User Manual
Nexto Series CPU
NX3035

MU214619 Rev. C

July 31, 2026

www.altus.com.br

No part of this document may be copied or reproduced in any form without the prior written consent of Altus Sistemas de Automação S.A. who reserves the right to carry out alterations without prior notice.

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

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

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

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

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

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

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

COPYRIGHTS

Nexto, Mastertool, Grano and WebPLC are the registered trademarks of Altus Sistemas de Automação S.A.

Windows, Windows NT and Windows Vista are registered trademarks of Microsoft Corporation.

OPEN SOURCE SOFTWARE NOTICE

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

Contents

1. Introduction . . . 1
   1.1. Nexto Series . . . 1
   1.2. Innovative Features . . . 2
   1.3. Documents Related to this Manual . . . 3
   1.4. Visual Inspection . . . 5
   1.5. Technical Support . . . 5
   1.6. Warning Messages Used in this Manual . . . 5
2. Technical Description . . . 6
   2.1. Panels and Connections . . . 6
   2.2. Product Features . . . 8
      2.2.1. General Features . . . 8
      2.2.2. Standards and Certifications . . . 9
      2.2.3. Memory . . . 10
      2.2.4. Protocols . . . 11
      2.2.5. Serial Interfaces . . . 12
         2.2.5.1. COM 1 . . . 12
      2.2.6. Ethernet Interfaces . . . 12
         2.2.6.1. NET 1 ... NET 6 . . . 12
      2.2.7. Redundancy Link . . . 13
         2.2.7.1. NET A / NET B . . . 13
      2.2.8. Memory Card Interface . . . 13
      2.2.9. Environmental Characteristics . . . 14
   2.3. Compatibility with Other Products . . . 14
   2.4. Performance . . . 15
      2.4.1. MainTask Interval Time . . . 15
      2.4.2. Application Times . . . 15
      2.4.3. Time for Instructions Execution . . . 15
      2.4.4. Initialization Times . . . 15
   2.5. Physical Dimensions . . . 16
   2.6. Purchase Data . . . 16
      2.6.1. Included Items . . . 16
      2.6.2. Product code . . . 17
   2.7. Related Products . . . 17
3. Installation . . . 18
   3.1. Mechanical Installation . . . 18
   3.2. Electrical Installation . . . 18
   3.3. Ethernet Network Connection . . . 19
      3.3.1. IP Address . . . 19

3.3.2. Gratuitous ARP . . . 20
3.3.3. Network Cable Installation . . . 21
3.4. Serial Network Connection RS-485 . . . 22
3.4.1. RS-485 Communication without termination . . . 22
3.4.2. RS-485 Communication with Internal Termination . . . 23
3.4.3. RS-485 Communication with External Termination . . . 24
3.4.4. Example of Connection of a RS-485 Network with External Termination and Master Redundancy . . . 25
3.5. Memory Card Installation . . . 25
3.6. Architecture Installation . . . 27
3.6.1. Module Installation on the Main Backplane Rack . . . 27
3.7. Programmer Installation . . . 27
4. Initial Programming . . . 28
4.1. Memory Organization and Access . . . 28
4.2. Project Profiles . . . 30
4.2.1. Fast Profile . . . 30
4.2.2. Process Profile . . . 30
4.2.3. Maximum Number of Tasks . . . 30
4.3. CPU Configuration . . . 31
4.4. Libraries . . . 32
4.5. Inserting a Protocol Instance . . . 32
4.5.1. MODBUS RTU . . . 32
4.5.2. MODBUS Ethernet . . . 34
4.6. Finding the Device . . . 35
4.7. Login . . . 37
4.8. Run Mode . . . 39
4.9. Stop Mode . . . 40
4.10. Writing and Forcing Variables . . . 40
4.11. Logout . . . 41
4.12. Project Upload . . . 41
4.13. CPU Operating States . . . 43
4.13.1. Run . . . 43
4.13.2. Stop . . . 43
4.13.3. Breakpoint . . . 43
4.13.4. Exception . . . 43
4.13.5. Reset Warm . . . 43
4.13.6. Reset Cold . . . 43
4.13.7. Reset Origin . . . 43
4.14. Programs (POUs) and Global Variable Lists (GVLs) . . . 44
4.14.1. MainPrg Program . . . 44
4.14.2. StartPrg Program . . . 44
4.14.3. UserPrg Program . . . 44
4.14.4. GVL System_Diagnostics . . . 44
4.14.5. GVL Disables . . . 45
4.14.6. GVL IOQualities . . . 46
4.14.7. GVL Module_Diagnostics . . . 47
4.14.8. GVL ReqDiagnostics . . . 48
5. Configuration . . . 51

5.1. Device . . . 51
5.1.1. User Management and Access Rights . . . 51
5.1.2. PLC Settings . . . 51
5.2. CPU Settings . . . 53
5.2.1. General Parameters . . . 53
5.2.1.1. Hot Swap . . . 54
5.2.1.1.1. Hot Swap Disabled, for Declared Modules Only . . . 54
5.2.1.1.2. Hot Swap Disabled . . . 55
5.2.1.1.3. Hot Swap Disabled, without Startup Consistency . . . 55
5.2.1.1.4. Hot Swap Enabled, with Startup Consistency for Declared Modules Only . . . 55
5.2.1.1.5. Hot Swap Enabled with Startup Consistency . . . 55
5.2.1.1.6. Hot Swap Enabled without Startup Consistency . . . 55
5.2.1.1.7. How to do the Hot Swap . . . 56
5.2.1.2. Retain and Persistent Memory Areas . . . 57
5.2.2. Time Synchronization . . . 58
5.2.2.1. SNTP . . . 59
5.2.2.2. Daylight Saving Time (DST) . . . 60
5.3. Serial Interface Configuration . . . 60
5.3.1. COM 1 . . . 60
5.3.1.1. Advanced Configurations . . . 61
5.4. Integrated Ethernet Interfaces Configuration . . . 61
5.4.1. Basic Ethernet Port Settings . . . 61
5.4.2. Ethernet Port Mode Configuration . . . 62
5.4.2.1. Simple Mode . . . 63
5.4.2.2. NIC Teaming Mode on the NX3035 CPU . . . 63
5.4.2.3. Switch Mode . . . 65
5.4.2.3.1. Switch Mode - Disabled . . . 65
5.4.2.3.2. Switch Mode - MRP . . . 65
5.4.2.3.3. Switch Mode - RSTP . . . 66
5.4.3. Ethernet Port Failure Mode Configuration . . . 68
5.4.4. Reserved TCP/UDP Ports . . . 69
5.4.5. Network Routing . . . 69
5.5. Configuration of Ethernet Interfaces with NX5000 Modules . . . 70
5.5.1. Basic Ethernet Port Settings for the NX5000 . . . 70
5.5.1.1. IP Addressing of Remote Ethernet Ports on the NX5000 (NET1) . . . 70
5.5.1.1.1. NET 1 . . . 70
5.5.2. Ethernet Port Mode Configuration for the NX5000 . . . 71
5.5.2.1. NIC Teaming Mode on the NX5000 . . . 71
5.5.3. Ethernet Port Failure Mode Configuration . . . 72
5.5.4. Other Settings of the NX5000 Module . . . 72
5.5.4.1. “Module Parameters” Tab . . . 72
5.5.4.2. “Bus I/O Mapping” Tab . . . 72
5.6. PROFIBUS Network Configuration with NX5001 Modules . . . 73
5.6.1. Inserting or Removing NX5001 Modules . . . 73
5.6.2. NX5001 Module Parameter Adjustment . . . 74
5.6.2.1. “General” Tab . . . 74
5.6.2.2. “Module Parameters” Tab . . . 75
5.6.2.3. “Bus I/O Mapping” Tab . . . 76

5.6.3. PROFIBUS Remote Settings . . . 76
5.6.3.1. Remote Stations Compatible with Redundant PROFIBUS Networks . . . 76
5.6.3.2. Watchdog Configuration on PROFIBUS Remote Stations in Redundant PLCs . . . 77
5.7. Protocols Configuration . . . 77
5.7.1. Protocol Behavior x CPU State . . . 79
5.7.2. MODBUS RTU Master . . . 80
5.7.2.1. MODBUS Master Protocol Configuration by Symbolic Mapping . . . 80
5.7.2.1.1. MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration . . . 80
5.7.2.1.2. Devices Configuration – Symbolic Mapping configuration . . . 82
5.7.2.1.3. Mappings Configuration – Symbolic Mapping Settings . . . 83
5.7.2.1.4. Requests Configuration – Symbolic Mapping Settings . . . 84
5.7.2.2. MODBUS Master Protocol Configuration for Direct Representation (%Q) . . . 87
5.7.2.2.1. General Parameters of MODBUS Master Protocol - setting by Direct Representation(%Q) . . . 87
5.7.2.2.2. Devices Configuration – Configuration for Direct Representation (%Q) . . . 88
5.7.2.2.3. Mappings Configuration – Configuration for Direct Representation (%Q) . . . 89
5.7.3. MODBUS RTU Slave . . . 91
5.7.3.1. MODBUS Slave Protocol Configuration via Symbolic Mapping . . . 91
5.7.3.1.1. MODBUS Slave Protocol General Parameters – Configuration via Symbolic Mapping . . . 91
5.7.3.1.2. Configuration of the Relations – Symbolic Mapping Setting . . . 94
5.7.3.2. MODBUS Slave Protocol Configuration via Direct Representation (%Q) . . . 95
5.7.3.2.1. General Parameters of MODBUS Slave Protocol – Configuration via Direct Representation (%Q) . . . 95
5.7.3.2.2. Mappings Configuration – Configuration via Direct Representation (%Q) . . . 96
5.7.4. MODBUS Ethernet . . . 98
5.7.5. MODBUS Ethernet Client . . . 100
5.7.5.1. MODBUS Ethernet Client Configuration via Symbolic Mapping . . . 100
5.7.5.1.1. MODBUS Client Protocol General Parameters – Configuration via Symbolic Mapping . . . 101
5.7.5.1.2. Device Configuration – Configuration via Symbolic Mapping . . . 101
5.7.5.1.3. Mappings Configuration – Configuration via Symbolic Mapping . . . 103
5.7.5.1.4. Requests Configuration – Configuration via Symbolic Mapping . . . 104
5.7.5.2. MODBUS Ethernet Client configuration via Direct Representation (%Q) . . . 107
5.7.5.2.1. General parameters of MODBUS Protocol Client - configuration for Direct Representation (%Q) . . . 107
5.7.5.2.2. Device Configuration – Configuration via Direct Representation (%Q) . . . 108
5.7.5.2.3. Mapping Configuration – Configuration via Direct Representation (%Q) . . . 109
5.7.5.3. MODBUS Client Relation Start in Acyclic Form . . . 111
5.7.6. MODBUS Ethernet Server . . . 112
5.7.6.1. MODBUS Server Ethernet Protocol Configuration for Symbolic Mapping . . . 112
5.7.6.1.1. MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping . . . 112
5.7.6.1.2. MODBUS Server Diagnostics – Configuration via Symbolic Mapping . . . 114
5.7.6.1.3. Mapping Configuration – Configuration via Symbolic Mapping . . . 114
5.7.6.2. MODBUS Server Ethernet Protocol Configuration via Direct Representation (%Q) 115
5.7.6.2.1. General Parameters of MODBUS Server Protocol – Configuration via Direct Representation (%Q) . . . 116
5.7.6.2.2. Mapping Configuration – Configuration via Direct Representation (%Q) . . . 117

5.7.7. OPC DA Server . . . 119
5.7.7.1. Creating a Project for OPC DA Communication . . . 120
5.7.7.2. Configuring a PLC on the OPC DA Server . . . 123
5.7.7.2.1. Importing a Project Configuration . . . 125
5.7.7.3. Configuration with the PLC on the OPC DA Server with Connection Redundancy . 125
5.7.7.4. OPC DA Communication Status and Quality Variables . . . 126
5.7.7.5. Limits of Communication with OPC DA Server . . . 128
5.7.7.6. Accessing Data Through an OPC DA Client . . . 128
5.7.8. OPC UA Server . . . 130
5.7.9. EtherCAT Master . . . 131
5.7.9.1. Installing and inserting EtherCAT Devices . . . 132
5.7.9.1.1. EtherCAT - Scan Devices . . . 132
5.7.9.2. EtherCAT Master Settings . . . 133
5.7.9.2.1. EtherCAT Master - General . . . 133
5.7.9.2.2. EtherCAT Master - Sync Unit Assignment . . . 135
5.7.9.2.3. EtherCAT Master - Overview . . . 135
5.7.9.2.4. EtherCAT Master - I/O Mapping . . . 135
5.7.9.2.5. EtherCAT Master - IEC Objects . . . 135
5.7.9.2.6. EtherCAT Master - Status / Information Tabs . . . 135
5.7.9.3. EtherCAT Slave Configuration . . . 135
5.7.9.3.1. EtherCAT Slave - General . . . 135
5.7.9.3.2. EtherCAT Slave - Process Data . . . 138
5.7.9.3.3. EtherCAT Slave - Edit PDO List . . . 140
5.7.9.3.4. EtherCAT Slave - Startup Parameters . . . 140
5.7.9.3.5. EtherCAT Slave - Module I/O Mapping . . . 140
5.7.9.3.6. EtherCAT Slave - Status and Information . . . 141
5.7.10. EtherNet/IP . . . 141
5.7.10.1. Ethernet Adapter . . . 142
5.7.10.2. EtherNet/IP Scanner Configuration . . . 144
5.7.10.2.1. General . . . 144
5.7.10.2.2. Connections . . . 145
5.7.10.2.3. Assemblies . . . 147
5.7.10.2.4. User-Defined Parameters . . . 148
5.7.10.2.5. EtherNet/IP I/O Mapping . . . 150
5.7.10.3. EtherNet/IP Adapter Configuration . . . 150
5.7.10.3.1. General . . . 150
5.7.10.3.2. EtherNet/IP Adapter: I/O Mapping . . . 150
5.7.10.4. EtherNet/IP Module Configuration . . . 151
5.7.10.4.1. Assemblies . . . 151
5.7.10.4.2. EtherNet/IP Module: I/O Mapping . . . 151
5.7.11. PROFINET Controller . . . 152
5.8. Communication Performance . . . 152
5.8.1. MODBUS Server . . . 152
5.8.1.1. CPU's Integrated Interfaces . . . 152
5.8.1.2. Remote Interfaces . . . 153
5.9. System Performance . . . 153
5.9.1. I/O Scan Time . . . 154
5.9.2. Memory Card . . . 154

5.10. RTC Clock . . . 154
5.10.1. Function Blocks for RTC Reading and Writing . . . 155
5.10.1.1. Function Blocks for RTC Reading . . . 155
5.10.1.1.1. GetDateAndTime . . . 156
5.10.1.1.2. GetTimeZone . . . 156
5.10.1.1.3. GetDayOfWeek . . . 157
5.10.1.2. RTC Writing Functions . . . 158
5.10.1.2.1. SetDateAndTime . . . 158
5.10.1.2.2. SetTimeZone . . . 159
5.10.2. RTC Data Structures . . . 160
5.10.2.1. EXTENDED_DATE_AND_TIME . . . 160
5.10.2.2. DAYS_OF_WEEK . . . 161
5.10.2.3. RTC_STATUS . . . 161
5.10.2.4. TIMEZONESETTINGS . . . 161
5.10.3. RTC Operating Limits . . . 162
5.11. User Files Memory . . . 162
5.12. CPU's Informative and Configuration Menu . . . 164
5.13. Function Blocks and Functions . . . 167
5.13.1. Special Function Blocks for Serial Interfaces . . . 167
5.13.1.1. SERIAL_CFG . . . 170
5.13.1.2. SERIAL_GET_CFG . . . 172
5.13.1.3. SERIAL_GET_CTRL . . . 173
5.13.1.4. SERIAL_GET_RX_QUEUE_STATUS . . . 174
5.13.1.5. SERIAL_PURGE_RX_QUEUE . . . 175
5.13.1.6. SERIAL_RX . . . 177
5.13.1.7. SERIAL_RX_EXTENDED . . . 178
5.13.1.8. SERIAL_TX . . . 180
5.13.2. Inputs and Outputs Update . . . 182
5.13.2.1. REFRESH_INPUT . . . 182
5.13.2.2. REFRESH_OUTPUT . . . 183
5.13.3. Timer Retain . . . 184
5.13.3.1. TOF_RET . . . 185
5.13.3.2. TON_RET . . . 186
5.13.3.3. TP_RET . . . 187
5.13.4. Non-Redundant Timer . . . 188
5.13.4.1. TOF_NR . . . 189
5.13.4.2. TON_NR . . . 189
5.13.4.3. TP_NR . . . 190
5.13.5. User Log . . . 190
5.13.5.1. UserLogAdd . . . 191
5.13.5.2. UserLogDeleteAll . . . 193
5.14. System Web Page . . . 193
5.14.1. General Overview . . . 194
5.14.2. Bus Information . . . 194
5.14.3. Management . . . 195
5.14.3.1. System . . . 196
5.14.3.1.1. Firmware Update . . . 196
5.14.3.1.2. Clock Setting . . . 197

5.14.3.1.3. SysLog . . . 197
5.14.3.1.4. System Control . . . 198
5.14.3.2. Network . . . 199
5.14.3.2.1. Configuration . . . 199
5.14.3.2.2. Sniffer . . . 201
5.14.3.3. SNMP . . . 202
5.14.3.3.1. SNMP on Nexto CPUs . . . 202
5.14.3.3.2. SNMP Configuration . . . 203
5.14.3.3.3. User and SNMP Communities . . . 204
5.14.3.4. Firewall . . . 204
5.14.3.4.1. General Configuration . . . 205
5.14.3.4.2. User Rules . . . 206
5.14.3.4.3. Switch Mode or NIC Teaming Mode . . . 208
5.14.3.4.4. Redundant architecture . . . 208
5.14.3.5. OpenVPN . . . 209
5.14.3.5.1. OpenVPN Configuration . . . 210
5.14.3.5.2. Security Files . . . 213
5.14.3.5.3. Status . . . 214
5.14.3.5.4. Files to Download . . . 215
5.14.3.5.5. Architectures Configuration . . . 216
5.14.3.6. FTP Server . . . 217
5.14.3.6.1. General Configuration . . . 218
5.14.3.6.2. User Configuration . . . 219
5.14.3.6.3. Status . . . 219
5.14.3.7. Memory Card . . . 219
5.14.3.7.1. Configuration . . . 221
6. Redundancy with NX3035 CPU . . . 225
6.1. Introduction . . . 225
6.2. Technical Description and Configuration . . . 226
6.2.1. Minimum Configuration of a Redundant CPU . . . 226
6.2.2. Synchronization Channels between PLCA and PLCB . . . 227
6.2.3. Typical Configurations of a Redundant CPU . . . 227
6.2.3.1. Adding NX5001 Modules for PROFIBUS Networks . . . 228
6.2.3.2. Adding NX5000 Modules for Ethernet Networks . . . 228
6.2.3.3. Adding I/O Modules . . . 229
6.2.4. General Characteristics of a Redundant CPU . . . 229
6.2.5. Purchase Data for Redundancy . . . 232
6.3. Principles of Operation . . . 232
6.3.1. Identification of an NX3035 CPU . . . 232
6.3.2. Single Redundant Project . . . 232
6.3.3. Features Configured on the WEB Page . . . 232
6.3.4. Redundant Project Structure . . . 233
6.3.4.1. Redundancy Template . . . 233
6.3.4.2. Single and Cyclic Task MainTask . . . 233
6.3.4.3. MainPrg Program . . . 233
6.3.4.4. ActivePrg Program . . . 234
6.3.4.5. NonSkippedPrg Program . . . 234
6.3.4.6. Redundant and Non-redundant Variables . . . 234

6.3.4.7. Redundant and Non-redundant %I Variables . . . 235
6.3.4.8. Redundant and Non-redundant %Q Variables . . . 235
6.3.4.9. Redundant and Non-redundant %M Variables . . . 238
6.3.4.10. Redundant and Non-Redundant Symbolic Variables . . . 239
6.3.4.11. Symbolic Variables Addressed with the AT Directive . . . 240
6.3.5. Data Structures for Diagnostics, Commands, and User Information . . . 240
6.3.6. Cyclic Synchronization Services through NETA and NETB . . . 240
6.3.6.1. Diagnostics and Commands Exchange . . . 241
6.3.6.2. Redundant Data Synchronization . . . 241
6.3.6.3. Redundant Forcing List Synchronization . . . 241
6.3.7. Sporadic Synchronization Services through NETA and NETB . . . 241
6.3.7.1. Project Synchronization . . . 242
6.3.8. Project Synchronization Disabling . . . 242
6.3.9. PROFIBUS Network Configuration . . . 243
6.3.10. Redundant Ethernet Networks with NIC Teaming . . . 244
6.3.11. IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup . . . 244
6.3.11.1. IP Address Configuration . . . 244
6.3.11.2. Ethernet Port Mode Configuration . . . 245
6.3.11.3. Ethernet Port Failure Mode Configuration . . . 245
6.3.12. IP Address Configuration on NX5000 Modules in a Redundant CPU Setup . . . 245
6.3.12.1. Fixed IP . . . 245
6.3.12.2. Exchange IP . . . 245
6.3.12.3. Active IP . . . 246
6.3.12.4. Multiple IP . . . 247
6.3.13. Combined Use of NIC Teaming and Switch Configurations with Active IP . . . 247
6.3.14. Ethernet Interfaces Use with Vital Fault Configuration . . . 248
6.3.14.1. Rapid Vital Failure Detection Through Link Loss on Ethernet Ports . . . 248
6.3.14.2. Vital Failure Detection Through Communication Failure with All MODBUS Servers 248
6.3.14.3. Keep-Alive Messages via NET1 and NET2 Between the Redundant CPs . . . 248
6.3.14.4. Using Additional Diagnostics for Vital Failure Maintenance . . . 249
6.3.14.5. Specific Behavior of the Active and Non-Active CPs . . . 249
6.3.15. Use of OPC DA Communication with Redundant Projects . . . 249
6.3.16. Redundant CPU States . . . 249
6.3.16.1. Active . . . 250
6.3.16.2. Standby . . . 250
6.3.16.3. Inactive . . . 250
6.3.16.4. Not-Configured . . . 250
6.3.16.5. Initializing . . . 250
6.3.17. Redundancy State Machine . . . 250
6.3.17.1. Not-Configured State . . . 251
6.3.17.1.1. Transition 1 – Not-Configured to Initializing . . . 252
6.3.17.2. Initializing State . . . 252
6.3.17.2.1. Transition 2 – Initializing to Not-Configured . . . 252
6.3.17.2.2. Transition 3 – Initializing to Inactive . . . 252
6.3.17.2.3. Transition 4 – Initializing to Active . . . 253
6.3.17.2.4. Transition 5 – Initializing to Standby . . . 253
6.3.17.3. Inactive State . . . 253

