
%QD(n+375)			%QD(n+502)		DWORD ARRAY (32)	WHSB. adwModulePresenceSta tus	Status of presence of declared I/O modules in buses, individually: The Array[0..31] represents 32 backplane racks, being each position made up by 32 bits. Each bit of this DWORD represents a position in the bus, being the Bit-0 equivalent to position 0. So, if the module is present, the bit will be true. PS.: This diagnostic is valid for all modules, except for non-redundant power supplies, non-declared modules and CPUs, that is, do not present a presence in the bus in its respective positions (bit remains in false).
%QB(n+503)			%QB(n+630)		BYTE	WHSB. byWHSBBusErrors	Counter of failures in the WHSB bus. This counter is restarted in the energization. (0 to 255)

Table 7-22. 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 stop can be consulted below:

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 with 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 7-23. Codes of the Situations in which the Application Stops

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QB(n+504)			%QB(n+631)		BYTE	Application. byCPUState	Informs the operation state of the CPU: 01: All user applications are in Start Mode. 03: All user application is in Stop Mode.
%QX(n+505).0			%QX(n+632).0		BIT	Application. bForcedIOs	There is one or more forced I/O points.

Table 7-24. Application Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QX(n+532).0			%QX(n+633).0		BIT	SNTP. bServiceEnabled	SNTP Service enabled.
%QB(n+533)			%QB(n+634)		BYTE	SNTP. byActiveTimeServer	Indicates which server is active: 00: None active server. 01: Active Primary Server. 02: Active Secondary Server.
%QW(n+534)			%QW(n+635)		WORD	SNTP. wPrimaryServerDownCount	Counter of times in which the primary server is unavailable. (0 to 65535)
%QW(n+536)			%QW(n+637)		WORD	SNTP. wSecondaryServerDownCount	Counter of times in which the secondary server is unavailable. (0 to 65535)
%QD(n+538)			%QD(n+639)		DWORD	SNTP. dwRTCTimeUpdatedCount	Counter of times the RTC was updated by the SNTP service. (0 to 4294967295)
%QB(n+542)			%QB(n+643)		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)			%QB(n+644)		BYTE	SNTP. byLastUpdateTimeServer	Indicates which server was used in the last update: 00: None update. 01: Primary Server. 02: Secondary Server.
%QB(n+544)			%QB(n+645)		BYTE	SNTP. sLastUpdateTime.byDayOfMonth	Day of the last update of the RTC.
%QB(n+545)			%QB(n+646)		BYTE	SNTP.sLastUpdateTime.byMonth	Month of the last update of the RTC.
%QW(n+546)			%QW(n+647)		WORD	SNTP.sLastUpdateTime.wYear	Year of the last update of the RTC.
%QB(n+548)			%QB(n+649)		BYTE	SNTP.sLastUpdateTime.byHours	Hour of the last update of the RTC.
%QB(n+549)			%QB(n+650)		BYTE	SNTP.sLastUpdateTime.byMinutes	Minute of the last update of the RTC.
%QB(n+550)			%QB(n+651)		BYTE	SNTP.bReservedAlign	Reserved
%QB(n+551)			%QB(n+652)		BYTE	SNTP.sLastUpdateTime.bySeconds	Second of the last update of RTC.
%QW(n+552)			%QW(n+653)		WORD	SNTP.sLastUpdateTime.wMilliseconds	Millisecond of the last update of RTC.
%QW(n+554)			%QW(n+655)		WORD	SNTP.wReserved_0	Reserved
%QW(n+556)			%QW(n+657)		WORD	SNTP.wReserved_1	Reserved

Table 7-25. SNTP Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
NA			%QX(n+659).0		BIT	SOE[1]. bConnectionStatus	Connection status of client 01
NA			%QX(n+659).1		BIT	SOE[1]. bOverflowStatus	Queue status of client events 01 FALSE – There was no overflow TRUE – Exceeded queue limit
NA			%QB(n+660)		BYTE	SOE[1]. byReserved_0	Reserved
NA			%QW(n+661)		WORD	SOE[1]. wEventsCounter	Counter of events in the client queue 01
NA			%QX(n+663).0		BIT	SOE[2]. bConnectionStatus	Connection status of client 02

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
NA			%QX(n+663).1		BIT	SOE[2]. bOverflowStatus	Queue status of client events 02 FALSE – There was no overflow TRUE – Exceeded queue limit
NA			%QB(n+664)		BYTE	SOE[2]. byReserved_0	Reserved
NA			%QW(n+665)		WORD	SOE[2]. wEventsCounter	Counter of events in the client queue 02

Table 7-26. SOE Detailed Diagnostics Group Description

Notes:

Synchronism of the SOE diagnostics group in systems operating with Half-Cluster redundancy: When a project is configured with Half-Cluster redundancy, the diagnostics of the SOE group are not synchronized between the two Half-Clusters.