6.3.17.3.1. Transition 6 – Inactive to Not-Configured . . . 253
6.3.17.4. Active State . . . 253
6.3.17.4.1. Transition 7 – Active to Not-Configured . . . 254
6.3.17.4.2. Transition 8 – Active to Inactive . . . 254
6.3.17.4.3. Transition 9 – Active to Standby . . . 254
6.3.17.5. Standby State . . . 254
6.3.17.5.1. Transition 10 – Standby to Not-Configured . . . 254
6.3.17.5.2. Transition 11 – Standby to Inactive . . . 254
6.3.17.5.3. Transition 12 – Standby to Active . . . 255
6.3.18. First Instants in Active State . . . 255
6.3.19. Common Failures which Cause Automatic Switchovers between Half-Clusters . . . 255
6.3.20. Switch-overs between Half-Clusters Managed by the User . . . 256
6.3.21. Fault Tolerance . . . 256
6.3.21.1. Simple Failure with Unavailability . . . 257
6.3.21.2. Simple Failure without Unavailability Causing a Switchover . . . 257
6.3.21.3. Double Failure without Unavailability Causing a Switchover . . . 257
6.3.22. Redundancy Overhead . . . 257
6.4. Redundant CPU Programming . . . 258
6.4.1. Wizard for a New Redundant Project Creation . . . 258
6.4.2. Half-Clusters Configuration . . . 261
6.4.2.1. Fixed Configuration in the 0 to 5 Rack Positions . . . 261
6.4.3. Configuration of the Integrated Ethernet Ports of a Redundant NX3035 CPU (NET 1 to NET 6) . . . 261
6.4.3.1. IP Address Configuration . . . 261
6.4.3.2. NIC Teaming Configuration . . . 261
6.4.3.3. NX5001 Modules Configuration . . . 261
6.4.3.4. NX5000 Modules Configuration . . . 261
6.4.3.5. NX3035 CPU Configurations Related to Redundancy . . . 262
6.4.4. I/O Driver Configurations . . . 262
6.4.5. MainTask Configurations . . . 263
6.4.5.1. ActivePrg Program . . . 263
6.4.5.2. NonSkippedPrg Program . . . 264
6.4.6. *Redundancy Configuration* Object . . . 264
6.4.7. GVL *Module_Diagnostics* and GVL *System_Diagnostics* . . . 265
6.4.8. GVLs with Redundant or Non-Redundant Symbolic Variables . . . 265
6.4.9. GVLs with AT Directives . . . 265
6.4.10. Program-Type POUs with Redundant or Non-Redundant Symbolic Variables . . . 265
6.4.11. Breakpoints Utilization in Redundant Systems . . . 265
6.4.12. MODBUS Instances Managing in Redundant System . . . 265
6.4.13. Limitations on a Redundant PLC Programming . . . 266
6.4.13.1. Limitations in Redundant GVLs and POUs . . . 266
6.4.13.2. Limitations in Non-Redundant POUs . . . 266
6.4.14. Getting the Redundancy State of a Half-Cluster . . . 267
6.4.15. Reading Non-Redundant Diagnostics . . . 267
6.4.16. Example of Managing a MODBUS TCP Client in a Redundant CPU . . . 268
6.4.16.1. Configuration of NET3 / NET4 Interfaces in the Redundant CPU . . . 269
6.4.16.2. MODBUS TCP Server Configuration for Test Communication . . . 269
6.4.16.3. MODBUS TCP Client Configuration . . . 269

6.4.16.4. GVLs and POUs for Managing the MODBUS TCP Client . . . 270
6.5. Redundant CPU Program Downloading . . . 275
6.5.1. Initial Downloading of a Redundant Project . . . 275
6.5.2. Mastertool Connection with an NX3035 CPU in a Redundant CPU Setup . . . 278
6.5.3. Modification Download in a Redundant Project . . . 279
6.5.4. Offline and Online Modification Download . . . 279
6.5.4.1. Modifications which Demand Offline Download and the Interruption of the Process Control . . . 279
6.5.4.2. Modifications which Demand Offline Download . . . 280
6.5.4.3. Modifications which Allow Online Download . . . 280
6.5.5. Online Download of Modifications . . . 280
6.5.6. Offline Download of Modifications with Process Control Interruption . . . 281
6.6. Redundancy Maintenance . . . 282
6.6.1. Hot Swapping of Modules in a Redundant CPU . . . 282
6.6.2. Mastertool Warning Messages . . . 283
6.6.2.1. Blocking the Loading of a Redundant or Non-Redundant Project . . . 283
6.6.2.2. Alerts Before Commands That Can Stop the Active CPU . . . 283
6.6.2.3. Alert Before Logging into the Non-Active CPU . . . 284
6.6.3. Redundancy Diagnostics on the NX3035 CPU Graphical Display . . . 284
6.6.3.1. CPU Redundancy Status . . . 284
6.6.3.2. Screens Below the REDUNDANCY Menu . . . 284
6.6.4. Structure of Diagnostics, Commands, and User Information for Redundancy . . . 285
6.6.4.1. Redundancy Diagnostics . . . 285
6.6.4.2. Redundancy Commands . . . 291
6.6.4.3. User Information Exchanged between Local and Remote CPUs . . . 293
6.6.4.4. MODBUS Diagnostics Used in Redundancy . . . 293
6.6.4.5. Redundancy Event Log . . . 293
7. Maintenance . . . 295
7.1. Module Diagnostics . . . 295
7.1.1. One Touch Diag . . . 295
7.1.2. Diagnostics via LED . . . 297
7.1.2.1. DG (Diagnostic) . . . 297
7.1.2.2. WD (Watchdog) . . . 297
7.1.2.3. RJ45 Connector LEDs . . . 298
7.1.3. Diagnostics via System Web Page . . . 298
7.1.4. Diagnostics via Variables . . . 298
7.1.4.1. Summarized Diagnostics . . . 299
7.1.4.2. Detailed Diagnostics . . . 301
7.1.4.2.1. Target Detailed Diagnostics Group . . . 301
7.1.4.2.2. Hardware Detailed Diagnostics Group . . . 301
7.1.4.2.3. Exception Detailed Diagnostics Group . . . 302
7.1.4.2.4. RetainInfo Detailed Diagnostics Group . . . 302
7.1.4.2.5. Reset Detailed Diagnostics Group . . . 303
7.1.4.2.6. Thermometer Detailed Diagnostics Group . . . 303
7.1.4.2.7. Serial Detailed Diagnostics Group . . . 303
7.1.4.2.8. Ethernet Detailed Diagnostics Group . . . 304
7.1.4.2.9. UserFiles Detailed Diagnostics Group . . . 305
7.1.4.2.10. UserLogs Detailed Diagnostics Group . . . 306

7.1.4.2.11. MemoryCard Detailed Diagnostics Group . . . 306
7.1.4.2.12. WHSB Detailed Diagnostics Group . . . 306
7.1.4.2.13. Application Detailed Diagnostics Group . . . 307
7.1.4.2.14. SNTP Detailed Diagnostics Group . . . 307
7.1.4.2.15. Rack Detailed Diagnostics Group . . . 308
7.1.4.2.16. ApplicationInfo Detailed Diagnostics Group . . . 308
7.1.4.2.17. Firewall Detailed Diagnostics Group . . . 308
7.1.4.2.18. OpenVPN Detailed Diagnostics Group . . . 309
7.1.4.2.19. FTP Detailed Diagnostics Group . . . 309
7.1.5. Diagnostics via Function Blocks . . . 309
7.1.5.1. GetTaskInfo . . . 310
7.2. Graphic Display . . . 311
7.3. System Log . . . 313
7.4. Not Downloading the Application at Startup . . . 313
7.5. Power Failure . . . 313
7.6. Most Common Problems . . . 314
7.7. Troubleshooting . . . 314
7.8. Preventive Maintenance . . . 315
8. Appendixes . . . 316
8.1. TLS Key and Certificate Management . . . 316
8.1.1. Easy-RSA Certificate Generation . . . 316
8.1.2. OpenSSL Certificate Generation . . . 321
8.1.3. TA Key Generation by OpenVPN . . . 323
8.2. Offline Program Download Without Interrupting Process Control . . . 323
8.2.1. General Recommendations . . . 323
8.2.2. PROFIBUS Changes . . . 324
8.2.3. Initial Condition . . . 324
8.2.4. Change Procedure . . . 325
8.2.5. Steps . . . 325
8.2.5.1. Step 1 – Preparation for the Change . . . 325
8.2.5.2. Step 2 – Download to the Standby CPU . . . 325
8.2.5.3. Step 3 – Control Handover . . . 325
8.2.5.4. Step 4 – Final Validation . . . 325

1. Introduction

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

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

With 64-bit processing and extensive memory capacity, the NX3035 is designed to meet applications demanding high performance, availability, and reliability. Featuring 384 Kbytes for %I and %Q points and 8 Mbytes of retentive or persistent memory, the CPU supports robust applications, with performance ensured by its integrated floating-point unit. Equipped with six high-speed Ethernet interfaces, two SFP interfaces for redundancy synchronization, and an RS-485 serial interface, the NX3035 guarantees reliable connectivity and redundancy support, making it ideal for critical systems. Its compatibility with protocols such as OPC DA/UA, PROFINET, EtherCAT, MODBUS TCP, SNMP, and EtherNet/IP allows seamless integration with different platforms and devices. Features like clock synchronization via SNTP and a real-time clock ensure greater precision. Combining a compact design and advanced diagnostics, its support for microSD cards enhances storage flexibility, while the redundancy mode strengthens availability for continuous operations.



Figure 1: NX3035

1.1. Nexto Series

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

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

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



Figure 2: Nexto Series – Overview

1.2. Innovative Features

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

VPN: Nexto products have an embedded VPN service, which creates a private tunnel that connects directly to the CPU. This functionality, available on some models of the family, allows accessing a control network remotely and completely securely..

FTP: Supporting FTP-type connections, the series equipment is able to exchange data with a server that uses this same technology model. This functionality allows the files generated by the controller, such as logs collected through a datalogger function, to be accessed remotely.

Linux: Another innovative feature of the series is its embedded Linux platform. The feature makes possible the virtualization of software developed for operating systems with Unix technology. The feature gives more versatility and speed to the operation of the system, as it allows the processing of multiple data within the CPU itself.

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

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

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

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

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

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

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

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

1.3. Documents Related to this Manual

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

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

The following documents are recommended as additional sources of information:

Code	Description	Language
CE114000	Nexto Series – Technical Characteristics	English
CT114000	Série Nexto – Características Técnicas	Portuguese
CE114108	NX3035 Technical Characteristics	English
CT114108	Características Técnicas NX3035	Portuguese
CE114200	NX8000 Power Supply Module Technical Characteristics	English
CT114200	Características Técnicas Fonte de Alimentação NX8000	Portuguese
CE114700	Nexto Series Backplane Racks Technical Characteristics	English
CT114700	Características Técnicas dos Bastidores da Série Nexto	Portuguese
CE114810	Nexto Series Accessories for Backplane Rack Technical Characteristics	English
CT114810	Características Técnicas Acessórios para Bastidor Série Nexto	Portuguese
CE114902	Nexto Series PROFIBUS-DP Master Technical Characteristics	English
CT114902	Características Técnicas do Mestre PROFIBUS-DP da Série Nexto	Portuguese
CE114903	Nexto Series Ethernet Module Technical Characteristics	English
CT114903	Características Técnicas Módulo Ethernet Série Nexto	Portuguese
CE114908	NX5110 and NX5210 PROFIBUS-DP Heads Technical Characteristics	English
CT114908	Características Técnicas Interfaces Cabeça PROFIBUSDP NX5110 e NX5210	Portuguese
CE157204	NX9500 / NX9501 Technical Characteristics	English
CT157204	Características Técnicas NX9500 / NX9501	Portuguese
MU214600	Nexto Series User Manual	English
MU214000	Manual de Utilização Série Nexto	Portuguese
MU214619	NX3035 CPU User Manual	English
MU214107	Manual de Utilização UCP NX3035	Portuguese
MU299611	Mastertool X User Manual	English

Code	Description	Language
MU299049	Manual de Utilização Mastertool X	Portuguese
MP399609	IEC 61131 Programming Manual	English
MP399048	Manual de Programação IEC 61131	Portuguese
MU214601	NX5001 PROFIBUS DP Master User Manual	English
MU214001	Manual de Utilização Mestre PROFIBUS-DP NX5001	Portuguese
MU214608	Nexto PROFIBUS-DP Head Utilization Manual	English
MU214108	Manual de Utilização da Cabeça PROFIBUS-DP Nexto	Portuguese
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
MU214603	Nexto Series HART Manual	English
MU214606	MQTT User Manual	English
MU214609	OPC UA Server for Altus Controllers User Manual	English
MU214610	PID - Advanced Control Functions User Manual	English
MU214621	Nexto Series PROFINET Manual	English
MU214622	LibSparkPlug User Manual	English
NAP151	Utilização do Tunneller OPC	Portuguese
NAP169	RSTP in Nexto CPUs	English

Table 1: Related Documents

1.4. Visual Inspection

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

CAUTION

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

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

1.5. Technical Support

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

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

1.6. Warning Messages Used in this Manual

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

DANGER

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

CAUTION

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

ATTENTION

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

2. Technical Description

This chapter presents all the technical features of the Nexto Series NX3035 CPU.

2.1. Panels and Connections

The following figure shows the CPU front panel.



Figure 3: NX3035

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

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

LED	Description
DG	Diagnostics LED
WD	Watchdog LED

Table 2: LEDs Description

Nexto Series CPUs has two switches available to the user. The table below shows the description of these switches. For further information regarding the diagnostics switch, see sections One Touch Diag.

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.

Table 3: Keys Description

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

Interfaces	Description
NET 1	RJ45 connector for communication using the 10/100/1000Base-TX standard. Enables point-to-point or network communication. For more information on usage, see section Integrated Ethernet Interfaces Configuration.
NET 2	RJ45 connector for communication using the 10/100/1000Base-TX standard. Enables point-to-point or network communication. For more information on usage, see section Integrated Ethernet Interfaces Configuration.
NET 3	RJ45 connector for communication using the 10/100/1000Base-TX standard. Enables point-to-point or network communication. For more information on usage, see section Integrated Ethernet Interfaces Configuration.
NET 4	RJ45 connector for communication using the 10/100/1000Base-TX standard. Enables point-to-point or network communication. For more information on usage, see section Integrated Ethernet Interfaces Configuration.
NET 5	RJ45 connector for communication using the 10/100/1000Base-TX standard. Enables point-to-point or network communication. For more information on usage, see section Integrated Ethernet Interfaces Configuration.
NET 6	RJ45 connector for communication using the 10/100/1000Base-TX standard. Enables point-to-point or network communication. For more information on usage, see section Integrated Ethernet Interfaces Configuration.
NET A	SFP connector for communication over the redundancy link. (bottom part of the product.)
NET B	SFP connector for communication over the redundancy link. (bottom part of the product.)
COM 1	Female DB9 connector for communication using the RS-485 standard. For more information on usage, see section Serial Interface Configuration.
MEMORY SLOT	Memory card connector. Allows the use of a memory card for different types of data storage such as user logs, web pages, project documentation, and source files. For more information on usage, see section Memory Card.

Table 4: Connection Interfaces

2.2. Product Features

2.2.1. General Features

	NX3035
Backplane rack occupation	4 sequential slots
Current consumption from backplane rack	2600 mA
Power supply integrated	No
Ethernet TCP/IP integrated interface	6
Serial Interface	1
CAN Interface	No
USB Port Host	No
Memory Card Interface	1
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 System Web Page CPU internal memory
Programming languages	Structured Text (ST) Ladder Diagram (LD) Sequential Function Chart (SFC) Function Block Diagram (FBD) Continuous Function Chart (CFC)
Tasks	Cyclic (periodic) Event (software interrupt) Freewheeling (continuous) Status (software interrupt)
Online changes	Yes
Maximum number of tasks	32
Maximum number of rack expansion	24
Rack expansion redundancy support	Yes
Maximum number of I/O modules on the CPU local bus	128
Maximum number of additional Ethernet TCP/IP interface modules (NX5000)	6
Ethernet TCP/IP interface redundancy support	Yes
Maximum number of PROFIBUS-DP networks (using master modules PROFIBUS-DP)	6
PROFIBUS-DP network redundancy support	Yes
Redundancy support (half-clusters)	Yes
Hot Swap support	Yes
Event oriented data reporting (SOE) **Protocol** **Maximum Event Queue Size**	No - -
User web pages (Webvisu)	No
Firewall	Yes

	NX3035
VPN	Yes
FTP	Yes
Docker	No
One Touch Diag (OTD)	Yes
Electronic Tag on Display (ETD)	Yes

Table 5: General Features

Notes:

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

Maximum number of I/O modules: The maximum number of I/O modules refers to the sum of all modules on the local bus and rack expansions. Additional I/O modules can be installed in remote I/O systems (PROFIBUS, MODBUS, PROFINET, Ethernet/IP, etc).

2.2.2. Standards and Certifications

Standards and Certifications	
IEC	61131-2: Industrial-process measurement and control - Programmable controllers - Part 2: Equipment requirements and tests 61131-3: Programmable controllers - Part 3: Programming languages
	DNV Type Approval – DNV-CG-0339 (TAA000013D)
	2014/30/EU (EMC) 2014/35/EU (LVD) 2011/65/EU and 2015/863/EU (ROHS)
	S.I. 2016 No. 1091 (EMC) S.I. 2016 No. 1101 (Safety) S.I. 2012 No. 3032 (ROHS)
	TR 004/2011 (LVD) CU TR 020/2011 (EMC)

Table 6: Standards and Certifications

2.2.3. Memory

	NX3035
Addressable input variables memory (%I)	384 Kbytes
Addressable output variables memory (%Q)	384 Kbytes
Direct representation variable memory (%M)	128 Kbytes
Symbolic variable memory	48 Mbytes
Persistent or Retain symbolic variables memory	8 Mbytes
Full Redundant Data Memory	2912 Kbytes
Direct representation input variable memory (%I)	368 Kbytes
Direct representation output variable memory (%Q)	368 Kbytes
Direct representation variable memory (%M)	128 Kbytes
Symbolic variable memory	2912 Kbytes
Program memory	64 Mbytes
Source code memory (backup)	256 Mbytes
User files memory	2 Gbytes

Table 7: Memory

Notes:

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

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

This Nexto Series CPU model allows the definition of an area of redundant variables inserted within the direct representation output variable memory area %Q. The subset of direct representation output variable memory types are part of the total memory available.

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

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

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

ATTENTION

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

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

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

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

Table 8: Variables Behavior after the Event

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

Total redundant data memory: *Total redundant data memory* is the maximum amount of memory that can be used as redundant memory between two CPUs that form the redundant pair. This value is not a different memory. Note that the sum of all redundant variables (*Direct representation input variable memory*, *Direct representation output variable memory*, *Direct representation variable memory*, and *Symbolic variable memory*) must be equal to or less than the *Total redundant data memory*.

Program memory: Memory area corresponding to the maximum size allowed for the user application. This area is shared with the source code memory and compilation data, the total area being the sum of (Program memory + Source code memory + Compilation memory).

Source code memory (backup): Memory area used as a backup for the project, i.e., if the user wishes to import their project, the Mastertool software will retrieve the necessary information from this area. It is important to note that the user must remember to update the project that is saved as a backup every time they send the application, to prevent information from being lost. This area is shared with the program memory, with the total area being the sum of Program memory + Source code memory.

User file memory: This area of memory is intended for storing files such as doc, pdf, images, among others; that is, it allows data to be recorded as if it were a memory card. The usable capacity may be less than the specified nominal capacity due to system files and formatting. For more information, see the chapter Configuration - User File Memory.

2.2.4. Protocols

	NX3035 Simple PLC	NX3035 Redundant PLC	Interface
Open Protocol	Yes	Yes	COM1
MODBUS RTU Master	Yes	Yes	COM1
MODBUS RTU Slave	Yes	Yes	COM1
MODBUS TCP Client	Yes	Yes	NET1 ... NET6
MODBUS TCP Server	Yes	Yes	NET1 ... NET6
MODBUS RTU over TCP Client	Yes	Yes	NET1 ... NET6
MODBUS RTU over TCP Server	Yes	Yes	NET1 ... NET6
CANopen Master	No	No	-
CANopen Slave	No	No	-
CAN low level	No	No	-
SAE J-1939	No	No	-

	NX3035 Simple PLC	NX3035 Redundant PLC	Interface
OPC DA Server	Yes	Yes	NET1 ... NET6
OPC UA Server	Yes	No	NET1 ... NET6
EtherCAT Master	Yes	No	NET1 ... NET6
SNMP Agent	Yes	Yes	NET1 ... NET6
SOE (Event-oriented data)	No	No	-
DNP3 Server	No	No	-
IEC 60870-5-104 Server	No	No	-
EtherNet/IP Scanner	Yes	No	NET1 ... NET6
EtherNet/IP Adapter	Yes	No	NET1 ... NET6
MQTT Client	Yes	Yes	NET1 ... NET6
SparkPlugB	Yes	Yes	NET1 ... NET6
SNTP Client (for clock synchronism)	Yes	Yes	NET1 ... NET6
PROFINET Controller	Yes	No	NET1 ... NET6
PROFINET Device	No	No	-
OpenVPN Client	Yes	Yes	NET1 ... NET6
OpenVPN Server	Yes	Yes	NET1 ... NET6
FTP Server	Yes	Yes	NET1 ... NET6
RSTP	Yes	Yes	NET1 ... NET6
MRP (only with the PROFINET Controller)	Yes	No	NET1 ... NET6
SysLog	Yes	Yes	NET1 ... NET6

Table 9: Protocols

2.2.5. Serial Interfaces

2.2.5.1. COM 1

	COM 1
Connector	Shielded female DB9
Physical interface	RS-485
Communication direction	half duplex
RS-485 max. transceivers	32
Termination	Yes (optional through parameter)
Baud rate	9600, 19200, 38400, 57600, 115200 bps
Isolation	
Logic to Serial Port	1000 Vac / 1 minute
Serial Port to protection eartho ⏚	1000 Vac / 1 minute

Table 10: COM 1 Serial Interface Features

Note:
Physical interface: The list of cables can be found in the section Related Products.
RS-485 maximum transceivers: It is the maximum number of RS-485 interfaces that can be used on the same bus.

2.2.6. Ethernet Interfaces

2.2.6.1. NET 1 ... NET 6

	NET 1 / NET 2 / NET 3 / NET 4 / NET 5 / NET 6
Connector	Shielded female RJ45
Auto crossover	Yes
Maximum cable length	100 m
Cable type	UTP or ScTP, Category 5e or higher
Baud rate	10/100/1000 Mbps
Physical layer	10BASE-TE/100BASE-TX/1000BASE-T
Data link layer	LLC (Logical Link Control)
Network layer	IP (Internet Protocol)
Transport layer	TCP (Transmission Control Protocol) UDP (User Datagram Protocol)
Diagnostics	LED - green 1000 Mbps (link/activity) LED – yellow 100 Mbps (link/activity) LEDs – green and yellow 10 Mbps (link/activity)
Isolation **Ethernet interface to logic and earth** **Ethernet interface to Ethernet interface**	 1500 Vac / 1 minute 1500 Vac / 1 minute

Table 11: NET 1 to NET 6 Interface Features

The NET 1 Interface is the interface used for programming using the Mastertool tool.

2.2.7. Redundancy Link

2.2.7.1. NET A / NET B

	NET A / NET B
Connector	SFP receptacle
Internal data rate	1,25 Gbps
Over current protection	Yes
Hot swap	Yes

Table 12: NET A and NET B Redundancy Link Features

Note:
The list of compatible SFP transceivers can be found in the section Related Products.

2.2.8. Memory Card Interface

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

	Memory Card
Maximum Capacity	32 Gbytes
Minimum Capacity	2 Gbytes
Type	MicroSD
File System	FAT32
Remove card safely	Yes, through a specific menu for this function.

	Memory Card

Table 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.

2.2.9. Environmental Characteristics

	NX3035
Dissipation	11 W
Operating temperature	0 to 60 °C
Storage temperature	-25 to 75 °C
Relative humidity	5% to 96%, non-condensing
Conformal coating	Yes
IP Level	IP 20
Module dimensions (W x H x D)	72.20 x 114.63 x 115.30 mm
Package dimensions (W x H x D)	77 x 119 x 145 mm
Weight	490 g
Weight with package	620 g

Table 14: Environmental Characteristics

Notes:

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

2.3. Compatibility with Other Products

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

Nexto Series CPUs	Mastertool	Firmware version
NX3035	4.0.0 or above	1.14.0.0 or above

Table 15: Compatibility with other products

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

2.4. Performance

The Nexto Series CPUs performance relies on:

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

2.4.1. MainTask Interval Time

The MainTask interval time configuration depends on the selected project profile. For the Process profile, the interval can be configured with values from 1 ms to 750 ms. For the Fast profile, the interval can be configured with values from 1 ms to 100 ms.

2.4.2. Application Times

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

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

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

2.4.3. Time for Instructions Execution

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

Instruction	Language	Variable Type	Time (μs)
1000 Contacts	LD	BOOL	1.05
1000 Divisions	LD, ST	INT	4.0
		REAL	2.3
1000 Multiplications	LD, ST	INT	2.3
		REAL	2.3
1000 Sums	LD, ST	INT	2.3
		REAL	2.3

Table 16: Instruction Times

2.4.4. Initialization Times

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

2.5. Physical Dimensions

Dimensions in mm.

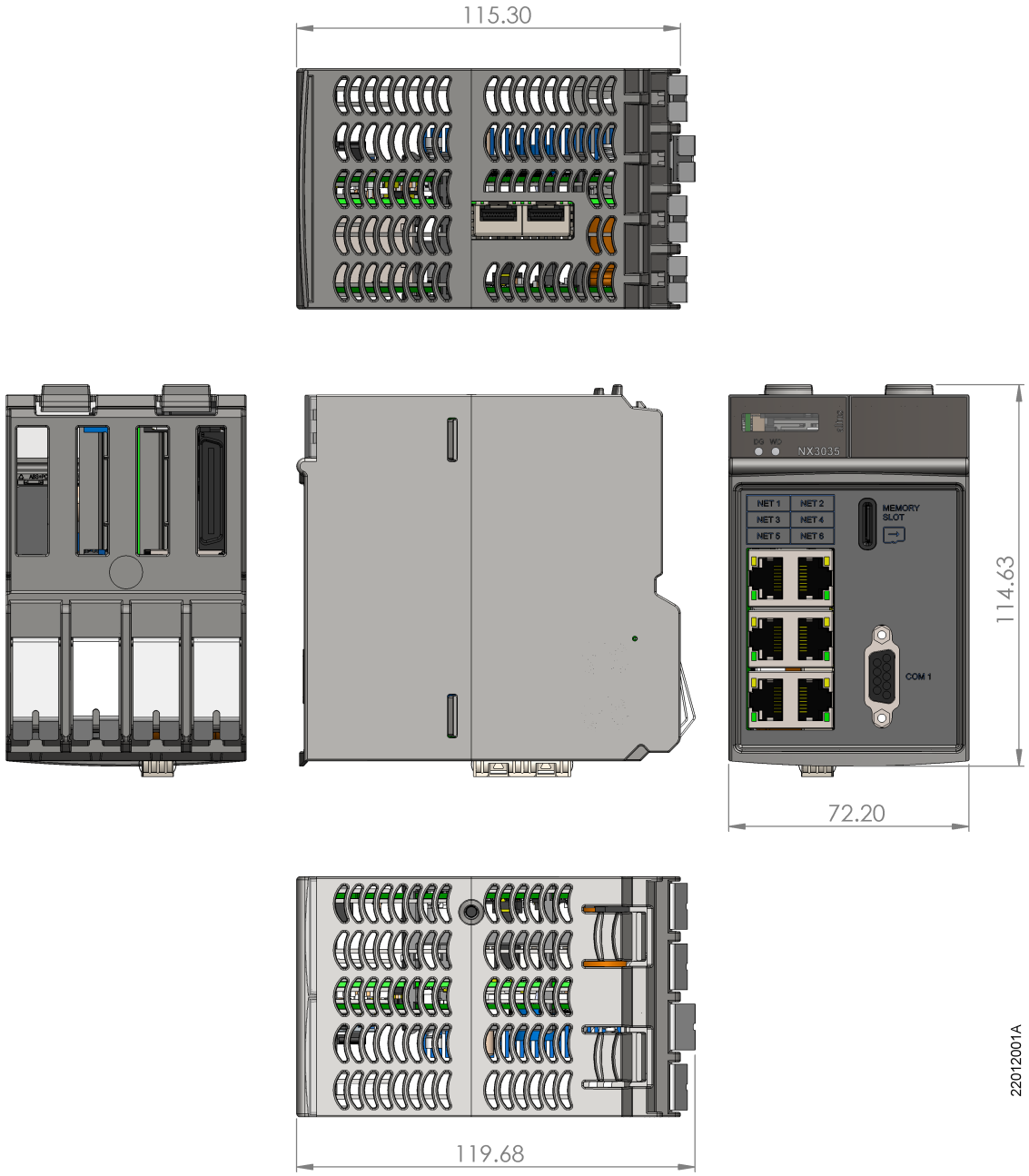


Figure 4: Physical Dimensions

2.6. Purchase Data

2.6.1. Included Items

The product package contains the following items:

- NX3035 module

2.6.2. Product code

The following code should be used to purchase the product:

Code	Description
NX3035	High-speed CPU, 6 Ethernet ports, 2 SFP ports, 1 serial channel, memory card interface, remote rack expansion and redundancy support

Table 17: Product Code

2.7. Related Products

The following products must be purchased separately when necessary:

Code	Description
MT9000	Mastertool X
AL-2600	RS-485 network branch and terminator
AL-2306	RS-485 cable for MODBUS or CAN network
AL-2319	RJ45-RJ45 Cable
AL-1763	CMDB9-Terminal Block Cable
NX9101	32 GB microSD memory card with miniSD and SD adapters
NX9202	RJ45-RJ45 2 m Cable
NX9205	RJ45-RJ45 5 m Cable
NX9210	RJ45-RJ45 10 m Cable
NX9000	8-Slot Backplane Rack
NX9001	12-Slot Backplane Rack
NX9002	16-Slot Backplane Rack
NX9003	24-Slot Backplane Rack
NX8000	30 W 24 Vdc Power Supply Module
NX9500	Gigabit SFP multimode fiber transceiver (550m)
NX9501	Gigabit SFP single-mode fiber transceiver (10Km)

Table 18: Related Products

Notes:

AL-2600: This module is used for branch and termination of RS-422/485 networks. For each network node, an AL-2600 is required. The AL-2600 that is at the ends of network must be configured with termination, except when there is a device with active internal termination, the rest must be configured without termination.

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

AL-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-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 for Ethernet communication and to interconnect the rack expansion modules.

3. Installation

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

CAUTION

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

3.1. Mechanical Installation

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

3.2. Electrical Installation

DANGER

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

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

The figure below shows the Nexto Series CPU's electric diagram installed in a Nexto Series backplane rack.

The connectors placement depicted are merely illustrative.

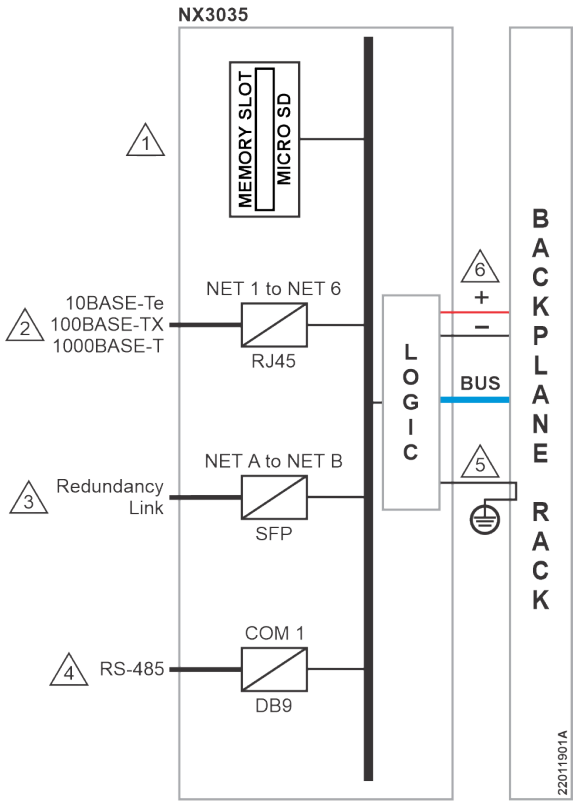


Figure 5: Electrical Diagram of the NX3035 CPU

Diagram Notes:

1. MicroSD interface
2. Standard Ethernet interfaces 10BASE-Te/100BASE-TX/1000BASE-T.
3. Redundancy Link.
4. RS-485 Interface.
5. The CPU is grounded through the Nexto Series backplane racks.
6. The module's power supply comes from the backplane rack connection, requiring no external connections.

3.3. Ethernet Network Connection

The isolated communication interfaces NET 1 through NET 6 enable connection to an Ethernet network, with NET 1 being the most suitable interface for communication with Mastertool.

The Ethernet network connection uses twisted-pair cables (10BASE-Te/100BASE-TX/1000BASE-T), and the speed is automatically detected by the Nexto CPU. This cable must have one of its endings connected to the interface that is likely to be used and another one to the HUB, switch, microcomputer or other Ethernet network point.

3.3.1. IP Address

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

	NET 1
IP Address	192.168.15.1
Subnetwork Mask	255.255.255.0
Gateway Address	192.168.15.253

Table 19: Default Parameters Configuration for Ethernet NET 1 Interface

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

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

The NET 2 Ethernet interface is used only for Ethernet communication and comes with the following default parameters configuration:

	NET 2
IP Address	192.168.16.1
Subnetwork Mask	255.255.255.0
Gateway Address	0.0.0.0

Table 20: 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 CPU's Informative and Configuration Menu section.

The NET 2 interface network parameters can be changed through Mastertool. For further information regarding configuration and parameters modifications, see Integrated Ethernet Interfaces Configuration section.

	NET 3
IP Address	192.168.17.1
Subnetwork Mask	255.255.255.0
Gateway Address	0.0.0.0

	NET 3

Table 21: Default Parameters Configuration for Ethernet NET 3 Interface

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

The NET 3 interface network parameters can be changed through Mastertool. For further information regarding configuration and parameters modifications, see Integrated Ethernet Interfaces Configuration section.

	NET 4
IP Address	192.168.18.1
Subnetwork Mask	255.255.255.0
Gateway Address	0.0.0.0

Table 22: Default Parameters Configuration for Ethernet NET 4 Interface

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

The network parameters of the NET 4 interface can be modified through Mastertool. For more details on configuring and changing network parameters, see section Integrated Ethernet Interfaces Configuration.

	NET 5
IP Address	192.168.19.1
Subnetwork Mask	255.255.255.0
Gateway Address	0.0.0.0

Table 23: Default Parameters Configuration for Ethernet NET 5 Interface

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

The network parameters of the NET 5 interface can be modified through Mastertool. For more details on configuring and changing network parameters, see section Integrated Ethernet Interfaces Configuration.

	NET 6
IP Address	192.168.20.1
Subnetwork Mask	255.255.255.0
Gateway Address	0.0.0.0

Table 24: Default Parameters Configuration for Ethernet NET 6 Interface

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

The network parameters of the NET 6 interface can be modified through Mastertool. For more details on configuring and changing network parameters, see section Integrated Ethernet Interfaces Configuration.

3.3.2. Gratuitous ARP

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

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

3.3.3. Network Cable Installation

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

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

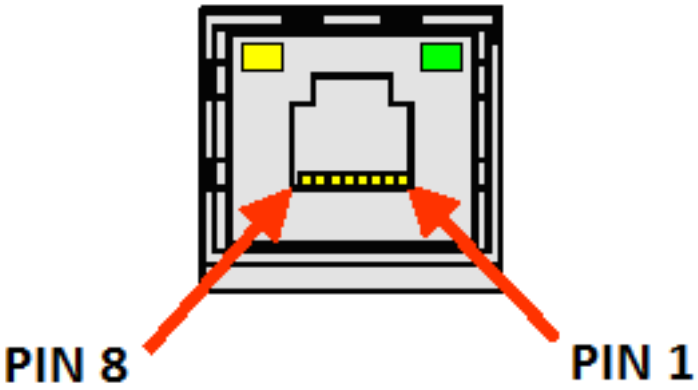


Figure 6: RJ45 Female Connector

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

Table 25: RJ45 Female Connector Pinout - 10BASE-TE and 100BASE-TX

The table below shows the female RJ45 connector of Nexto CPUs, with the identification and description of the valid pinout for the 1GBASE-T physical level.

Pin	Signal	Description
1	TxRx A +	Bidirectional, positive
2	TxRx A -	Bidirectional, negative
3	TxRx B +	Bidirectional, positive
4	TxRx C +	Bidirectional, positive
5	TxRx C -	Bidirectional, negative
6	TxRx B -	Bidirectional, negative
7	TxRx D +	Bidirectional, positive
8	TxRx D -	Bidirectional, negative

Table 26: RJ45 Female Connector Pinout - 1GBASE-T

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

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

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

3.4. Serial Network Connection RS-485

The isolated COM 1 communication interface enables connection to an RS-485 network and uses a female DB9 connector for the serial interface, whose pin identification and signal description are shown below.

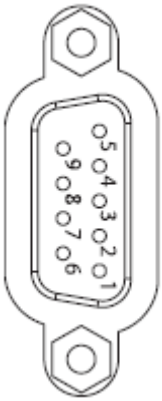


Figure 7: DB9 Female Connector

Pin	Signal	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 27: COM 1

3.4.1. RS-485 Communication without termination

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

AL-1763 terminals	Device terminal signals
0	Shield

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

Table 28: RS-485 Connections without Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

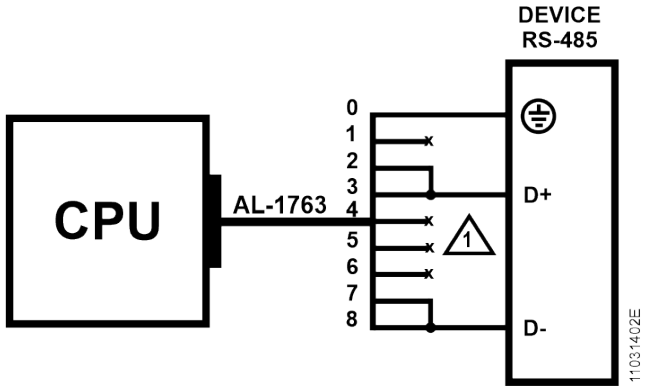


Figure 8: RS-485 Connections without Termination Diagram

Diagram Note:

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

3.4.2. RS-485 Communication with Internal Termination

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

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

Table 29: RS-485 Connections with Internal Termination

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

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

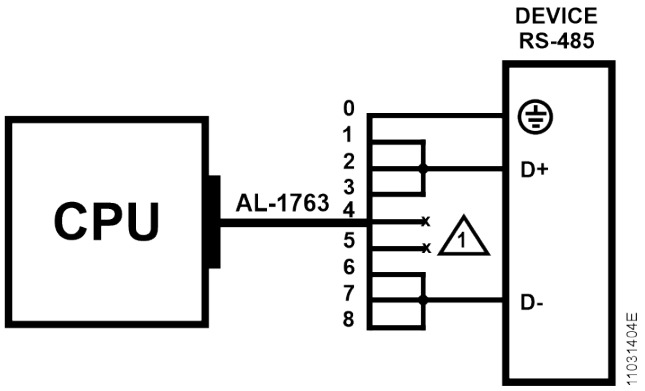


Figure 9: RS-485 Connections with Internal Termination Diagram

Diagram Note:

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

3.4.3. RS-485 Communication with External Termination

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

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

Table 30: RS-485 Connections with External Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

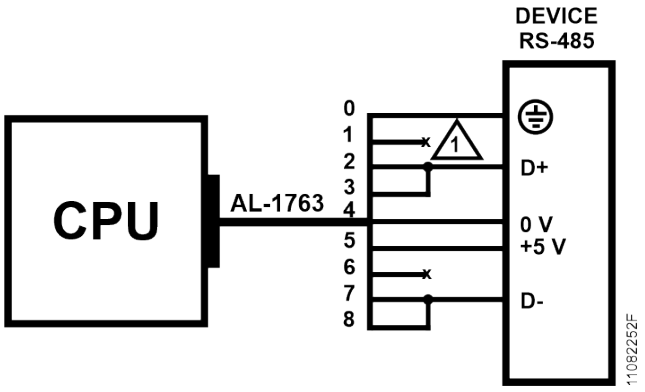


Figure 10: RS-485 Connections with External Termination Diagram

Diagram Note:

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

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

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

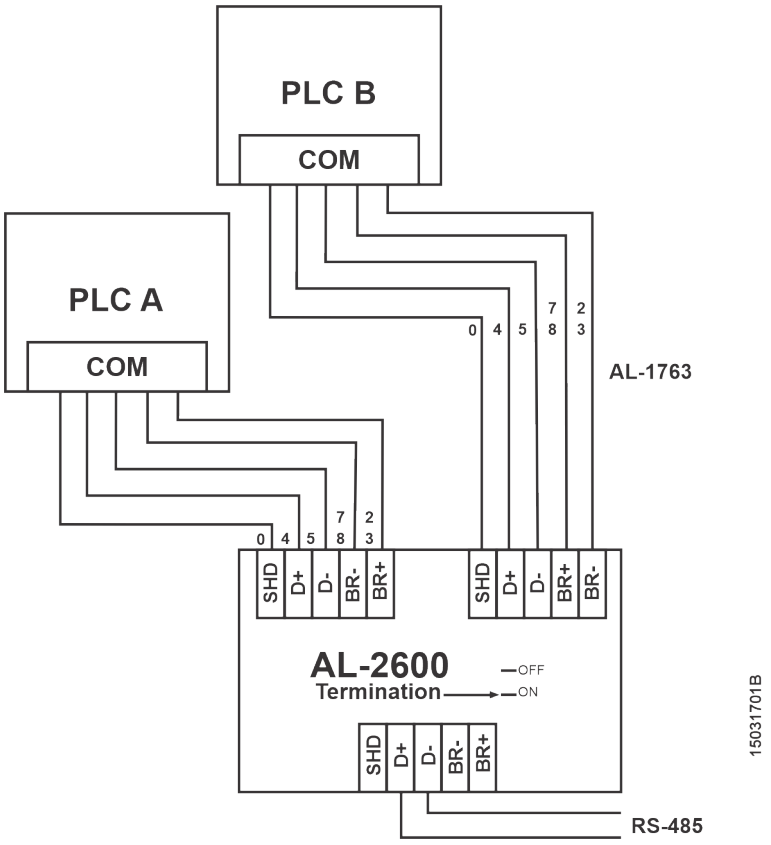


Figure 11: Connection Diagram of a RS-485 Network with External Termination and Master Redundancy

3.5. Memory Card Installation

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

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

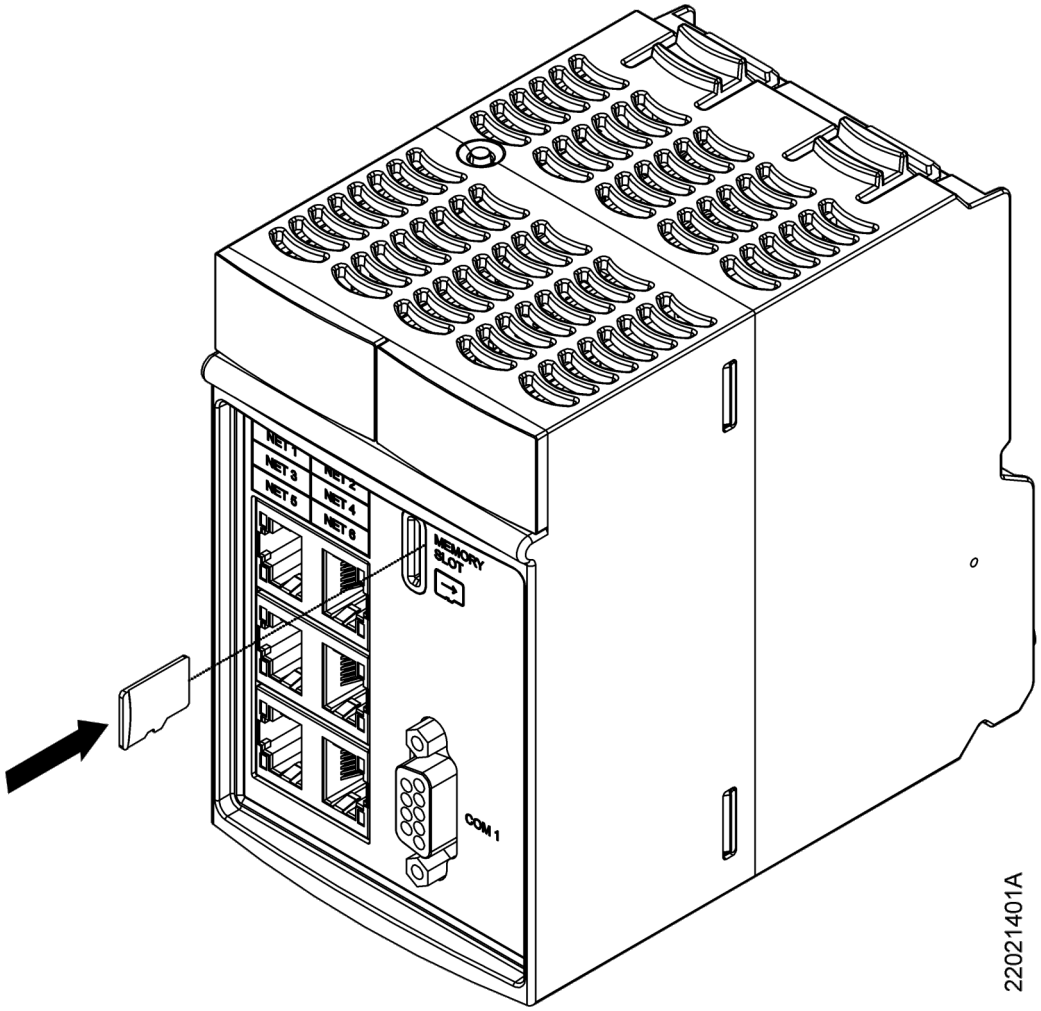


Figure 12: Inserting the Memory Card into the CPU

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

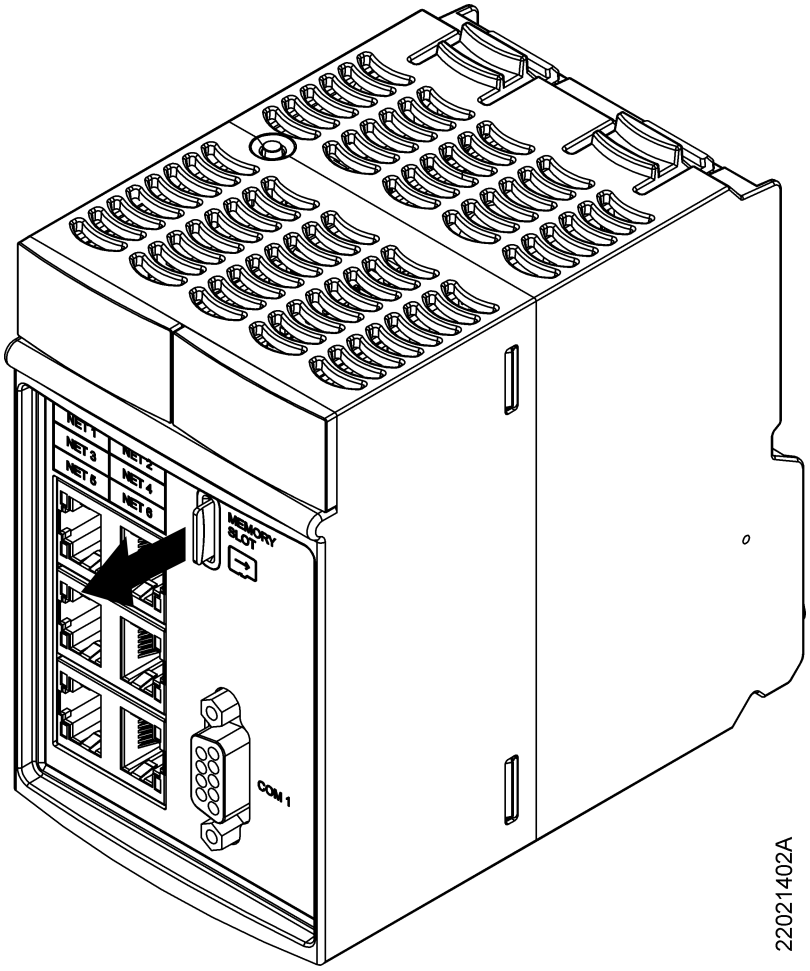


Figure 13: Removing the Memory Card

3.6. Architecture Installation

3.6.1. Module Installation on the Main Backplane Rack

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

3.7. Programmer Installation

To execute the Mastertool development software installation, it is necessary download the installation file from the site https://www.altus.com.br/en/. For further information about the step by step to installation, consult Mastertool User Manual.