SOE Diagnostics group update during transition to Active state: When a Half-Cluster passes from Stand-by to Active state the diagnostics of the SOE group starts be to updated on the third cycle.

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QD(n+506)			%QD(n+667)		DWORD	Rack. dwAbsentRacks	Each bit presents a rack identification number, if the bit is TRUE, it means the rack is absent.
%QD(n+510)			%QD(n+671)		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)			%QD(n+675)		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.
%QW(n+518)			%QW(n+679)		WORD	Rack.wReserved_0	Reserved

Table 7-27. Rack Detailed Diagnostics Group Description

CPU Direct Representation Variable					Size	DG_Modulo.tDetailed.* AT Variable	Description
NX3004	NX3005	NX3010	NX3020	NX3030			
%QD(n+520)			%QD(n+681)		DWORD	ApplicationInfo. dwApplicationCRC	32 bits CRC of Application. When the application is modified and sent to the CPU, a new CRC is generated
%QD(n+524)			%QD(n+685)		DWORD	ApplicationInfo. dwReserved_0	Reserved
%QD(n+528)			%QD(n+689)		DWORD	ApplicationInfo. dwReserved_0	Reserved

Table 7-28. ApplicationInfo Detailed Diagnostics Group Description

Diagnostics via Function Blocks

The function blocks allow the visualization of some parameters which cannot be accessed otherwise. The three functions regarding advanced diagnostics are in the Nexto Standard library and are described below.

GetTaskInfo

This function returns the task information of a specific application.

Figure 7-8. 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 7-29. GetTaskInfo Input Parameters

The data returned by the function, through the pointer informed in the input parameters are described on Table 7-30.

Returned Parameters	Size	Description
dwCurScanTime	DWORD	Task cycle time (execution) with 1µs resolution
dwMinScanTime	DWORD	Task cycle minimum time with 1µs resolution
dwMaxScanTime	DWORD	Task cycle maximum time with 1µs resolution
dwAvgScanTime	DWORD	Task cycle average time with 1µs resolution
dwLimitMaxScan	DWORD	Task cycle maximum time before watchdog occurrence
dwIECCycleCount	DWORD	IEC cycle counter

Table 7-30. GetTaskInfo Returned Parameters

Possible ERRORCODE:

- NoError: success execution
- TaskNotPresent: the desired task does not exist

Example of utilization in ST language:

```
PROGRAM MainPrg
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.
```

Graphic Display

The graphic display available in the Nexto Series CPUs is 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.

Below, on Figure 7-9 it is possible to observe the available characters in the Nexto CPU graphic display and, next, its respective meanings.

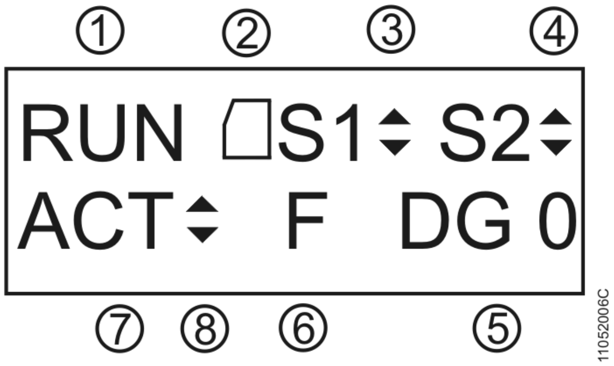


Figure 7-9. CPU Status Screen

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. For further details, see Run Mode and Stop Mode section.
2. Memory Card presence indication. Further details regarding its installation see Memory Card chapter.
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 1 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 Run Mode section.
7. Identification of the CPU redundancy state (message only valid in NX3030 in redundant mode). If the CPU is the active PLC, the ACT information will be presented. The other possible states are NCF (Not-configured), STR (Starting), INA (Inactive) and SBY (Stand-by).
8. Indication that the project synchronization is being executed. The up arrow (▲) indicates project data transmission and the down arrow (▼) indicates project data reception. For further information about the project synchronization see Project Synchronization 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.