4. Initial Programming

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

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

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

For further information regarding programming see the Mastertool user manual.

4.1. Memory Organization and Access

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

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

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

Table 31: Memory Organization and Access Example

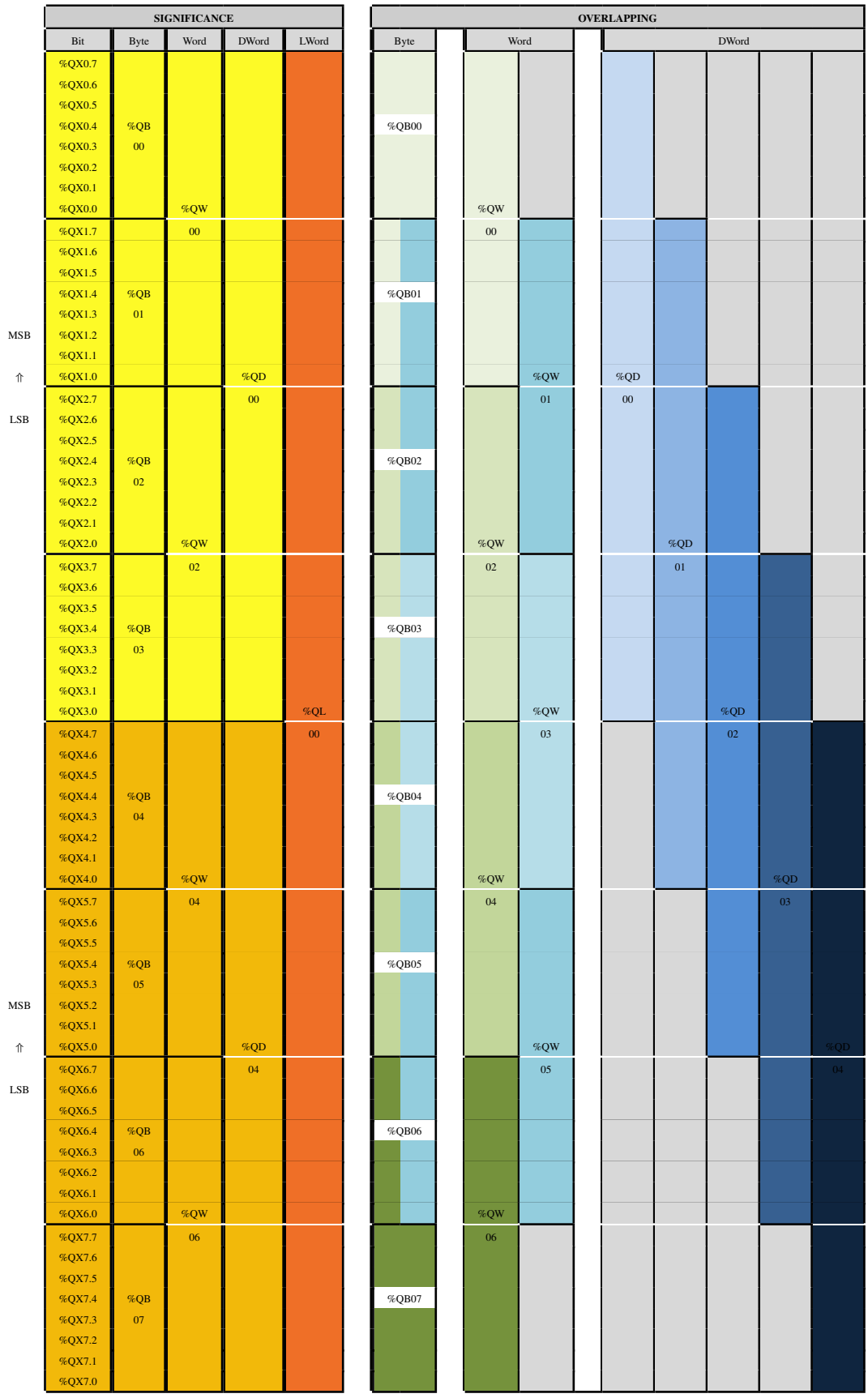


Table 32: Memory Organization and Access

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

4.2. Project Profiles

A project profile consists of an application template combined with a set of validation rules that guide application development, reducing programming complexity.

The project profile is selected in the project creation wizard. Each project profile defines a default naming template for tasks and programs, which are pre-created according to the selected profile. Additionally, during project compilation (code generation), Mastertool verifies all rules defined by the selected profile.

The following sections detail the characteristics of each profile. It is important to note that Mastertool allows changing the profile of an existing project (see the project update section in the Mastertool manual), but it is the developer's responsibility to make the necessary adjustments so that the project complies with the rules of the newly selected profile.

- There are no naming restrictions for tasks or POUs
- There are no mandatory fixed task structures
- The task limit is defined exclusively by the controller's capabilities

4.2.1. Fast Profile

In the Fast profile, by default, the application has a user task of the Cyclic type called MainTask. This task is responsible for implementing a single Program Organization Unit (POU) of the Program type, called MainPrg. This program can call other programming units of the Program, Function, or Function Block types, but all user code will be executed exclusively by the MainTask task.

This profile is characterized by allowing shorter intervals in MainTask, enabling faster execution of user code. Additionally, this profile may include an interrupt task with higher priority than MainTask, and therefore it can interrupt its execution at any time.

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

Table 33: Fast Profile Task

4.2.2. Process Profile

In the Process profile, by default, the application has a user task of the Cyclic type called MainTask. This task is responsible for implementing a single Program Organization Unit (POU) of the Program type, called MainPrg. This program can call other programming units of the Program, Function, or Function Block types, but all user code will be executed exclusively by the MainTask task.

Priorities, intervals, and associations between tasks and programs are fully configurable.

This profile is intended for applications that require greater flexibility and modularization of the control logic.

Task	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	100 ms	-

Table 34: Process Profile Task

4.2.3. Maximum Number of Tasks

The user can create multiple types of tasks, as long as the limits established for each type and the total value are not exceeded. The table in General Features shows the maximum number of tasks. Mastertool displays the detailed limits for task types in the "Properties" tab of the Task Configuration.

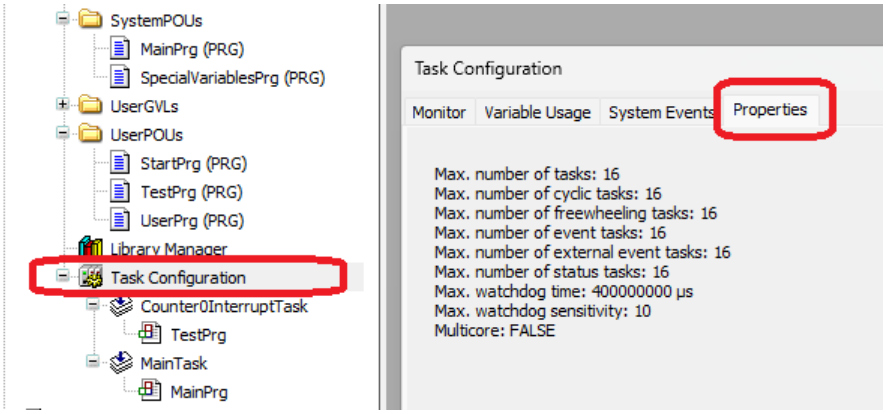


Figure 14: Maximum number of tasks

4.3. CPU Configuration

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

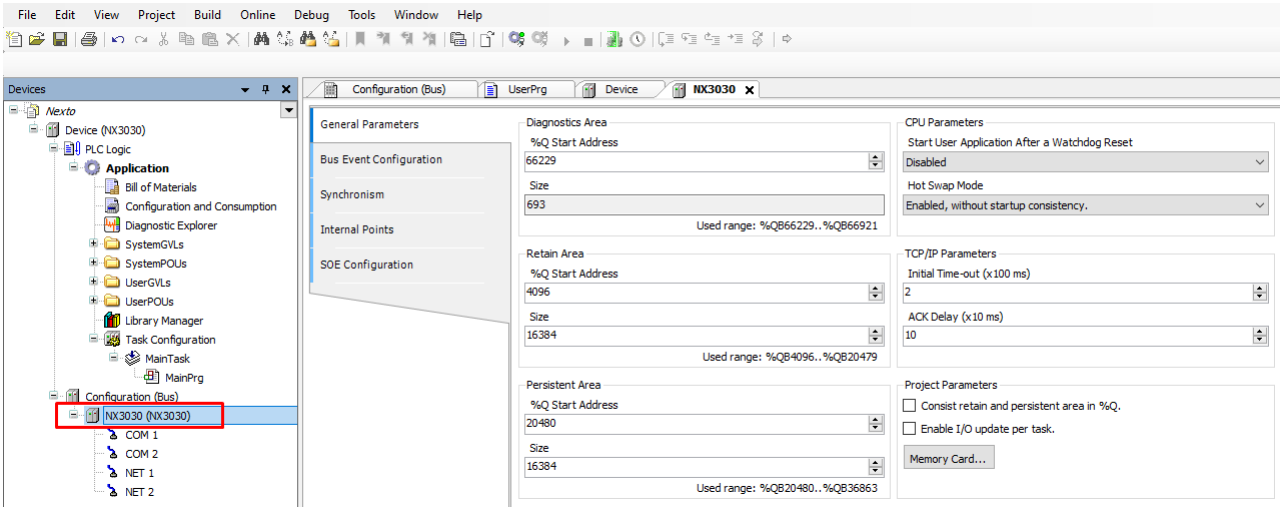


Figure 15: CPU Configuration

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

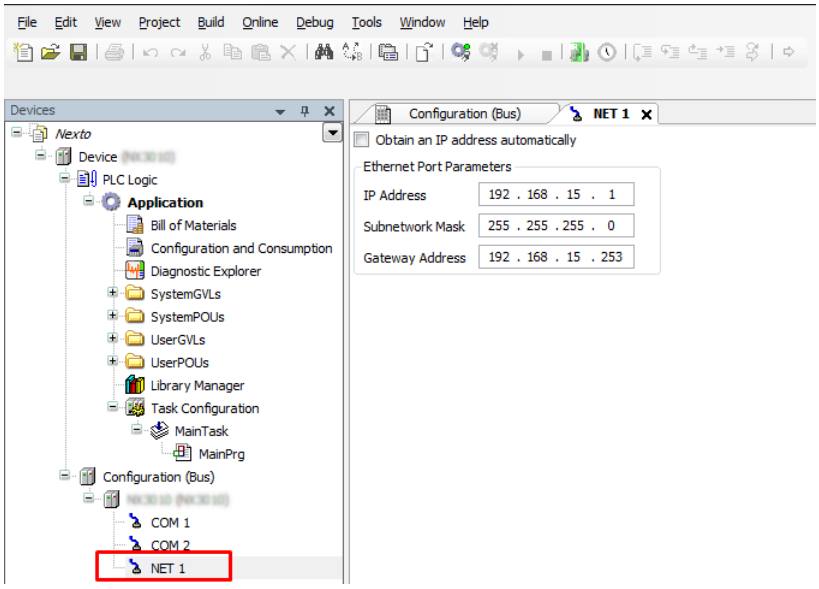


Figure 16: Configuring the CPU Communication Port

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

4.4. Libraries

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

4.5. Inserting a Protocol Instance

The Nexto Series CPUs, as described in the Protocols section, offer several communication protocols. Some protocols have a specific configuration procedure and are not added directly to the communication interfaces. For the remaining protocols, insertion can be done by simply right-clicking on the desired communication interface, selecting to add the device, and finally configuring the protocol instance as described in the Protocols Configuration section. An example is presented below.

4.5.1. MODBUS RTU

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

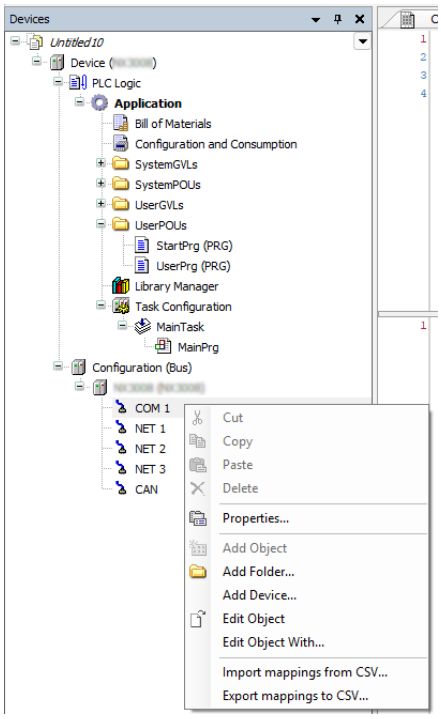


Figure 17: Adding an Instance

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

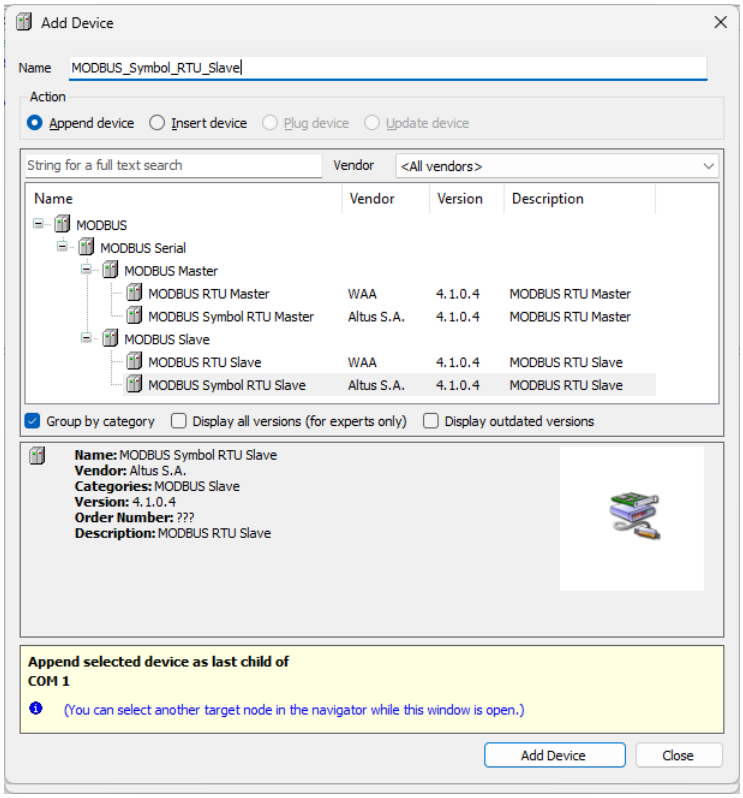


Figure 18: Selecting the Protocol

4.5.2. MODBUS Ethernet

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

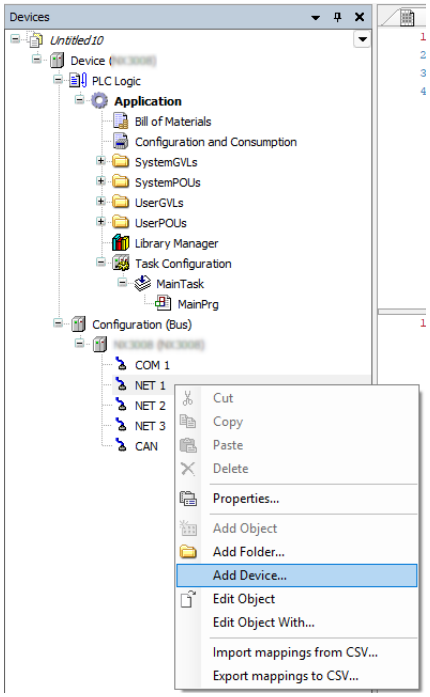


Figure 19: Adding the Instance

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

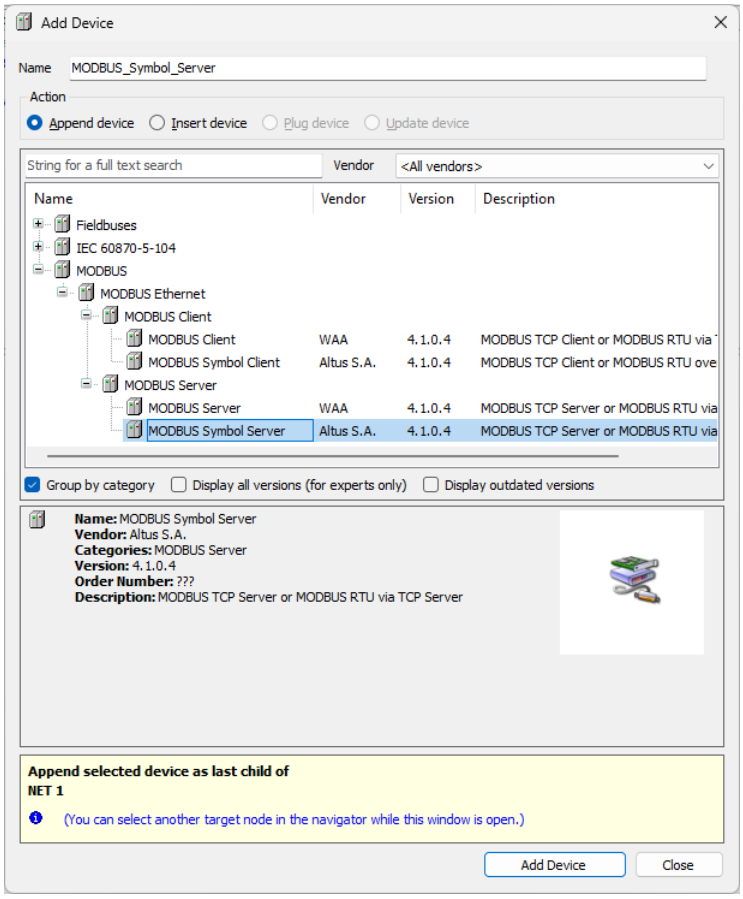


Figure 20: Selecting the Protocol

4.6. Finding the Device

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

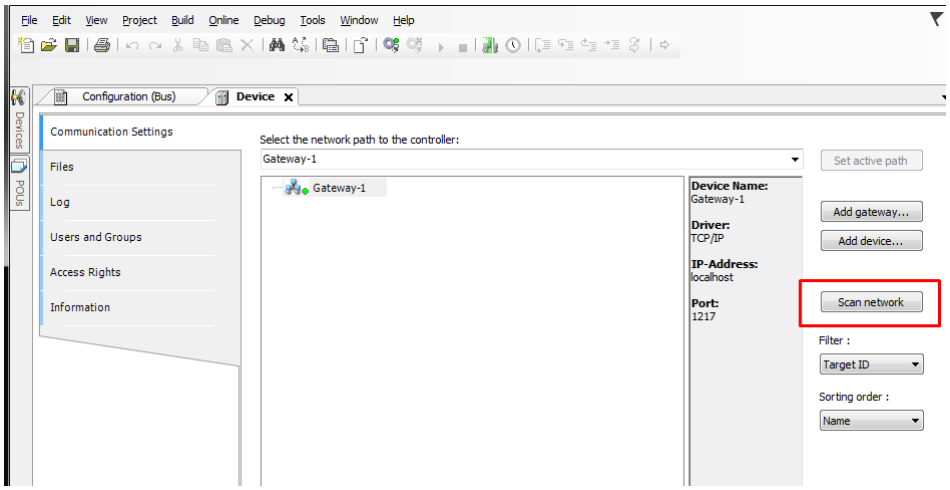


Figure 21: Finding the CPU

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

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

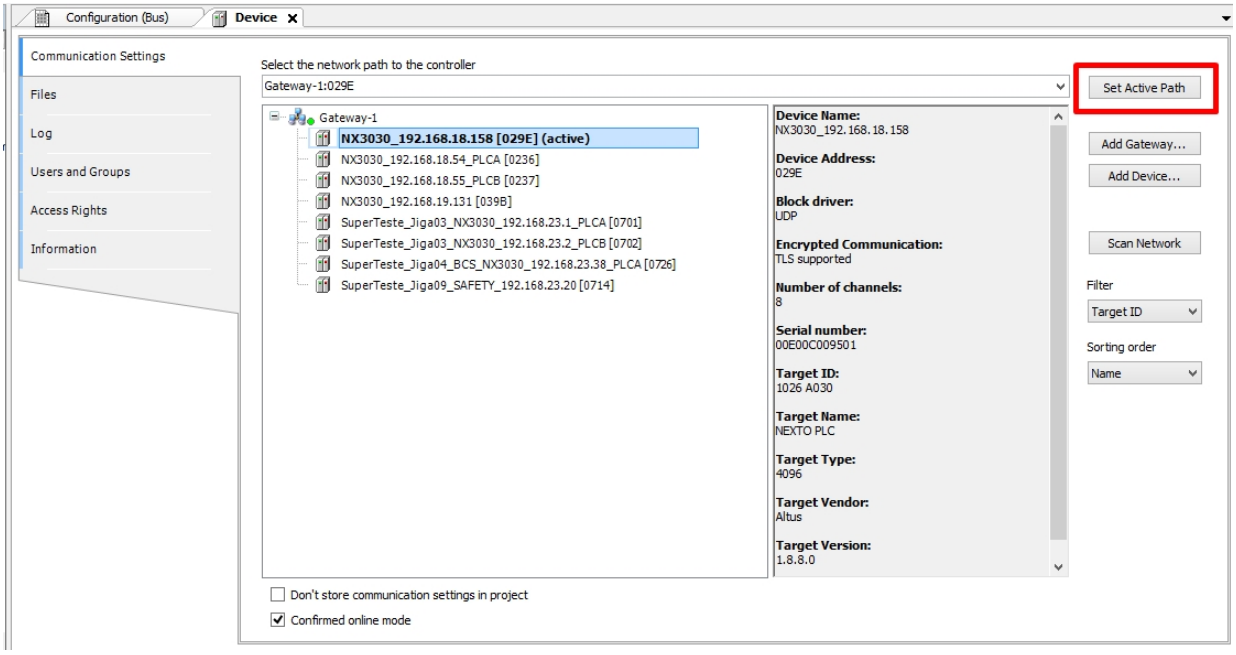


Figure 22: Selecting the CPU

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

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

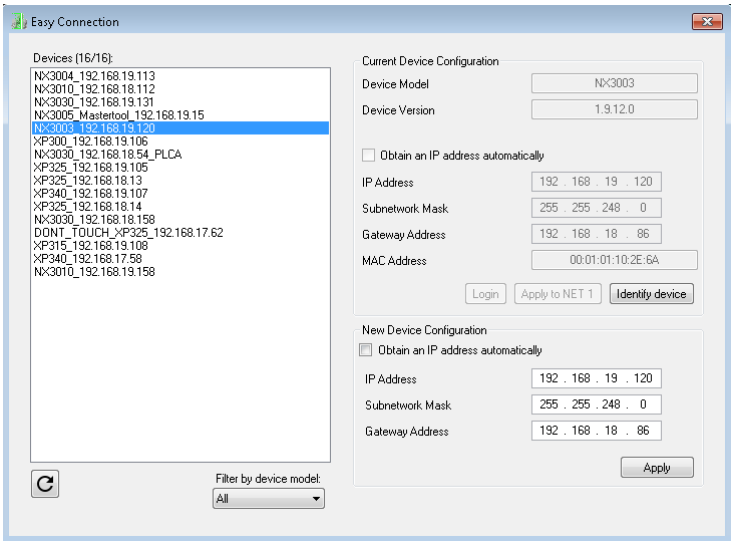


Figure 23: Easy Connection

This command performs a MAC level communication with the NET 1 interface of the device, allowing to permanently

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

4.7. Login

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

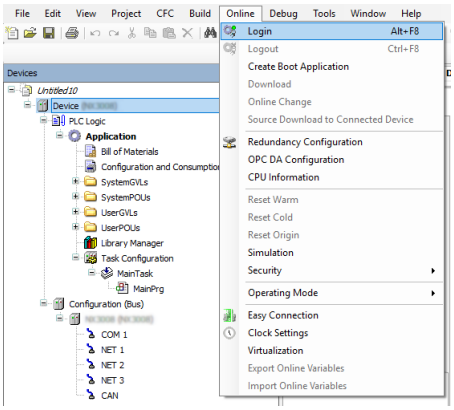


Figure 24: Sending the Project to the CPU

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

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

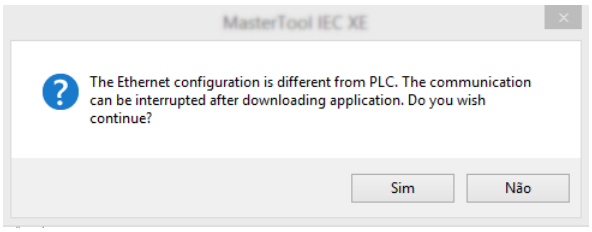


Figure 25: IP Configuration Warning

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



Figure 26: No application on the device

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

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

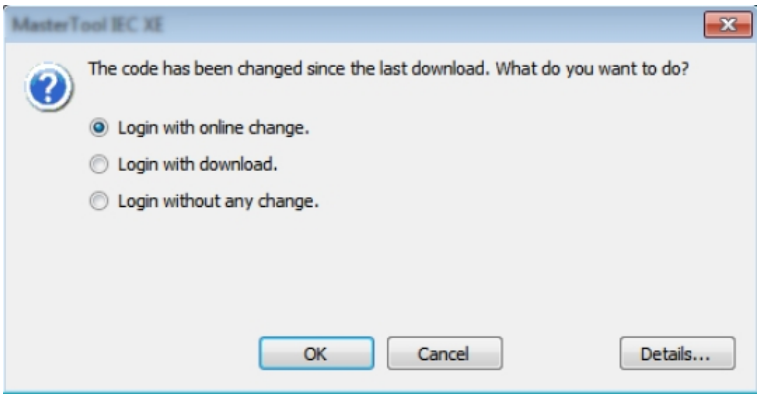


Figure 27: Project Update at the CPU

ATTENTION

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

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

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

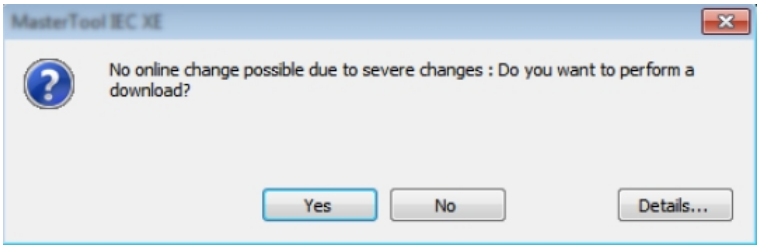


Figure 28: Configuration Changes

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

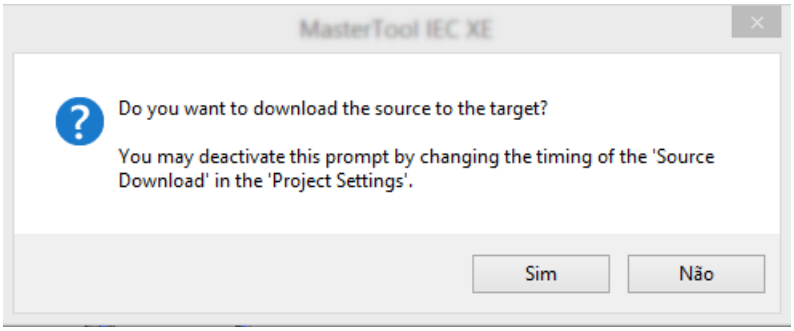


Figure 29: Source code download

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

4.8. Run Mode

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

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

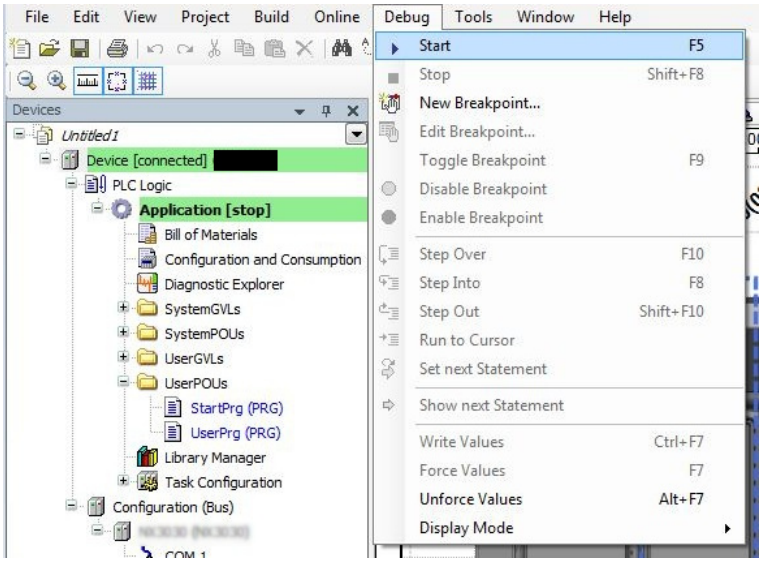


Figure 30: Starting the Application

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

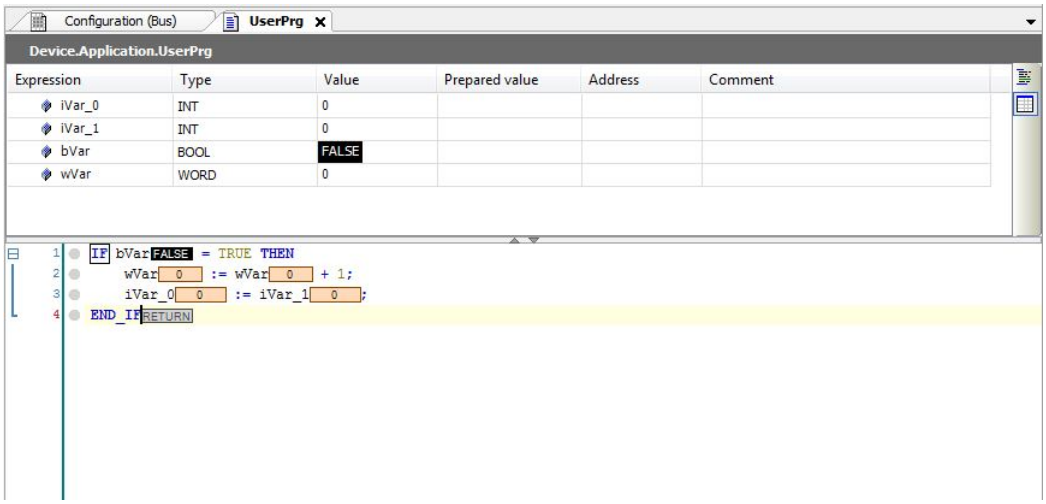


Figure 31: Program running

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

4.9. Stop Mode

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

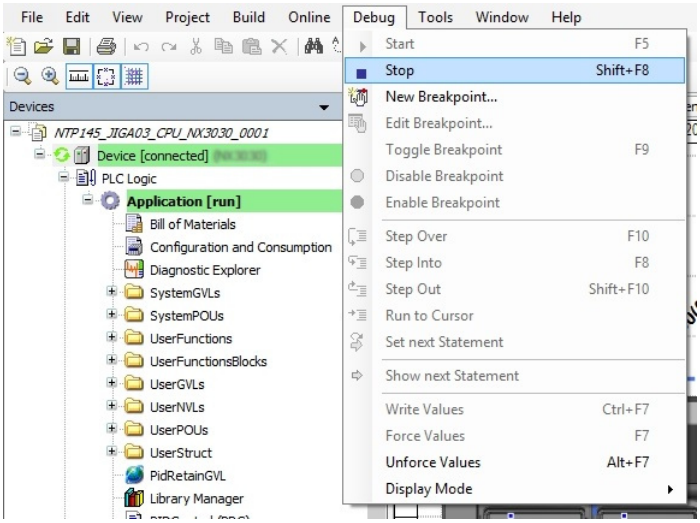


Figure 32: Stopping the Application

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

4.10. Writing and Forcing Variables

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

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

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

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

ATTENTION

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

ATTENTION

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

4.11. Logout

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

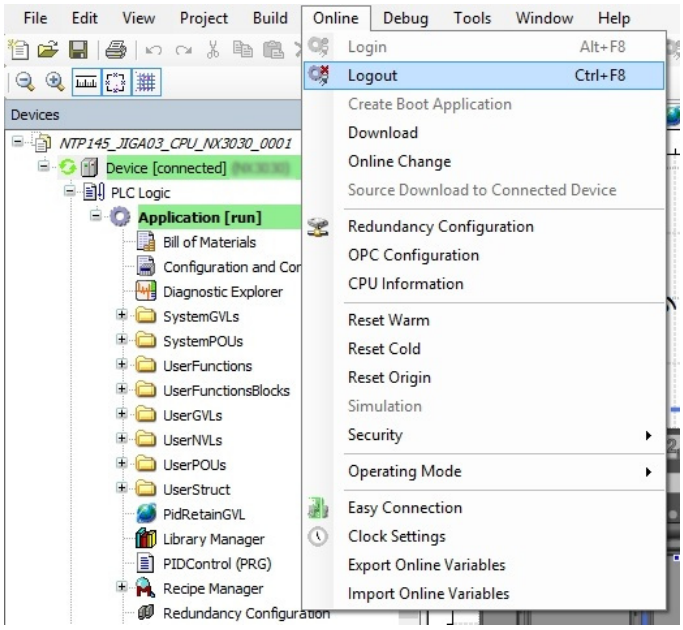


Figure 33: Finalizing the online communication with the CPU

4.12. Project Upload

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

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

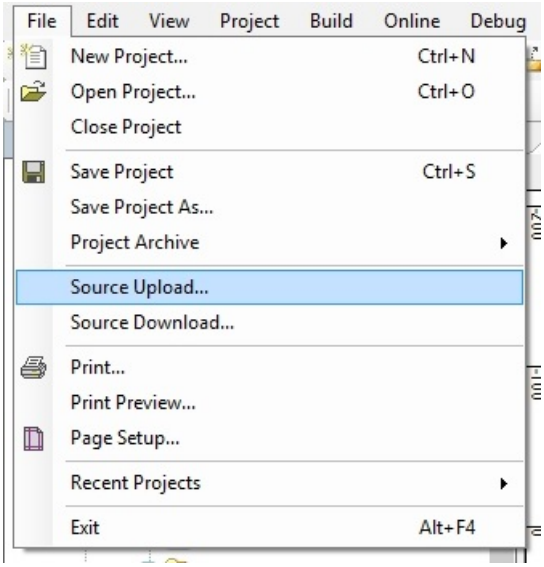


Figure 34: Project Upload Option

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

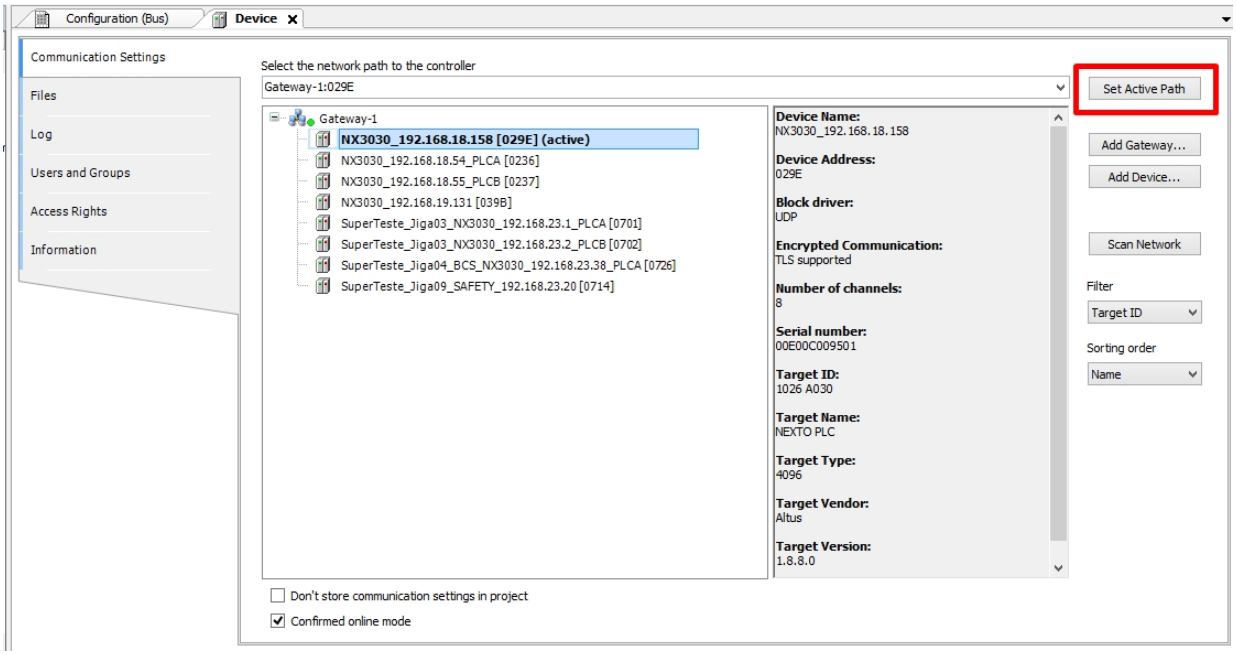


Figure 35: Selecting the CPU

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

ATTENTION

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

ATTENTION

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

4.13. CPU Operating States

4.13.1. Run

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

4.13.2. Stop

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

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

4.13.3. Breakpoint

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

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

4.13.4. Exception

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

4.13.5. Reset Warm

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

4.13.6. Reset Cold

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

4.13.7. Reset Origin

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

Notes:

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

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

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

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

4.14.1. MainPrg Program

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

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

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

PROGRAM MainPrg
VAR
   isFirstCycle : BOOL := TRUE;
END_VAR

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

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

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

4.14.2. StartPrg Program

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

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

4.14.3. UserPrg Program

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

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

4.14.4. GVL System_Diagnostics

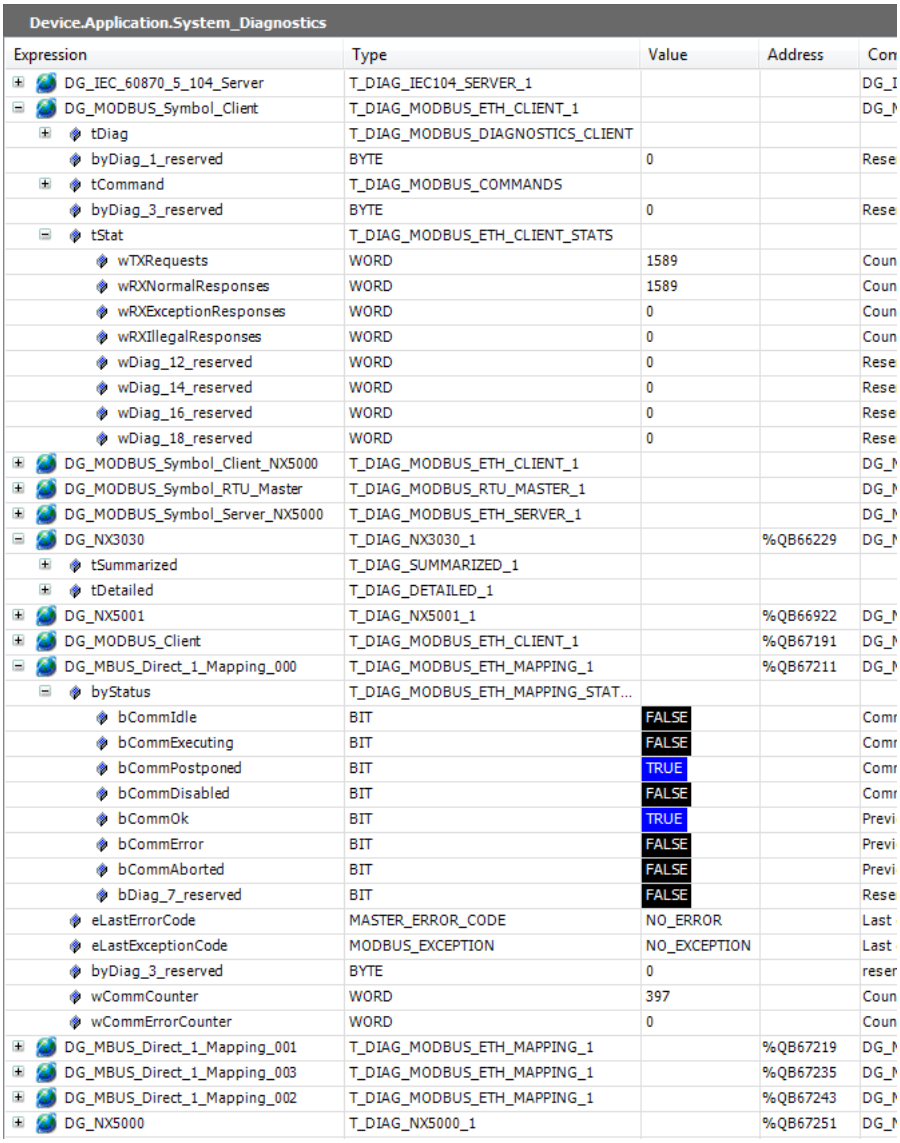
The *System_Diagnostics* GVL contains the diagnostic variables of the CPU, of the communication interface (Ethernet and PROFIBUS) and of all communication drivers. This GVL isn't editable and the variables are declared automatically with type specified by the device to which it belongs when it is added to the project.

ATTENTION

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

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

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



Device.Application.System_Diagnostics

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

Figure 36: System_Diagnostics GVL in Online Mode

4.14.5. GVL Disables

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

Requisition disabling variables declaration:

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

Where:

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

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

Example:

Device.Application.Disables

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

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

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

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

Device.Application.Disables

Expression	Type	Value	Preparec
MODBUS_Slave_1_DISABLE_0001	BOOL	FALSE	
MODBUS_Slave_1_DISABLE_0002	BOOL	TRUE	
MODBUS_Slave_1_DISABLE_0003	BOOL	FALSE	
MODBUS_Slave_1_DISABLE_0004	BOOL	FALSE	
MODBUS_Server_1_DISABLE_0001	BOOL	FALSE	
MODBUS_Server_1_DISABLE_0002	BOOL	FALSE	
MODBUS_Server_1_DISABLE_0003	BOOL	FALSE	
MODBUS_Server_1_DISABLE_0004	BOOL	TRUE	

Figure 37: Disable GVL in Online Mode

4.14.6. GVL IOQualities

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

Example: Device.Application.IOQualities

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

```
END_VAR
```

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

4.14.7. GVL Module_Diagnostics

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

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

Device.Application.Module_Diagnostics

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

Figure 38: Module_Diagnostics GVL in Online Mode

4.14.8. GVL ReqDiagnostics

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

Requisition diagnostic variable declaration:

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

Where:

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

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

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

ATTENTION

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

Example:

Device.Application.ReqDiagnostics

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

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

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

Device.Application.ReqDiagnostics

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

Figure 39: ReqDiagnostics GVL in Online Mode

5. Configuration

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

5.1. Device

5.1.1. User Management and Access Rights

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

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

5.1.2. PLC Settings

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

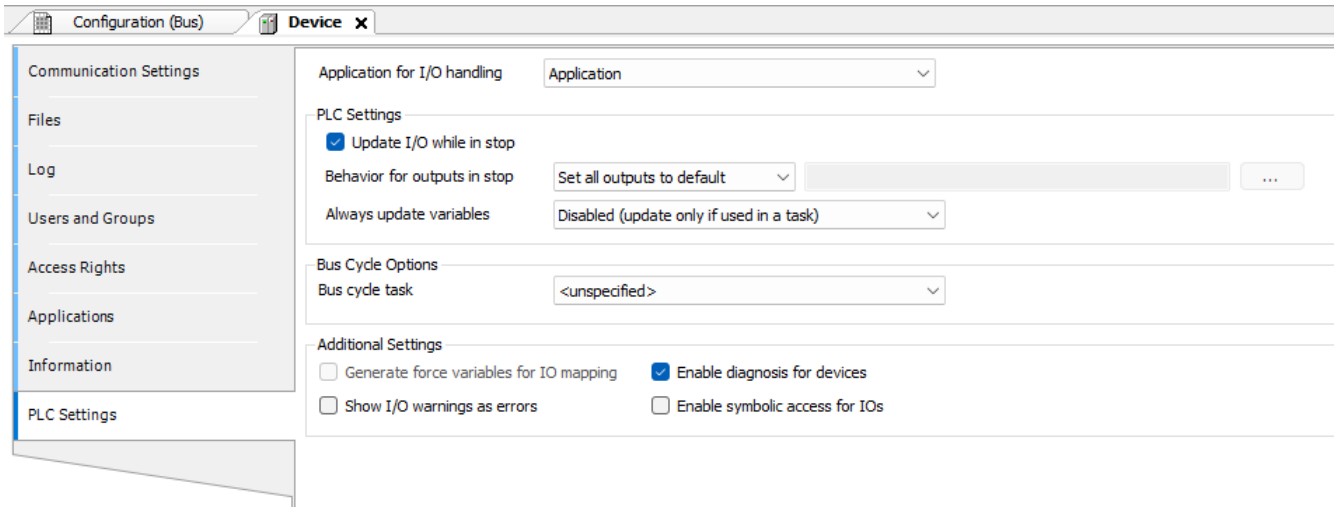


Figure 40: PLC Settings

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

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

Table 35: PLC Settings

ATTENTION

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

5.2. CPU Settings

5.2.1. General Parameters

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

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

Settings	Description	Default	Options
Start User Application after a Watchdog Reset or Runtime Reset	**CPU Parameters**		
	When enabled, it starts the user application after a hardware watchdog reset or after a runtime restart, while maintaining the diagnostic indication via the WD LED and via variables.	Disabled	Enabled Disabled
Hot Swap	Module Hot Swap Mode	Enabled, without consistency at startup. (may vary according to the CPU model)	- Disabled, only for declared modules - Disabled (with consistency at startup) - Disabled, without consistency at startup - Enabled, with consistency at startup only for declared modules - Enabled, with consistency at startup - Enabled, without consistency at startup
%Q Start Adress	**Diagnostics Area (%Q)**		
	Sart Address of CPU Diagnostics (%Q)	Automatically allocated when the project is created.	0 a 393217
Size	Size of the diagnostics area in bytes	Variable, depends on the project.	It is not possible to change the size of the CPU diagnostics area.
%Q Start Adress	**Retentive Area (%Q)**		
	Start address of the retentive data memory (%Q)	0	0 to 393215
Size	Size of the retentive data memory in bytes	0	0 a 393216
%Q Start Adress	**Persistent Area (%Q)**		
	Start address of the persistent data memory (%Q)	0	0 to 393215
Size	Size of the persistent data memory in bytes	0	0 to 393216

Table 36: CPU Settings

ATTENTION

When the start address or the size of the retentive or persistent data memory is changed in the user application, the memory is fully reallocated, causing the retentive and persistent variable areas to be cleared. Therefore, the user must take precautions to avoid losing the data stored in memory.

ATTENTION

In situations where the persistent symbolic memory area is modified, a message is displayed by the Mastertool programmer so that a behavior for this area can be selected after loading the modified program. This choice does not affect the direct representation persistent area, which is always cleared.

5.2.1.1. Hot Swap

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

CAUTION

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

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

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

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

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

Table 37: Bus Abnormal Situations

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

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

5.2.1.1.1. Hot Swap Disabled, for Declared Modules Only

In this configuration, the CPU is immediately in *Stop* Mode when an abnormal bus situation (as described on Table 37) happens. The LED DG starts to blink 4x (according to Table 38). In this case, in order to make the CPU to return to the normal state *Run*, in addition to undo what caused the abnormal situation, it is necessary to execute a *Reset Warm* or a *Reset Cold*. If a *Reset Origin* is carried out, it will be necessary to perform the download so that the CPU can return to the normal state (*Run*). The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by Mastertool in the *Online* menu.