Table 7-31 present the messages and their respective descriptions:

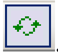
Message	Description
FORMATTING...	Indicates the CPU is formatting the memory card
FORMATTING ERROR	Indicates that an error occurred while formatting the memory card by the CPU.
WRONG FORMAT	Indicates that the memory card format is incorrect.
INCORRECT PASSWORD	Indicates the typed password is different 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
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 a 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 7-31. Other Messages of the Graphic Display

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, application download, 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(s).

Another way to filter the messages displayed to the user is to select the component desired to view.

The Log tab's *Timestamp* is shown by MasterTool after information provided by the device (CPU). MasterTool can display the *Timestamp* 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 *Timestamp* shown in MasterTool 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 subchapter of this manual.

> ATTENTION:
> The system logs of the Nexto series CPUs, starting in firmware version 1.4.0.33, 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.

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 LOAD" 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.

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 Features - 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 SUPPLY 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 SUPPLY FAILURE” message is shown on the CPU screen and the diagnostic is activated.

Figure 7-10. Power Supply Failure Message

The user can change the value of the variable attributed to the power supply failure to FALSE during the applicative execution, facilitating the verification and treatment of this diagnostic.

The POWER SUPPLY 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 chapter.

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 correct the voltage?
- Is the power supply the 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 are 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.

Troubleshooting

Table 7-32 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	For models NX3010, NX3020 and NX3030, the CPU must be placed in slots 2 and 3 of rack 0. Put it in the correct slots. CPUs NX3004 and NX3005 must be placed in slots 0 and 1 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 7-32. Troubleshooting Table

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

8. Glossary

Active CPU In a redundant system, the Active CPU performs the system control, reading the input points values, executing the applicative program and driving the output values.

ActivePrg POU from the program type, created automatically, which should be completed by the user. It runs only on the Active PLC, and used to control the automated process.

Active IP Strategy to facilitate the connection of Ethernet clients to Ethernet servers executed on a redundant PLC.

Active PLC Half-cluster (PLCA or PLCB) which is momentarily in Active state.

Adjust Bridge Address or configuration selecting switch composed by pins present in the circuit board and a small removable connector, used for selecting.

Addressable Variables The variable can be accessed directly in the memory using the desired address. E.g. QB0%, %MW100.

Applicative Project Part of the project corresponding to the executable code.

Applicative Program It's the program loaded into a PLC, which determines the operation of a machine or process.

AT Variable Reserved words in the programming software, used to indicate diagnostics.

Bus Electric signals cluster logically grouped with the function to transmit information and control between different elements of a sub-system.

Bit Basic information unit which can assume state 0 or 1.

Backoff Time the network node type CSMA/CD waits before re-transmit data after the occurrence of collisions in the physical environment.

Baud rate Rate at which information bits are transmitted via a serial interface or communications network (measured in bits/second).

Bridge Equipment for connecting two communication networks within the same protocol.

Broadcast Dissemination of information simultaneously to all nodes connected to a communication network.

Byte Information unit composed by 8 bits.

Category 5 One of the UTP cable categories:. Unshielded twisted-pair 100 ohm impedance and electrical characteristics supporting up to 100 MHz transmission frequency defined by the TIA / EIA 568-A, can be used in 10Base-T and 100Base networks TX, among others.

Cluster Set formed by the controllers (half-clusters) PLCA and PLCB.

Commercial code Product code, formed by the PO letters followed by four numbers.

Communication Network Group of devices (nodes) interconnected by communication channels.

Configuration Module Also called a module C. It is a single module in a PLC program that contains various parameters required to the controller functioning, such as the amount of variables and layout of the I/O modules in the bus.

CPU Abbreviation for central processing unit. Controls the information flow, interprets and executes program instructions and monitors the devices in the system.

CSMA/CD Physical layer access protocol, based in data collision, used for Ethernet networks.

Cycle Time It is the time that the CPU takes to run a particular application task.

Database Data base.

Default Pre-defined value for a variable, used in case there's no definition.

Deterministic Communication Network Communication network where the transmission and reception of information between different nodes is guaranteed with a maximum known time.

Diagnostic Procedure used to detect and isolate failures. It's also the data group used for such determination, which serves for problem analysis and correction.