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

5.2.1.1.2. Hot Swap Disabled

This setting does not allow any abnormal situation in the bus (as shown in Table 37) 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 38). For these cases, to turn the CPU back to normal *Run*, in addition to undo what caused the abnormal situation it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is done, you need to download the project so that the CPU can return to normal *Run*. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by Mastertool in the *Online* menu.

5.2.1.1.3. Hot Swap Disabled, without Startup Consistency

Allows the system to start up even when some module is in an abnormal bus situation (as shown in Table 37). Abnormal situations are reported via diagnosis.

Any modification to the bus will cause the CPU to enter *Stop* Mode, and the DG LED will start blinking 2x (as in Table 38). In order for the CPU to return to the normal *Run* state in these cases, it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is performed, it will be necessary to download the CPU so that the CPU can return to the normal *Run* state. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by Mastertool in the *Online* menu.

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

"Startup" is the interval between the CPU energization (or reset command or application download) until the first time the CPU gets in *Run* Mode after been switched on. This configuration verifies if any abnormal bus situation has occurred (as described on Table 37) during the start. In affirmative case, the CPU gets in *Stop* Mode and the LED DG starts to blink 4x (according to Table 38). Afterwards, in order to set the CPU in *Run* mode, further to fix what caused the abnormal situation, it is necessary to execute a *Reset Warm* or *Reset Cold* command, which can be done by the Mastertool (Online menu). If a *Reset Origin* is carried out, it will be necessary to perform the download so that the CPU can return to the normal state (*Run*).

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

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

ATTENTION

In this configuration when a power fault occurs (even temporally), *Reset Warm* Command,*Reset Cold* Command or a new application *Download* has been executed, and if any module is in an abnormal bus situation, the CPU will get into *Stop* Mode and the LED DG will start to blink 4x (according to Table 38). This is considered a startup situation.
This is the most advised option because guarantee the system integrity on its initialization and allows the modules change with a working system.

5.2.1.1.5. Hot Swap Enabled with Startup Consistency

This setting checks whether there has been any abnormal situation in the bus (as shown in Table 37) during the startup, even if there is no declared modules and present on the bus; if so, the CPU goes into *Stop* mode and the LED DG starts to blink 4x (as shown in Table 38). For these cases, to turn the CPU back to normal *Run*, in addition to undo what caused the abnormal situation it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is done, you need to download the project so that the CPU can return to normal *Run*. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by Mastertool in the *Online* menu.

5.2.1.1.6. Hot Swap Enabled without Startup Consistency

Allows the system to start working even if a module is in an abnormal bus situation (as described on Table 37). The abnormal situations are reported via diagnostics during and after the startup.

ATTENTION

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

5.2.1.1.7. How to do the Hot Swap

CAUTION

Before performing the Hot Swap it is important to discharge any possible static energy accumulated in the body. To do that, touch (with bare hands) on any metallic grounded surface before handling the modules. Such procedure guaranties that the module static energy limits are not exceeded.

ATTENTION

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

The hot swap proceeding is described below:

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

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

It is important to note that in the cases the CPU gets in *Stop* Mode and the DG LED starts to blink 4x (according to Table 38, due to any abnormal bus situation (as described on Table 37, 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 38 presents the bus conditions and the Nexto CPU DG LED operation state. For further information regarding the diagnostics LEDs states, see Diagnostics via LED section.

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

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

Table 38: Hot Swap and Conditions Relations

Note:

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

5.2.1.2. Retain and Persistent Memory Areas

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

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

```
PROGRAM UserPrg
VAR RETAIN
```

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

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

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

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

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

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

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

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

```
VAR_GLOBAL PERSISTENT RETAIN
  wGlobalSymbolicPersistentVariable_01 : WORD;
END_VAR

VAR_GLOBAL RETAIN
  wGlobalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

5.2.2. Time Synchronization

For the time synchronization, Nexto Series CPUs use the SNTP (*Simple Network Time Protocol*) or the synchronism through IEC 60870-5-104.

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

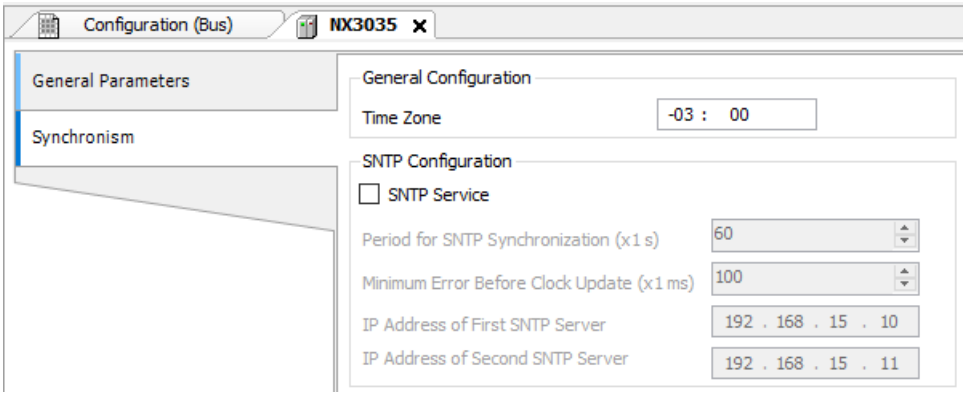


Figure 41: SNTP Configuration

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

Table 39: SNTP Configurations

Notes:

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

Time zone: The time zone configuration is used to convert local time to UTC time and vice versa. While some synchronization sources use local time (examples: IEC 60870-5-104 protocol, SetDateAndTime function), others use UTC time (SNTP). UTC time is normally used to timestamp events (examples: DNP3, IEC 60870-5-104, Mastertool Device LOG), whereas local time is used by other CPU functionalities (examples: GetDateAndTime function, OTD date and time information).

5.2.2.1. SNTP

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

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

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

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

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

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

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

ATTENTION

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

CAUTION

It is vital to setup at least one SNTP server. It is recommended to configure two SNTP servers (primary and secondary). SNTP synchronism is necessary to generate events with a coherent time stamp between the PLCA and PLCB and with the world time. Another usefulness is to avoid discontinuities during a switchover in applications that reference date and time, considering that there is no synchronization of date and time between the PLCs through the NETA and NETB synchronism channels.

5.2.2.2. Daylight Saving Time (DST)

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

For further information, see the section RTC Clock.

5.3. Serial Interface Configuration

5.3.1. COM 1

The COM 1 communication interface consists of a female DB9 connector for the RS-485 standard. It enables point-to-point or network communication using the open MODBUS RTU Slave or MODBUS RTU Master protocols.

When the MODBUS Master/Slave protocol is used, some of these parameters (such as *Serial Mode*, *Data Bits*, *RX Threshold*, and *Serial Events*) are automatically adjusted by the Mastertool to ensure correct protocol operation.

Below are the parameters that must be configured for proper application performance.

Configuration	Description	Default	Options
Serial Type	Serial channel configuration.	RS-485	RS-485

Configuration	Description	Default	Options
Baud Rate	Serial communication port speed configuration.	115200	9600, 19200, 38400, 57600, 115200 bps
Parity	Serial port parity configuration.	None	Odd Even None
Data Bits	Sets the data bits quantity in each serial communication character.	8	8
Stop Bits	Sets the serial port stop bits.	1	1 and 2
Serial Mode	Sets the serial port operation mode.	Normal Mode	- Extended Mode: Extended operation mode which delivers information regarding the received data frame - Normal Mode: Serial communication normal operation mode

Table 40: RS-485 Standard Serial Configurations

5.3.1.1. Advanced Configurations

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

Configuration	Description	Default	Options
Threshold de RX da UART	Number of bytes that must be received to generate a new UART interrupt. Low values make the TIMESTAMP more precise when EXTENDED MODE is used and minimize overrun errors. However, low values may generate too many interruptions, which can slow down the CPU.	8	1, 4, 8 e 14

Table 41: Advanced Configuration of the RS-485 Serial Interface

5.4. Integrated Ethernet Interfaces Configuration

The CPU NX3035 has six integrated Ethernet interfaces (NET 1 to NET 6). Each of these interfaces has an RJ45 communication connector in the 10BASE-Te/100BASE-TX/1000BASET standard. These interfaces enable communication using open protocols such as MODBUS TCP Client, MODBUS RTU via TCP Client, MODBUS TCP Server, and MODBUS RTU via TCP Server. These interfaces can also be used to program the PLC using the Mastertool programmer.

5.4.1. Basic Ethernet Port Settings

The following screen shows the basic settings for Ethernet ports NET1 to NET6.

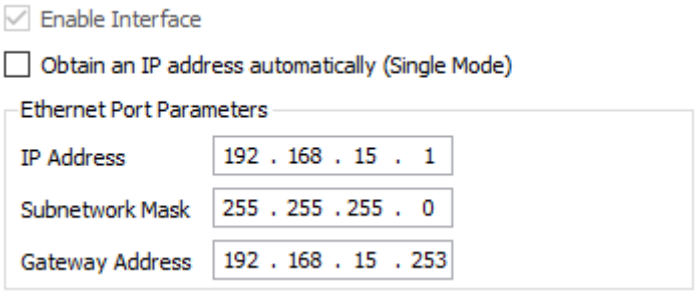


Figure 42: Basic Ethernet Port Settings

The *Enable Interface* checkbox must be checked for the port to work. In the specific case of the NET1 port, this enablement is fixed, as it is the default port for connecting the Mastertool programmer. The other ports (NET2 ... NET6) are disabled by default.

The following table describes how to fill in the other fields on the previous screen.

Configuration	Description	Default	Options
Obtain an IP address automatically	Enables the DHCP Client functionality on the device for automatic IP assignment	Unchecked	Checked or Unchecked
IP Address	IP address of the port on the Ethernet network	NET1: 192.168.15.1 NET2: 192.168.16.1 NET3: 192.168.17.1 NET4: 192.168.18.1 NET5: 192.168.19.1 NET6: 192.168.20.1	1.0.0.1 to 223.255.255.254
Subnet Mask	Subnet mask of the port on the Ethernet network	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Gateway address of the port on the Ethernet network	NET1: 192.168.15.253 NET2: 0.0.0.0 NET3: 0.0.0.0 NET4: 0.0.0.0 NET5: 0.0.0.0 NET6: 0.0.0.0	0.0.0.0 to 223.255.255.254

Table 42: IP Addressing of Integrated Ports of the NX3035 CPU

ATTENTION

It is not possible to configure two different Ethernet ports of the CPU NX3035 on the same subnet, as this type of configuration is blocked by the Mastertool tool. Therefore, each Ethernet port of the CPU NX3035 must be configured on a separate subnet.
In a redundant PLC (see chapter Redundancy with NX3035 CPU), the *IP Address* parameter is replaced by three different IP addresses, which must be on the same Ethernet subnet. These three IP addresses are described in the section IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup.

5.4.2. Ethernet Port Mode Configuration

The following screen shows the mode settings for Ethernet ports NET1 to NET6.

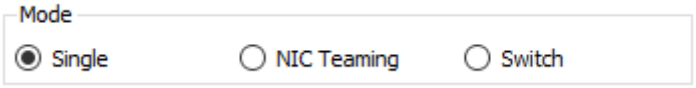


Figure 43: Ethernet Port Mode Configuration

You can select one of the following three modes for an Ethernet port:

- The *Simple* mode indicates that the Ethernet port operates individually (without redundancy).
- The *NIC Teaming* mode indicates that this Ethernet port and the next one form a pair with NIC Teaming redundancy. For more details, see section NIC Teaming Mode on the NX3035 CPU.
- The *Switch* mode is a future feature and should not be used yet.

5.4.2.1. Simple Mode

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

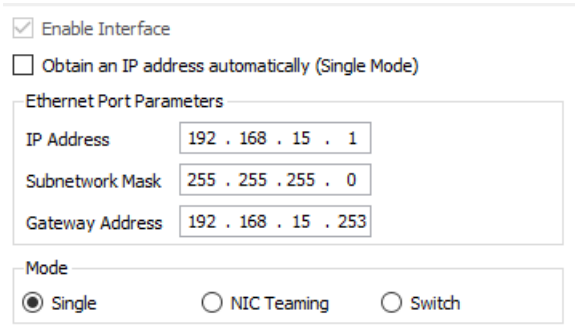


Figure 44: Advanced Configuration of Local Ethernet Interfaces - Single Mode

5.4.2.2. NIC Teaming Mode on the NX3035 CPU

In this mode, the interface forms a redundant pair with the subsequent interface, operating in an active/stanby scheme. A *NIC Teaming* pair has a single IP address, associated with the interface that is currently active. Thus, a client in a SCADA or Mastertool, connected to a server on the CPU, doesn't have to worry about changing the IP address if any of the ports in the peer fail. In addition, each of the interfaces that form a redundant communication pair has separate diagnostics, to facilitate the debugging of faults that may eventually arise.

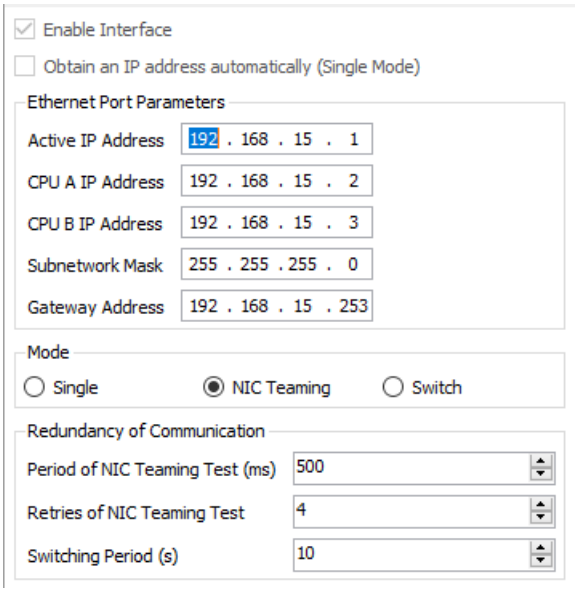


Figure 45: Advanced Configuration of Local Ethernet Interfaces - NIC Teaming Mode

When the *NIC Teaming* Mode is selected, on the same screen other parameters are automatically enabled and must be configured:

- **Period of NIC Teaming Test (ms):** Period for sending the communication test frame between the two NETs. Can be configured with values between 100 and 9900, default 500
- **Retries of NIC Teaming Test:** Maximum number of times the NET that sent the frame will wait for a response. Can be configured to values between 1 and 100, default 4
- **Switching Period (s):** Maximum time that NET Active will wait for any given packet. Can be configured with values between 1 and 25, default 10

If the response time of the *NIC Teaming* reaches *Test Period* times the *Number of Retries* and the active interface remains longer than the *Switching Period* without receiving any packets, a switchover occurs, making the previously inactive interface, active. It is important to note that have a delay between fault detection and activation of the inactive interface due to the time required for its configuration. This delay can be up to a few tens of milliseconds.

When one of the NETs is active, it will assume the configured IP address, and the inactive NET will remain with its IP Address, Subnet Mask and Gateway Address parameters blank in the CPU diagnostics.

NIC Teaming redundancy between two Ethernet ports on the NX3035 CPU has the following characteristics:

- Redundancy is formed by a pair of CPU ports, which can be:
  - NET1 + NET2
  - NET3 + NET4
  - NET5 + NET6
- At any given moment, one of the ports in the pair will be active, and the other will be in standby.
- Communications with other devices via Ethernet using TCP/IP or UDP/IP protocols will occur through the active port.
- Regarding the IP addressing of a pair of NIC Teaming ports:
  - The standby port does not have an IP address.
  - The active port may have one or two IP addresses. It will only have two IP addresses if the NX3035 CPU is installed in a redundant PLC, and if the half-cluster with this NX3035 is momentarily in Active state (see chapter Redundancy with NX3035 CPU and section IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup).
  - This way, a client such as SCADA or Mastertool, connected to a server on the PLC, does not need to worry about changing the IP address if any of the ports on the NIC Teaming pair fail.
- The active port may switch to standby, and the standby port to active, in the following situations:
  - Loss of connection from the active port (link-down).
  - Verification of communication inactivity on the active port's TCP/IP connections, together with the inability to exchange test messages between the active and backup ports.

- The two ports that form a NIC Teaming pair have separate diagnostics to indicate individual failures.
- Configuring port parameters and adding devices such as MODBUS will only be possible on the first port of the pair (NET1, NET3, or NET5). The second port (NET2, NET4, or NET6) will have its configuration parameters and device insertion blocked, as it shares these parameters and devices with the first port of the NIC Teaming pair.

Architectures with NIC Teaming aim for high availability. To this end, it is important that the two ports in the pair are connected to two different switches, so that the failure of a single switch does not compromise communication. These two switches must be interconnected in some way, so that the two ports in the NIC Teaming pair can exchange test messages with each other. These periodic test messages are used to control the switch between port states (active and standby) in certain situations.

5.4.2.3. Switch Mode

In this mode, the interface is paired with the subsequent interface, operating as a two-port Ethernet switch and allowing communication through both ports. The loop protection configuration options for switch mode are described in the following subsections.

5.4.2.3.1. Switch Mode - Disabled

In this mode, the two subsequent ports of the CPU operate as a simple unmanaged switch, without any loop protection.

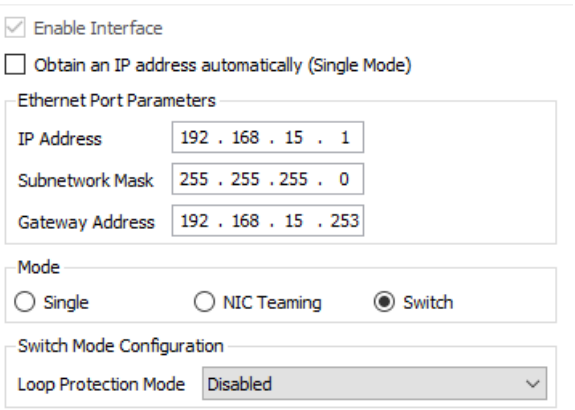


Figure 46: Advanced Configuration of Local Ethernet Interfaces - Switch Mode - Disabled

The following figure shows an example of a topology using two ports with this configuration. The three CPUs are connected directly, eliminating the need for an external switch.

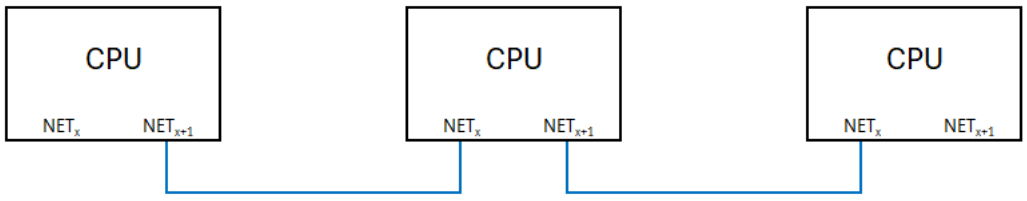


Figure 47: Example Topology with Switch Mode – Loop Protection Disabled

5.4.2.3.2. Switch Mode - MRP

MRP Protection mode allows ring topologies to be configured using the Media Redundancy Protocol (MRP).

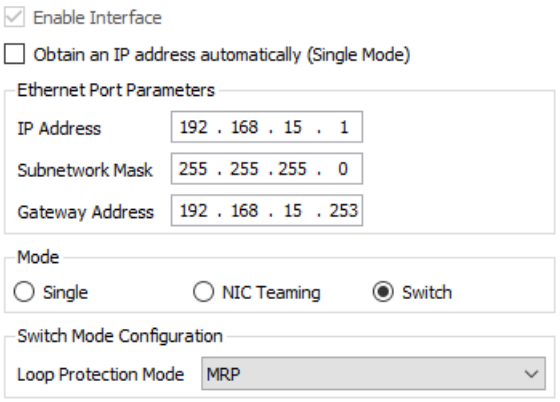


Figure 48: Advanced Configuration of Local Ethernet Interfaces - Switch Mode - MRP

This configuration is only possible if a dual-port PROFINET protocol instance (Ethernet / PN_Controller_Dual_port) is configured below the two subsequent ports. In this MRP ring:

- The CPU must assume the role of the *PROFINET Controller*, and all other devices must assume the role of *PROFINET Devices*.
- The CPU can be configured as either an *MRP Manager* or an *MRP Client*.

For detailed information about the configuration and operation of the MRP protocol, refer to Nexto Series PROFINET Manual - MU214621.

The following figure shows an example of an MRP ring using two subsequent ports of a CPU. Ring topologies provide high availability by maintaining communication in the event of a link failure between devices. In addition, in the event of a complete failure of a PROFINET Device, the CPU can continue communicating with the remaining PROFINET Devices.

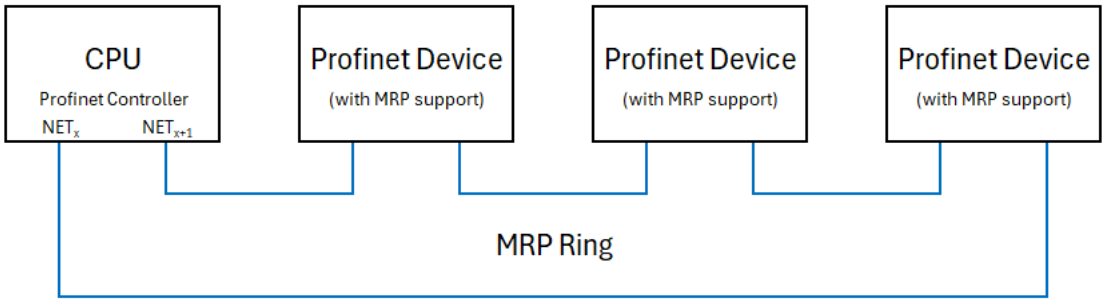


Figure 49: Example Ring Topology with Switch Mode – MRP

5.4.2.3.3. Switch Mode - RSTP

In this mode, the pair of network interfaces uses the *Rapid Spanning Tree Protocol* (RSTP) to prevent network loops and provide communication redundancy in topologies such as mesh or ring networks. However, if an STP frame is received on either port, the protocol is automatically downgraded to the *Spanning Tree Protocol* (STP), ensuring continued communication with the network. Note that the STP protocol has slower convergence times than the RSTP protocol.

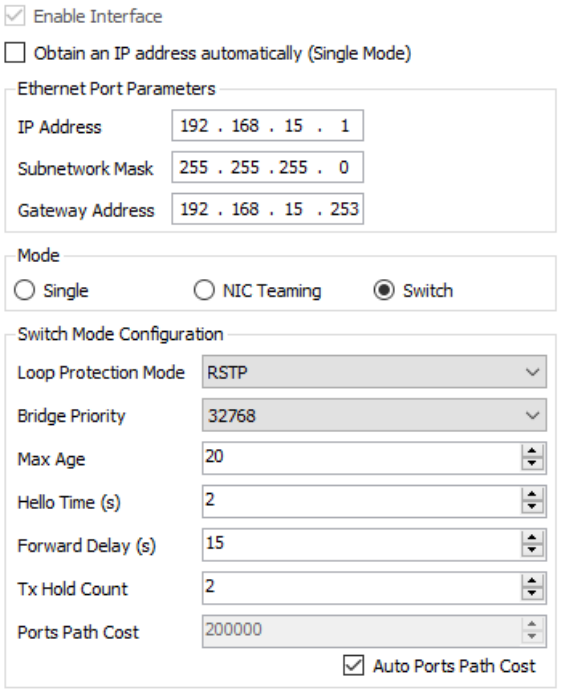


Figure 50: Advanced Configuration of Local Ethernet Interfaces - Switch Mode - RSTP

For more information about the use and configuration of the RSTP protocol, refer to Application Note NAP169 - RSTP in Nexto CPUs.

The following figure shows an example of the RSTP protocol being used on two subsequent ports of several CPUs in a mesh topology. The other switches in the network must also support RSTP or STP. If any switch in the network supports only STP, the CPU automatically downgrades from RSTP to STP, resulting in longer network recovery times after a failure.

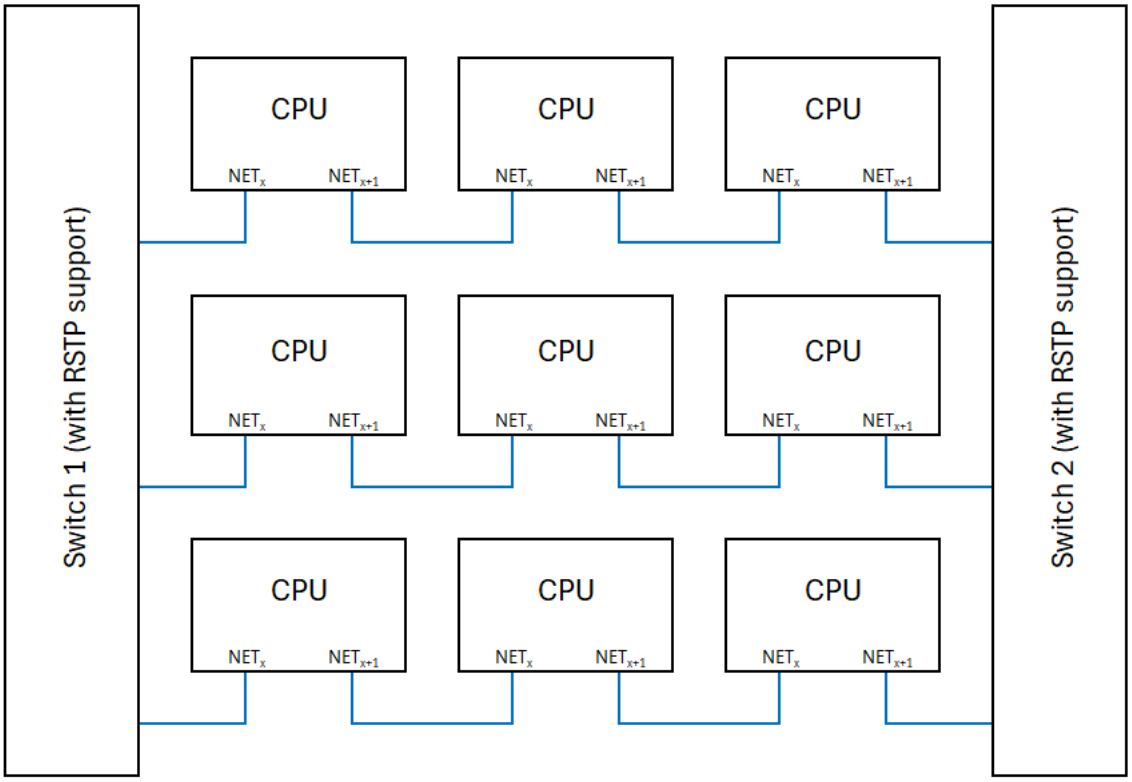


Figure 51: Example Mesh Topology with Switch Mode – RSTP

The following figure shows an example of the RSTP protocol being used on two subsequent ports of several CPUs in a ring topology. Unlike the ring topology using MRP Protection mode described earlier, this configuration does not require a PROFINET Controller on the subsequent ports.

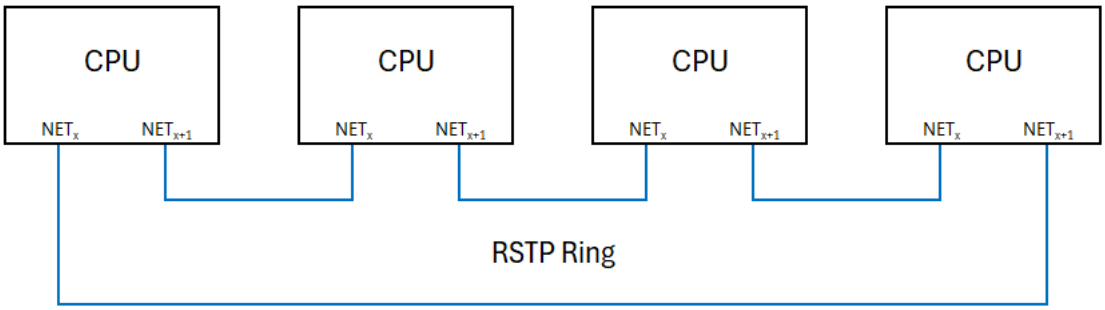


Figure 52: Example Ring Topology with Switch Mode – RSTP

5.4.3. Ethernet Port Failure Mode Configuration

The following screen only appears in a redundant PLC design (see chapter Redundancy with NX3035 CPU).

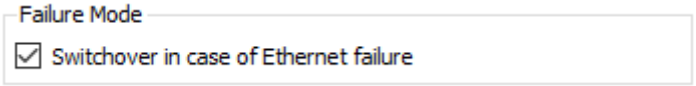


Figure 53: Ethernet Port Failure Mode Configuration

5.4.4. Reserved TCP/UDP Ports

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

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

Table 43: Reserved TCP/UDP ports

* Default port, but user changeable.

5.4.5. Network Routing

For integrated Ethernet ports, the CPU operating system has a network routing mechanism that allows the communication between different networks:

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

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

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

For remote Ethernet ports (NX5000 module), the routing is more restrict and does not interacts with the integrated ports:

1. Client protocols, when configured to communicate with a Server device located in a different network, will forward the communication to the Default Gateway configured on its NET interface.
2. Server protocols will accept connections only from its NET interface.

5.5. Configuration of Ethernet Interfaces with NX5000 Modules

In addition to the six integrated Ethernet interfaces of the NX3035 CPU (NET1 ... NET6), up to six NX5000 modules can be inserted into the same backplane rack as the NX3035 CPU, in backplane rack positions to the right of the NX3035 CPU. This may be necessary if the six integrated ports of the NX3035 CPU are not sufficient.

Ethernet ports on NX5000 modules can be used individually or organized into NIC Teaming pairs. NIC Teaming pairs are used to provide redundant Ethernet interfaces and require the installation of two NX5000 modules in adjacent backplane rack positions. This allows you to add up to six single Ethernet ports, or three redundant Ethernet ports with NIC Teaming, or other combinations that do not exceed six NX5000 modules (e.g., two redundant ports with NIC Teaming and two single ports).

5.5.1. Basic Ethernet Port Settings for the NX5000

The following screen shows the basic settings for the NX5000's NET1 Ethernet port.

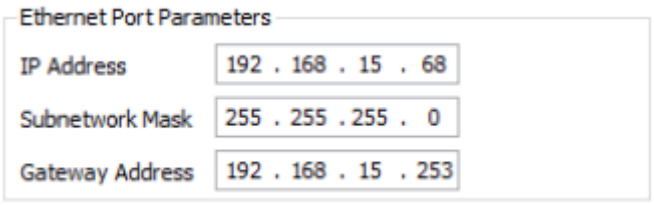


Figure 54: Basic Ethernet Port Settings of the NX5000

5.5.1.1. IP Addressing of Remote Ethernet Ports on the NX5000 (NET1)

5.5.1.1.1. NET 1

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

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

Configuration	Description	Default	Options
IP Address	IP address of the port on the Ethernet network	192.168.xx.68	1.0.0.1 to 223.255.255.254
Subnetwork Mask	Subnet mask of the port on the Ethernet network	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Gateway address of the port on the Ethernet network	192.168.xx.253	0.0.0.0 to 223.255.255.254

Table 44: NX5000 Remote NET 1 Configuration

In a redundant PLC (see chapter Redundancy with NX3035 CPU), the *IP Address* parameter can be replaced by several different IP addresses, which must belong to the same Ethernet subnet. The number of configured IP addresses depends on the cluster IP addressing method, whose selection is visible only in redundant PLC projects and is shown in the following figure.

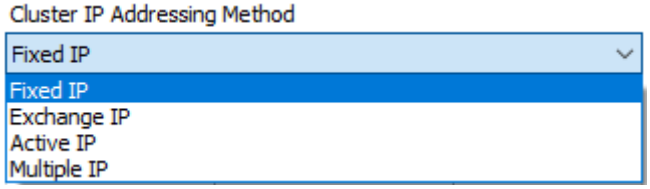


Figure 55: IP Addressing Method for the NX5000 in Redundant CPUs

These IP addresses, and the four addressing methods, are described in section IP Address Configuration on NX5000 Modules in a Redundant CPU Setup.

5.5.2. Ethernet Port Mode Configuration for the NX5000

The NX5000 modules can be inserted in the project to increase the number of Ethernet interfaces if the integrated CPU interfaces are not enough.

The Ethernet channels of the NX5000 modules can be used individually, or arranged in redundant pairs.

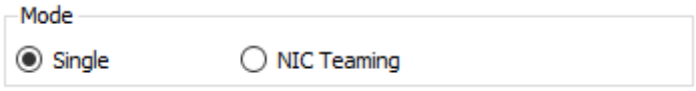


Figure 56: Ethernet Port Mode Configuration for the NX5000

It is possible to select one of the following two modes for an Ethernet port:

- The *Simple* mode indicates that the NX5000 operates individually (without redundancy).
- The *NIC Teaming* mode indicates that this NX5000 and the NX5000 to its right in the rack form a redundant NIC Teaming pair. For more details, see section NIC Teaming Mode on the NX5000.

5.5.2.1. NIC Teaming Mode on the NX5000

A set of two Ethernet ports forming a redundant pair has a single IP address associated with the port pair. In this way, a client such as a SCADA system or Mastertool, connected to a server on the CP, does not need to handle IP address changes in the event of a failure in any of the ports of the redundant pair.

To group two NX5000 modules as a redundant pair, the two modules must be installed in adjacent positions in the rack, and the *NIC Teaming* checkbox of the left-hand module must be selected. When this is done, parameter editing on the right-hand module is locked. The parameters edited on the left-hand module then apply to both modules.

Conversely, clearing the *NIC Teaming* checkbox of the left-hand module separates the modules, which then operate as individual, non-redundant modules.

When the *NIC Teaming* mode is selected, additional parameters are automatically enabled on the same screen and must be configured:

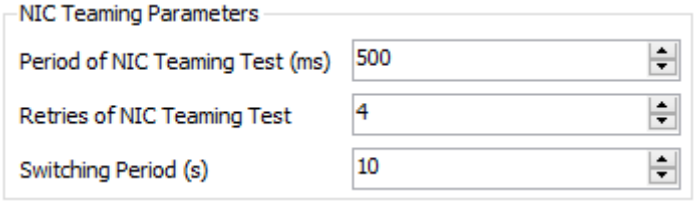


Figure 57: Additional Parameter Configuration for NIC Teaming on the NX5000

- **Redundancy Test Period (ms):** Interval for sending the communication test frame between the two NETs. Can be configured with values between 100 and 9900, default 500.
- **Redundancy Test Retry Count:** Maximum number of times the NET that sent the frame will wait for a response. Can be configured with values between 1 and 100, default 4.
- **Switchover Period (s):** Maximum time the Active NET will wait for any packet. Can be configured with values between 1 and 25, default 10.

If the response time of the *Redundancy Test* reaches the *Test Period* multiplied by the *Retry Count*, and the active interface remains without receiving any packet for a time longer than the *Switchover Period*, a switchover will occur, making the previously inactive interface active. It is important to note that there is a delay between failure detection and activation of the inactive interface due to the time required for its configuration. This delay may reach several tens of milliseconds.

When one of the NETs is active, it will assume the configured IP address, while the inactive NET will have its IP Address, Subnet Mask, and Gateway Address parameters left blank in the CPU diagnostics.

5.5.3. Ethernet Port Failure Mode Configuration

The following screen appears only in a redundant PLC project.

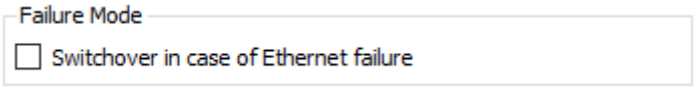


Figure 58: NX5000 Ethernet Port Failure Mode Configuration

5.5.4. Other Settings of the NX5000 Module

Double-clicking an NX5000 module opens several tabs with additional configuration options.

5.5.4.1. "Module Parameters" Tab

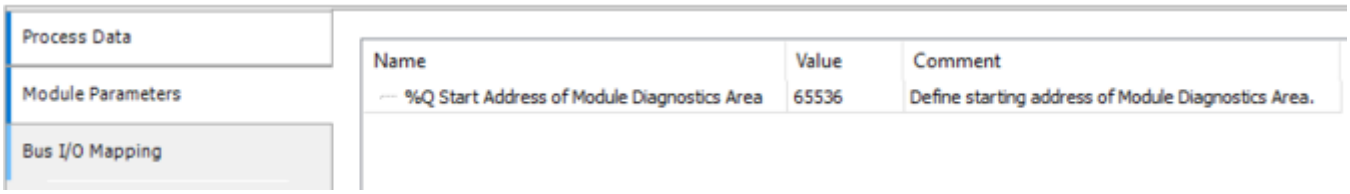


Figure 59: NX5000 *Module Parameters* Tab

The *%Q Start Adress of Module Diagnostics Area* defines the initial %Q address where the diagnostics for this NX5000 will be stored. This address is typically configured appropriately by Mastertool, but the user may modify it if desired.

5.5.4.2. "Bus I/O Mapping" Tab

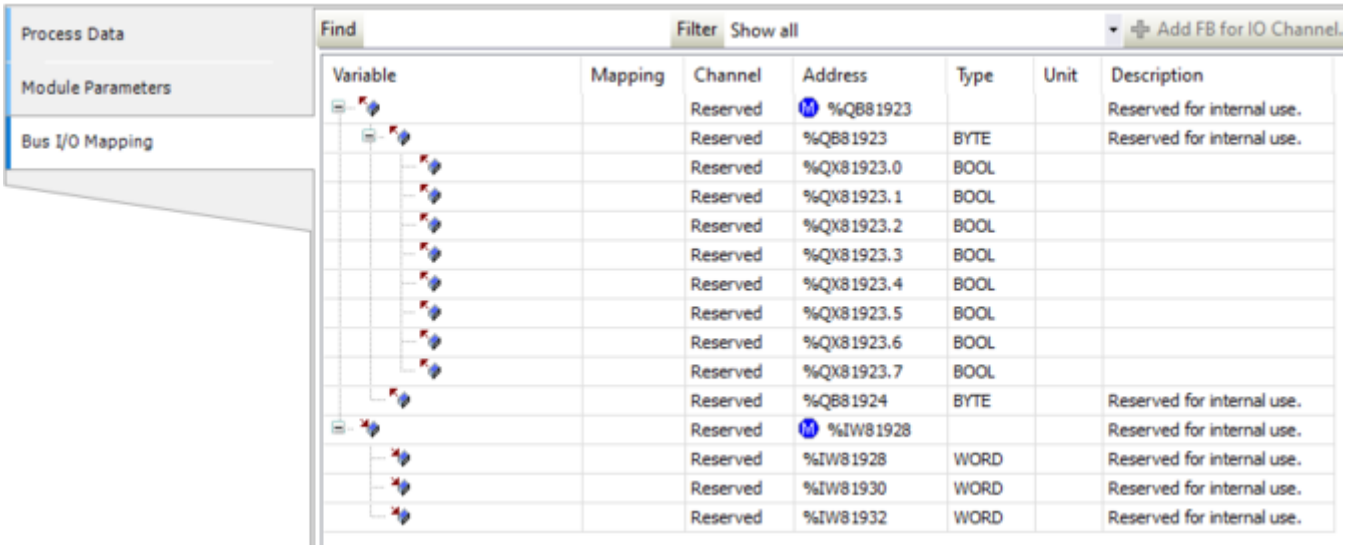


Figure 60: NX5000 *Bus I/O Mapping* Tab

This tab allocates 2 bytes of %Q for user commands, as well as 6 bytes of %I for quick diagnostics of the NX5000. These addresses are typically configured appropriately by Mastertool, but the user may modify them if desired

5.6. PROFIBUS Network Configuration with NX5001 Modules

The NX3035 CPU can be connected to PROFIBUS DP networks via the NX5001 PROFIBUS master module.