Direct Representation Variable The variable can be accessed directly in memory, using a web address. For example:% QB0,% MW100.

DG Used abbreviation to indicate diagnostics in the LEDs

Download Program or configuration load in the PLC.

ESD Electrostatic discharge.

EIA RS-485 Industrial pattern (physical layer) for data communication.

EN 50170 In PROFIBUS networks, it's the standard which defines the fieldbus.

Frame An information unit transmitted through the network.

Freeze In PROFIBUS networks, it's the network state when the input data are frozen.

Full Duplex Indicates that the devices can perform communication by transmitting / receiving data in both directions simultaneously, i.e. it can transmit and receive at the same time.

Gateway Equipment for connecting two networks with different communication protocols.

Hardkey Connector normally connected to the parallel interface of a PC in order to avoid the execution of software illegal copies.

Hardware Physical equipment used in data processing where the programs (software) are executed.

Half-cluster Alternative name for each of the two controllers (PLCB and PLCA) that form a cluster.

Half Duplex Indicates that the devices can perform transmitting communications / receiving data, but only in one direction at a time, or can transmit or receive data.

HSDN High Speed Deterministic Network. Deterministic network, often redundant, used for exchanging interlocking messages between interlocking PLCs.

Hot swapping Procedure for replacement of system modules without the need for de-energizing the same. Usually used in exchanges of I/O modules.

IEC Stands for International Electrotechnical Commission, or International Electrotechnical Commission is an international standardization body that prepares and publishes international standards for electrical scope, electronic and related technologies.

IEC 61131 Generic standard for operation and utilization of PLCs. Old IEC 1131.

IEC 61131-3 Third part of the generic standard for operation and use of PLCs, IEC61131.

Interface Device which adapts electrically and/or logically the signal transference between two pieces of equipment.

Interruption High priority attending event which temporarily stops the program execution and detour for a specific attending routine.

Interval Defines how often a task is performed..

I/O See Input/output

I/O Modules Module belonging to the inputs and outputs subsystem.

I/O Subsystem Set of analog or digital I/O modules and interfaces of a programmable controller.

Input/output Also called I/O. Data I/O devices of a system. In case of PLCs, typically correspond to digital or analog inputs or outputs modules which monitor or activate the controlled device.

ISOL. Stands for isolated or isolating.

Kbytes Memory quantity unit. Means 1024 bytes (if 1kbyte).

LCD Acronym for Liquid Crystal Display

LED Light emitting diode. It's a type of diode that emits light when electrically stimulated. Used for light indication.

Logic Graphic matrix where are inserted the language instructions of a relay diagram which compose a applicative program. A group of logics organized in sequence form a program module.

MainPrg POU form the program type, created and filled automatically by MasterTool. The user must not change this POU.

MainTask The only task allowed by a redundant PLC. Calls the MainPrg program.

MasterTool IEC XE Identifies the Altus program for microcomputers, executable in Windows environment, which allows the development of applications for the Nexto CPUs series. Throughout the manual, this program is referred to by the acronym or as MasterTool IEC XE programmer..

Master Equipment connected to a communication network where the commands requests to the other network devices originate.

Master-Slave Communication Network Communication network where information transfers are initiated only from a single node (network master) connected to the data bus. The other network nodes (slaves) respond only when requested.

Menu Set of options available and displayed by a program on video and that can be selected by the user to activate or perform a certain task.

Monomaster In PROFIBUS networks, it's a network with only one master.

Module address Address which the PLC accesses a specific I/O module.

Module (referencing hardware) Basic element of a complete system that has well defined functions. Normally the system is connected by connectors and can be easily replaced.

Module (referencing software) Part of an application program capable of performing a specific function. It can be run independently or in conjunction with other modules, exchanging information via parameter passing.

Multicast Simultaneous dissemination of information to a particular group of interconnected nodes in a communication network.

Multimaster In PROFIBUS networks, it's a network with more than one master.

Multimaster Communication Network Communication network where information transfers are initiated by any node connected to the data bus.

NET 1 and NET 2 Logical names for the Ethernet interfaces present in the NX3010 CPU (NET 1 and NET 2) and modules NX5000. The CPU NX3010 and the NX5000 modules have only the NET1 interface.

NETA Denomination of one of the two synchronism channels between PLCA and PLCB. The other is called NETB.

NETB Denomination of one of the two synchronism channels between PLCA and PLCB. The other is called NETA.

Network Access Method used by all nodes in a communication network to synchronize data transmissions and resolve potential conflicts of simultaneous transmissions.

NIC Teaming Strategy to define pairs of redundant ports within a half-cluster, sharing the same Ethernet IP address.

Node Any station network with an ability to communicate using a protocol established.

Non-Active CPU	It's the CPU that is not in the active state (controlling the system) or in the stand-by state (overseeing the Active CPU). It can't assume control of the system.
Non-Active PLC	Half-cluster (PLCA or PLCB) which is momentarily in any state other than Active (Stand-by, Inactive, Not-configure or Starting).
Non-Redundant Data	Variables that are not shared between PLCA and PLCB in a redundant application. These variables correspond to diagnostic or private command to each half-cluster (PLCA or PLCB).
NonSkippedPrg	POU from the program type, created automatically, which should be completed by the user. It is executed in two PLCs (PLCA and PLCB), and used for actions and private variables management of each PLC (PLCA and PLCB), such as switchovers diagnostics and management due to not vital failures.
Octet	Eight bits group numbered from 0 to 7.
Operands	Elements on which the instructions work. They may represent constants, variables or set of variables.
Peer to peer	Type of communication where two nodes on a network exchange data and/or warnings without relying on a master.
PLC	Acronym for programmable logic controller.
PLCA	Denomination of one of the two controllers composing a redundant PLC. The other is called PLCB.
PLCB	Denomination of one of the two controllers composing a redundant PLC. The other is called PLCA.
POU	Program Organization Unit or Unit Program Organization, is a subdivision of the application program that can be written in any of the available languages.
Programming language	A group of rules and conventions used for a program creation.
Programmable controller	Also called PLC. Equipment which executes a control under the applicative program command. It's composed by a CPU, a power supply and a I/O structure.
Project Archive	Part of the project corresponding to the source code.
Project	The PLC project as a whole, composed by the project archive (source code) and by the applicative project (executable code).
Protocol	Rules of procedures and standard formats that through control signals, allow the establishment of a data transmission and error recovery between equipment.
RAM	Acronym for random access memory. It's where all the memory addresses can be accessed directly at random and at the same speed. It is volatile, thus, its contents are lost when the device is powered down, unless you have a battery for retaining values.
Redundant System	System that contains reserve elements or doubled to perform a certain task, which can tolerate certain types of failure without damaging the task execution.
Redundant CPU	Corresponds to the other system CPU, e.g. the CPU1 in relation to CPU2 and vice versa.
Redundant PLC	Set consisting on a cluster (PLCA and PLCB), PX2612 control panel, and remote I/O systems.
Relays language and Altus blocks	Group of instructions and variables that allow an applicative program edition to be used in a PLC.
Ripple	Ripple present in DC supply voltage.
RS-232C	It is a standard for serial data exchange between two points (point to point)
RS-422	It is a standard for serial data exchange between two or more points (point to point full duplex)
RS-485	It is a standard for serial data exchange between two or more points (point to point half duplex).
RunTime	See Cycle Time
RX	Acronym used to indicate serial reception.
SCADA	Supervisory Control and Data Acquisition. Supervisory system used for the plant control and operation.
ScTP	Stands for screened twisted pair. Even UTP cable, but all pairs of wires are surrounded by a metal foil or a metal braid screen, in order to minimize radiation and susceptibility to external noise. It is also known for SUTP (Screened Unshielded Twisted Pair) or FTP (Foil Twisted Pair).
Serial Channel	A device interface that transfers data in serial mode.
Slave	Equipment connected to a communication network that transmits data only if requested by another device called the master.
SNTP	Simple Network Time Protocol. Protocol for network time synchronization.
SOE	Sequence Of Events. Service to monitor the variation of pre-configured digital inputs, saving the date / time of the change and its new state.
Software	Computer programs, procedures and rules related to the operation of a data processing system.
Socket	Device on which integrated circuits or other components fit, making it easier to replace them and simplifying the maintenance.
Stand-by CPU	In a redundant system, it's the CPU that oversees the Active CPU, not executing the system control, but being ready to take control in case of failure on the Active CPU.
Stand-by PLC	Half-cluster (PLCA or PLCB) which is momentarily in Stand-by state.
Start up	Procedure for final clearance of the control system when the programs of all stations and remote CPUs are run together, having been developed and verified individually.
Subnet	Segment of a communication network that interconnects a group of devices (nodes) in order to isolate the local traffic or use different protocols or physical layer.
Supervision station	Equipment connected to a PLC or instrumentation network in order to monitor or control the process variables.

Sweeping cycle	A complete execution of the applicative program in a programmable controller.
Symbolic Variables	IEC Variables created in POUs and GVLs during the applicative development, which are not addressed directly in the memory.
Tag	Name associated with a variable or a logic that allows a brief identification of its contents.
Time-out	Predetermined maximum time that a communication is completed. If exceeded retentive or diagnostic procedures will be activated.
Toggle	Element that has two stable states, alternately exchanged each activation.
Token	It is a mark that indicates who is the master of the bus at the time.
TX	Acronym used to indicate serial transmission.
Upload	PLC configuration or program reading.
UTP	Stands for unshielded twisted pair. Cable type formed by pairs of unshielded twisted wires. For network applications, UTP term generally refers to the cable 100 ohms, Category 3, 4 or 5, specified by the TIA / EIA 568-A. Normally the UTP cable has four pairs of wires twisted within the same sheath (outer package).
Varistor	Protection device against voltage surge.
WD	Watchdog.
Word	Information unit composed by 16 bits.
Watchdog circuit	Electronic circuit used to verify the equipment function integrity.
10Base-T	Physical layer for Ethernet type defined in the IEEE 1990 standard supports 802.3i baud rates of 10 Mbps over two pairs of category 3 twisted strands.

9. Annex A. DNP3 Interoperability

DNP3 Device Profile

DNP3 DEVICE PROFILE DOCUMENT	
Device Identification	
Vendor Name	Altus S/A
Device Name	NX3030
Device Function	Slave
DNP Levels Supported for	Requests: None Responses: None
Connections Supported	IP Networking
Methods to set Configurable Parameters	Software: Master Tool IEC XE
IP Networking	
Type of End Point:	TCP Listening (Outstation Only)
Accepts TCP Connections from	Allows all
IP Address(es) from which TCP Connections are accepted:	*.*.*.*
TCP Listen Port Number	Configurable, range 1to 65535
TCP Keep-alive timer	Configurable, range 0 to 4294967295
Multiple master connections	Supports up to two masters Based on TCP port number
Time synchronization support	SNTP
Link Layer	
Data Link Address	Configurable, range 0 to 65519
Self Address Support using address 0xFFFC	No
Requires Data Link Layer Confirmation	Never
Maximum number of octets Transmitted in a Data Link Frame	Fixed at 292
Maximum number of octets that can be Received in a Data Link Frame	Fixed at 292
Application Layer	
Maximum number of octets Transmitted in an Application Layer Fragment	Fixed at 2048
Maximum number of octets that can be received in an Application Layer Fragment	Fixed at 2048
Timeout waiting for Application Confirm of solicited response message	Fixed at 10000 ms
Device Trouble Bit IIN1.6	This bit will be set if PLC is not in Run mode
Event Buffer Overflow Behavior	Discard the oldest event
Sends Multi-Fragment Responses	Yes
Outstation Unsolicited Response Support	
Supports Unsolicited Reporting	No

DNP V3.0 Implementation Table

DNP OBJECT GROUP & VARIATION			REQUEST Master may issue Outstation must parse		RESPONSE Master must parse Outstation may issue	
Group Num	**Var Num**	**Description**	**Function Codes (dec)**	**Qualifier Codes (hex)**	**Function Codes (dec)**	**Qualifier Codes (hex)**
1	0	Binary Input – Any Variation	1 (read)	00, 01 (start-stop) 06 (no range, or all)		
1	1	Binary Input – Packed format	1 (read)	00, 01 (start-stop) 06 (no range, or all)	129 (response)	00, 01 (start-stop)
2	0	Binary Input Event – Any Variation	1 (read)	06 (no range, or all) 07, 08 (limited qty)		
2	2	Binary Input Event – With absolute time	1 (read)	06 (no range, or all) 07, 08 (limited qty)	129 (response)	17, 28 (index)
60	1	Class Objects – Class 0 data	1 (read)	06 (no range, or all)		
60	2	Class Objects – Class 1 data	1 (read)	06 (no range, or all) 07, 08 (limited qty)		
80	1	Internal Indications – Packed format	1 (read)	00, 01 (start-stop)	129 (response)	00 (start-stop)
			2 (write)	00 (start-stop) index=7		