Up to six NX5001 modules can be installed in the same backplane rack as the CPU NX3035. The connected PROFIBUS networks can be simple or redundant. A single NX5001 is required for connection to a simple PROFIBUS. For connection to a redundant PROFIBUS network, two NX5001 modules are required, positioned side by side in the backplane rack.

This way, you can have up to six simple PROFIBUS networks, or up to three redundant PROFIBUS networks. Other possibilities also exist as long as the limit of six NX5001 modules is not exceeded (for example, two redundant PROFIBUS networks and two simple PROFIBUS networks). If the PROFIBUS network "n" (where "n" is a number from 1 to 6) is redundant, the two networks that comprise it are named PROFIBUS "n"A and PROFIBUS "n"B. If the PROFIBUS network 'n' is simple, the network that comprises it is named PROFIBUS "n"A.

The following figure shows an example of an architecture where the NX3035 CPU connects to two PROFIBUS DP networks using three NX5001 modules. The first network is redundant (PROFIBUS 1A and PROFIBUS 1B). The second network is simple (PROFIBUS 2A).

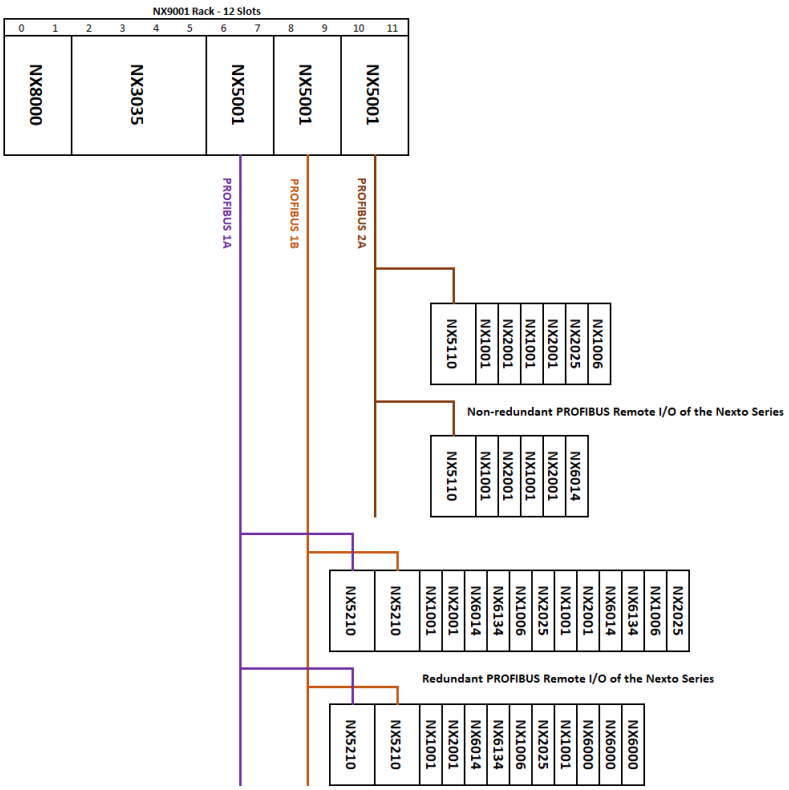


Figure 61: Example of connecting the NX3035 CPU to two PROFIBUS networks

5.6.1. Inserting or Removing NX5001 Modules

NX5001 modules can be inserted or removed from the rack of a PLC with the NX3035 CPU. To perform this operation correctly, you must be aware of the following rules:

- NX5001 modules can only be installed in the same backplane rack as the NX3035 CPU, in a position to the right of the NX3035 CPU.
- The number of NX5001 modules in this backplane rack can vary between zero and six.
- When a PROFIBUS network is simple, it requires a single NX5001 module. When it is redundant, it requires two NX5001 modules.
- Two NX5001 modules used to compose a redundant PROFIBUS network must occupy adjacent positions in the backplane rack.

5.6.2. NX5001 Module Parameter Adjustment

Each NX5001 module used in a simple PROFIBUS network, or each pair of NX5001 modules used in a redundant PROFIBUS network, has several parameters that must be adjusted. After double-clicking on the NX5001 module, several tabs appear with settings that are described in the following subsections.

Advanced details on PROFIBUS network configuration can be found in the **NX5001 PROFIBUS DP Master User Manual**.

5.6.2.1. "General" Tab

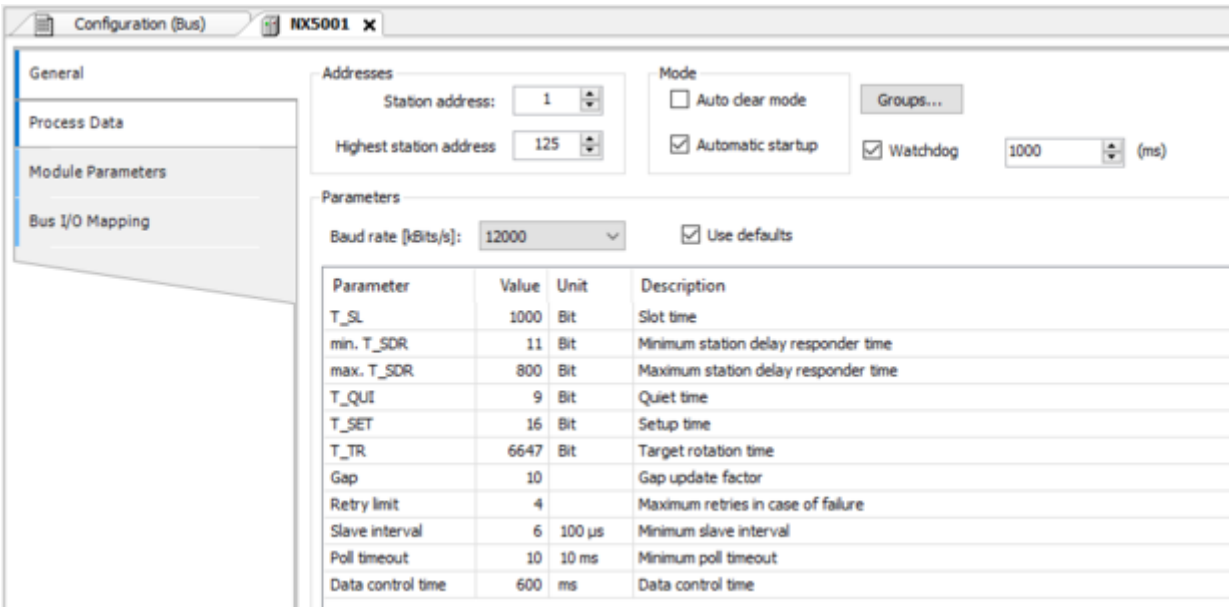


Figure 62: NX5001 *General* Tab

The *Station Addr.* parameter indicates the address of the NX5001 master on the PROFIBUS network.

It is recommended to keep the value 1 whenever possible.

ATTENTION

In a redundant PLC (see chapter Redundancy with NX3035 CPU), the *Station Addr.* parameter corresponds to the address of the NX5001 installed in the Active PLC, while this address minus 1 corresponds to the address of the NX5001 installed in the Non-Active PLC. Therefore, in a redundant PLC where *Station Addr.* is 1, two addresses are allocated to the PROFIBUS network: address 1 for the NX5001 in the Active PLC, and address 0 for the NX5001 in the Non-Active PLC.

The remaining PROFIBUS addresses should normally be allocated as follows:

- **2:** for a class 2 master, for example, used to collect asset management data from PROFIBUS slaves via the DP-V1 protocol.
- **3 to 125:** for NX5001 slaves on the PROFIBUS network, such as Nexto Series PROFIBUS remote I/O stations.
- Address 126 is widely used by slave devices when they leave the factory, so you should avoid using it.
- Address 127 is used to send broadcast frames and should also not be used.

The *Highest station address* parameter can be set to 125, but to optimize PROFIBUS network performance, it can be set to the highest address of an NX5001 slave on the PROFIBUS network.

For more details on other fields, refer to the **PROFIBUS DP NX5001 Master User Manual.**

The *Watchdog* checkbox and the watchdog time next to it have no effect on the PROFIBUS remotes typically used (Nexto Series or Ponto Series). These settings are intended for other special remotes. The *Watchdog* parameters for Nexto and Ponto series PROFIBUS remotes must be configured on the screens of these remotes (e.g., NX5210, NX5110, PO5065).

The *Baud Rate [kBits/s]* parameter selects the speed of the PROFIBUS network, with several options between 9.6 kbps and 12000 kbps. Next to this baud rate, it is recommended to leave the *Use defaults* checkbox checked, so that Mastertool automatically calculates several parameters shown below (*T_SL*, *T_SDR min.*, etc.). Otherwise, the user can change these parameters, which requires in-depth knowledge of the PROFIBUS network.

5.6.2.2. "Module Parameters" Tab

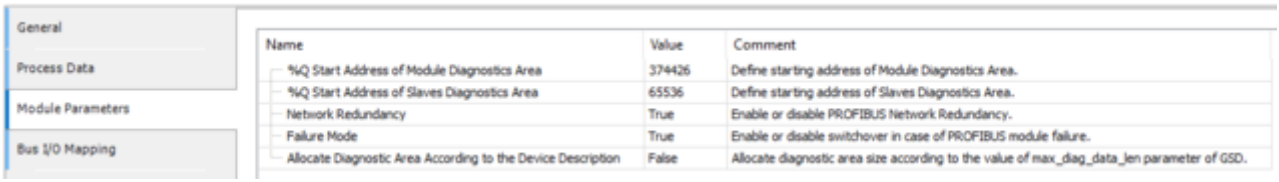


Figure 63: NX5001 *Module Parameters* Tab

The *Initial Module Diagnostic Address in %Q* defines the initial %Q address where the diagnostics for this NX5001 will be stored. This address is normally set satisfactorily by Mastertool, but the user can change it if desired.

The *Initial Diagnosis Address of Slaves in %Q* defines the initial %Q address from which the diagnostics of remote heads and remote PROFIBUS I/O modules installed below this NX5001 will be placed. This address is normally set satisfactorily by Mastertool, but the user can change it if desired.

To group two NX5001 modules in a redundant PROFIBUS network, select an ungrouped NX5001 module that has another ungrouped NX5001 module to its right in the backplane rack.

Next, set the *Network Redundancy* parameter to *True*. To ungroup them, simply follow the same procedure, but set the *Network Redundancy* parameter to *False*. If this parameter is set to True, the parameters of the NX5001 module on its right will be locked for editing, as they will be calculated automatically from the parameters of this NX5001 on the left.

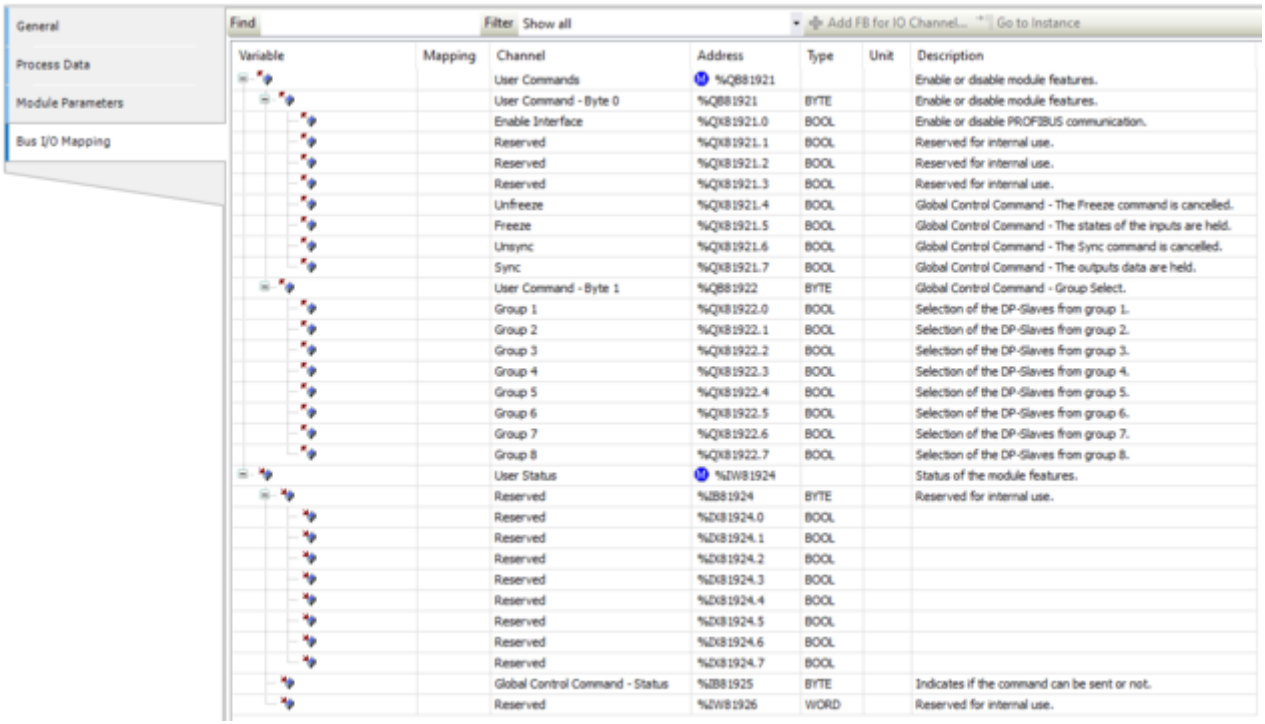
The *Failure Mode* parameter applies only to a redundant PLC (see chapter Redundancy with NX3035 CPU), and has two options:

- **True:** a total failure of the PROFIBUS network will cause a switch-over between Active and Standby PLCs. Total failure is characterized when the NX5001 master of the simple network, or both NX5001 masters of the redundant network, cannot communicate with any of their slaves.
- **False:** A total failure of the PROFIBUS network will not cause a switchover between Active and Standby PLCs.

The parameter *Allocate Diagnostic Area According to Device Description* defines the number of redundant diagnostic bytes in %Q that will be allocated to third-party remotes (does not apply to Altus PROFIBUS remotes):

- If the value is *False*, it will always allocate 244 bytes of diagnostics to third-party remotes.
- If the value is *True*, it will allocate the number of bytes defined in the third-party remote's GSD file using the *Max_Diag_Data_Len* parameter.

5.6.2.3. "Bus I/O Mapping" Tab



Variable	Mapping	Channel	Address	Type	Unit	Description
		User Commands	%QB81921			Enable or disable module features.
		User Command - Byte 0	%QB81921	BYTE		Enable or disable module features.
		Enable Interface	%QX81921.0	BOOL		Enable or disable PROFIBUS communication.
		Reserved	%QX81921.1	BOOL		Reserved for internal use.
		Reserved	%QX81921.2	BOOL		Reserved for internal use.
		Reserved	%QX81921.3	BOOL		Reserved for internal use.
		Unfreeze	%QX81921.4	BOOL		Global Control Command - The Freeze command is cancelled.
		Freeze	%QX81921.5	BOOL		Global Control Command - The states of the inputs are held.
		Unsync	%QX81921.6	BOOL		Global Control Command - The Sync command is cancelled.
		Sync	%QX81921.7	BOOL		Global Control Command - The outputs data are held.
		User Command - Byte 1	%QB81922	BYTE		Global Control Command - Group Select.
		Group 1	%QX81922.0	BOOL		Selection of the DP-Slaves from group 1.
		Group 2	%QX81922.1	BOOL		Selection of the DP-Slaves from group 2.
		Group 3	%QX81922.2	BOOL		Selection of the DP-Slaves from group 3.
		Group 4	%QX81922.3	BOOL		Selection of the DP-Slaves from group 4.
		Group 5	%QX81922.4	BOOL		Selection of the DP-Slaves from group 5.
		Group 6	%QX81922.5	BOOL		Selection of the DP-Slaves from group 6.
		Group 7	%QX81922.6	BOOL		Selection of the DP-Slaves from group 7.
		Group 8	%QX81922.7	BOOL		Selection of the DP-Slaves from group 8.
		User Status	%IW81924			Status of the module features.
		Reserved	%IB81924	BYTE		Reserved for internal use.
		Reserved	%IX81924.0	BOOL		
		Reserved	%IX81924.1	BOOL		
		Reserved	%IX81924.2	BOOL		
		Reserved	%IX81924.3	BOOL		
		Reserved	%IX81924.4	BOOL		
		Reserved	%IX81924.5	BOOL		
		Reserved	%IX81924.6	BOOL		
		Reserved	%IX81924.7	BOOL		
		Global Control Command - Status	%IB81925	BYTE		Indicates if the command can be sent or not.
		Reserved	%IW81926	WORD		Reserved for internal use.

Figure 64: NX5001 *Bus I/O Mapping* Tab

This tab allocates 2 bytes of %Q for user commands, in addition to 4 bytes of %I for quick diagnostics of the NX5001. These addresses are normally set satisfactorily by Mastertool, but the user can change them if desired.

ATTENTION

In *User Commands*, *User Commands – Byte 0*, the user should not modify the value of bit 0 (*Enable Interface*), as this bit is manipulated by the application.

5.6.3. PROFIBUS Remote Settings

To configure PROFIBUS remotes under an NX5001 master, consult the following manuals in addition to the **PROFIBUS DP NX5001 Master User Manual**:

- Ponto Series User Manual
- PROFIBUS PO5064 Head User Manual and PROFIBUS PO5065 Redundant Head User Manual
- HART over PROFIBUS Network User Manual
- PROFIBUS-DP Nexto Head User Manual

5.6.3.1. Remote Stations Compatible with Redundant PROFIBUS Networks

Currently, only three models of remote heads manufactured by Altus can be connected to a redundant PROFIBUS network:

- PO5063V5
- PO5065
- NX5210

5.6.3.2. Watchdog Configuration on PROFIBUS Remote Stations in Redundant PLCs

In a redundant PLC (see chapter Redundancy with NX3035 CPU), special care must be taken when configuring the watchdog parameter of the remotes. This applies to both simple and redundant PROFIBUS networks, since it is possible to use simple PROFIBUS networks in redundant PLCs.

If the *Watchdog Control* checkbox is enabled on the remote configuration screen, the *Time* field must be configured correctly. The watchdog time must be greater than or equal to the longest time between WD1 and WD2:

- WD1 = I x 2 + 500 ms
- WD2 = I x 3

In the previous equations, I is the interval configured for the MainTask. For example, for an interval of 200 ms for MainTask, the minimum watchdog value configured on each remote must be 900 ms. Therefore, a watchdog of 1000 ms is adequate.

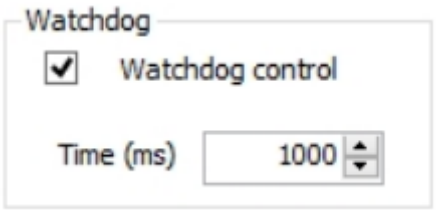


Figure 65: PROFIBUS Remote Watchdog Configuration in a Redundant PLC

WARNING

In the case of more than one PROFIBUS network, the watchdog time must be at least 2000 ms.

5.7. Protocols Configuration

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

	NX3035
Mapped Points	512000
Mappings (Per Instance / Total)	5120 / 20480
Requests	512
NETs – Client or Server Instances (Per NET / Total)	4 / 16
COM (n) – Master or Slave Instances	1
Control Centers	-

Table 45: Protocol Limits per CPU

Notes:

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

Mappings: A "mapping" is a relation between an internal application variable and an object of the application protocol. This field indicates the maximum number of mappings supported by the CPU. It corresponds to the sum of all mappings performed in the protocol instances and their respective devices.

Requests: The sum of communication protocol requests, declared in the devices, must not exceed the maximum number of requests supported by the CPU.

NETs – Client or Server Instances: This field defines the maximum number of protocol instances per Ethernet interface, as well as the total maximum distributed among all Ethernet interfaces in the system.

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

Control Centers: A "Control Center" is any client device connected to the CPU through the IEC 60870-5-104 protocol. This field indicates the maximum number of client devices, of the control center type, supported by the CPU. It corresponds to the sum of all client devices from the IEC 60870-5-104 Server protocol instances (does not include masters and clients of MODBUS RTU Slave, MODBUS Server, and DNP3 Server protocols).

The limitations of the MODBUS protocol for Direct Representation and symbolic mappings for CPUs can be seen in Table 46 and Table 47, respectively.

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

Table 46: MODBUS Protocol Limitations for Direct Representation

Notes:

Devices per instance:

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

Mappings per device: The maximum number of mappings per device, despite being listed above, is also limited by the protocol maximum number of mappings. Also to be considered the maximum CPU mappings as in Table 45.

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 from MODBUS Clients. 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 47: MODBUS Protocol Limitations for Symbolic Mappings

Notes:

Devices per instance:

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

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

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

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

5.7.1. Protocol Behavior x CPU State

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

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

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
FTP	Server	✓	✓	✓	✓	✓	✓
Syslog	Client	✓	✓	✓	✓	✓	✓

Table 48: Protocol Behavior x CPU State

Notes:

Symbol ✓: Protocol remains active and operating normally.

Symbol ✗: Protocol is disabled.

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

Tests performed using the default value defined in the *PLC Settings* (see Table PLC Settings), with the option *Update I/O while in stop* enabled and the option *Set all outputs to default* configured.

5.7.2. MODBUS RTU Master

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

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 User Manual or in the section Inserting a Protocol Instance. The remaining configuration steps are described below for each mode.

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

5.7.2.1. MODBUS Master Protocol Configuration by Symbolic Mapping

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

- Configure the general parameters of the MODBUS Master protocol, like: transmission delay times and minimum interframe as in Figure 66.
- 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 67.
- Add and configure the MODBUS mappings on Mappings tab as Figure 68, 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 69, specifying the function, the scan time of the request, the starting address (read/write), the data size (read/write) and generate diagnostic variables and disabling the request via the buttons at the bottom of the window.

5.7.2.1.1. MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration

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

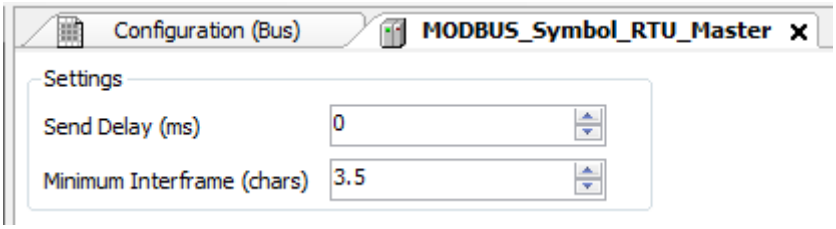


Figure 66: MODBUS RTU Master Configuration Screen

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

Table 49: MODBUS RTU Master General Settings

Notes:

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

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

The MODBUS protocol diagnostics and commands configured, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_RTU_MASTER_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below:

T_DIAG_MODBUS_RTU_MASTER_1.*	Size	Description
Diagnostic Bits:		
tDiag.bRunning	BIT	The master is running.
tDiag.bNotRunning	BIT	The master is not running (see bit: bInterruptedByCommand).
tDiag.bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the master was interrupted by the user through command bits.
tDiag.bConfigFailure	BIT	Configuration failure.
tDiag.bModuleFailure	BIT	Not implemented.
Error codes:		
eErrorCode	SERIAL_STATUS (BYTE)	0: there are no errors 1: invalid serial port 2: invalid serial port mode 3: invalid baud rate 4: invalid data bits 5: invalid parity 6: invalid stop bits 7: invalid modem signal parameter 8: invalid UART RX Threshold parameter 9: invalid time-out parameter 10: busy serial port 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid signal modem method 22: CTS time-out = true 23: CTS time-out = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from manual 33: invalid flow control for the configured serial port

T_DIAG_MODBUS_RTU_MASTER_1.*	Size	Description
		34: data reception not allowed in normal mode 35: data reception not allowed in extended mode 36: DCD interruption not allowed 37: CTS interruption not allowed 38: DSR interruption not allowed 39: serial port not-configured 50: internal error in the serial port
Command bits, automatically initialized:		
tCommand.bStop	BIT	Stop master.
tCommand.bRestart	BIT	Restart master.
tCommand.bResetCounter	BIT	Restart diagnostics statistics (counters).
Communication Statistics:		
tStat.wTXRequests	WORD	Counter of request transmitted by the master (0 to 65535).
tStat.wRXNormalResponses	WORD	Counter of normal responses received by the master (0 to 65535).
tStat.wRXExceptionResponses	WORD	Counter of responses with exception codes received by the master (0 to 65535).
tStat.wRXIllegalResponses	WORD	Counter of illegal responses received by master – invalid syntax, not enough received bytes, invalid CRC – (0 to 65535).
tStat.wRXOverrunErrors	WORD	Counter of overrun errors during reception - UART FIFO or RX line – (0 to 65535).
tStat.wRXIncompleteFrames	WORD	Counter of answers with construction errors, parity or failure during reception (0 to 65535).
tStat.wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using RTS/CTS handshake, during transmission (0 to 65535).

Table 50: MODBUS RTU Master Diagnostics

Note:

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

5.7.2.1.2. Devices Configuration – Symbolic Mapping configuration

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

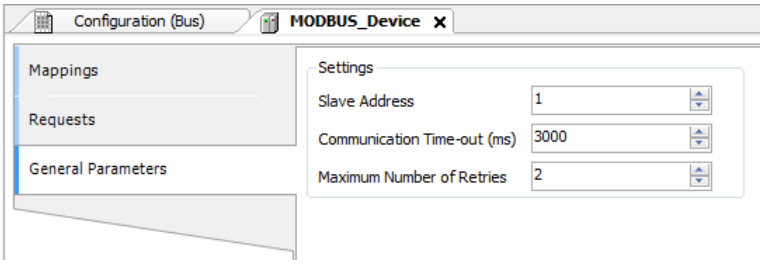


Figure 67: Device General Parameters - Settings

Settings	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 51: Device Settings

Notes:

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

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

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

5.7.2.1.3. Mappings Configuration – Symbolic Mapping Settings

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

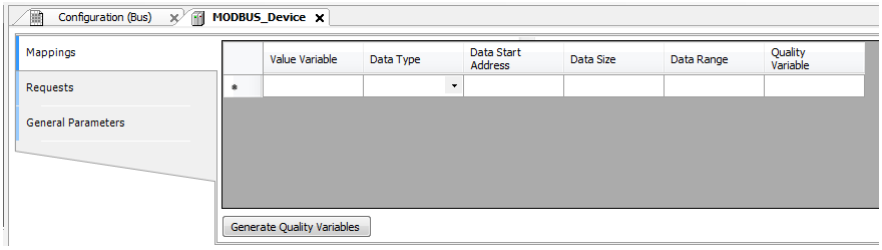


Figure 68: MODBUS Data Mappings Screen

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

Table 52: MODBUS Mappings Settings

Notes:

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

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

Data Type	Size [bits]	Description
Coil - Write	1	Writing digital output.

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

Table 53: Data Types Supported in MODBUS

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

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

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

5.7.2.1.4. Requests Configuration – Symbolic Mapping Settings

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

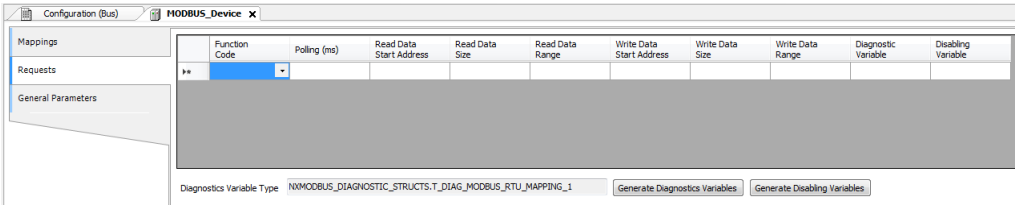


Figure 69: Data Requests Screen MODBUS Master

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

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

Table 54: MODBUS Relations Configuration

Notes:

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

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

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

Table 55: MODBUS Functions Supported by Nexto CPUs

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

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

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

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

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

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

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

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

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

Diagnostic Variable: The MODBUS request diagnostics configured by symbolic mapping or by direct representation, are stored in variables of type *T_DIAG_MODBUS_RTU_MAPPING_1* for Master devices and *T_DIAG_MODBUS_ETH_CLIENT_1* for Client devices and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 56.

T_DIAG_MODBUS_RTU_MAPPING_1.*	Size	Description
Communication Status Bits:		
byStatus.bCommIdle	BIT	Communication idle (waiting to be executed).
byStatus.bCommExecuting	BIT	Active communication.
byStatus.bCommPostponed	BIT	Communication deferred, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of time-out. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true.
byStatus.bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
byStatus.bCommOk	BIT	Communication terminated previously was held successfully.
byStatus.bCommError	BIT	Communication terminated previously had an error. Check error code.
Last error code (enabled when bCommError = true):		
eLastErrorCode	MASTER_ERROR_CODE (BYTE)	Informs the possible cause of the last error in the MODBUS mapping. Consult Table 79 for further details.
Last exception code received by master:		
eLastExceptionCode	MODBUS_EXCEPTION (BYTE)	NO_EXCEPTION (0) FUNCTION_NOT_SUPPORTED (1) MAPPING_NOT_FOUND (2) ILLEGAL_VALUE (3) ACCESS_DENIED (128)* MAPPING_DISABLED (129)* IGNORE_FRAME (255)*
Communication statistics:		
wCommCounter	WORD	Communications counter terminated, with or without errors. The user can test when communication has finished testing the variation of this counter. When the value 65535 is reached, the counter returns to zero.
wCommErrorCounter	WORD	Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero.

Table 56: MODBUS RTU Relations Diagnostics

Notes:

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

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

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

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

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

Table 57: MODBUS Relations Error Codes

ATTENTION

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

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

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

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

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

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

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

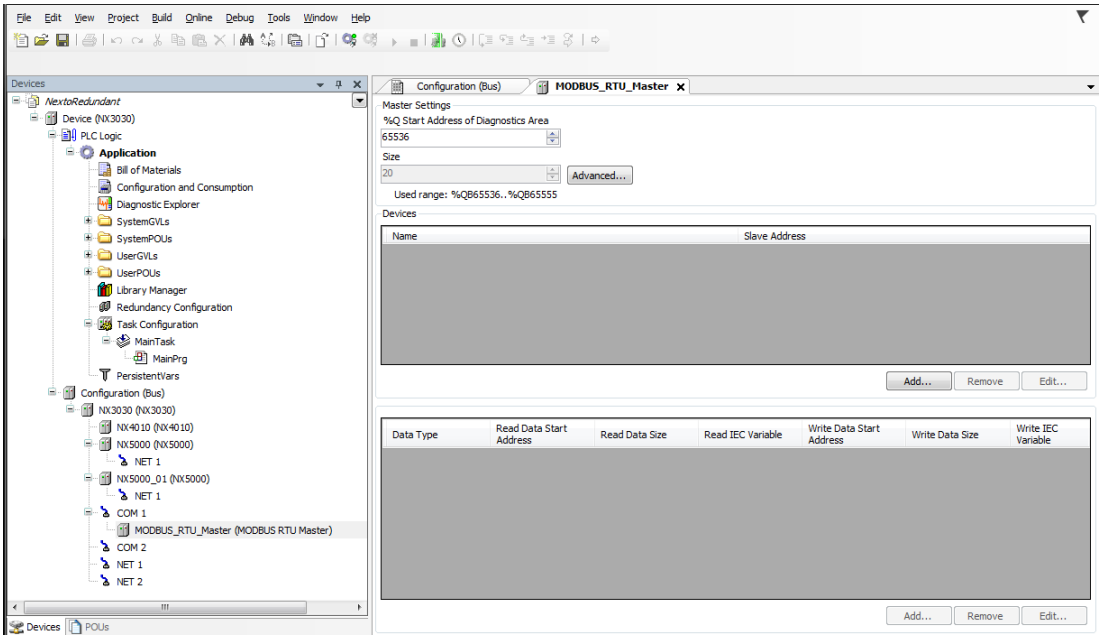


Figure 70: MODBUS RTU Master Setup Screen

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

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

Table 58: MODBUS RTU Master Configuration

Notes:

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

Default Value: the factory default value cannot be set to the *%Q Start Address of Diagnostics Area* field, because the creation of a Protocol instance may be held at any time on application development. The Mastertool 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 50.

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

5.7.2.2.2. Devices Configuration – Configuration for Direct Representation (%Q)

The configuration of the devices, viewed in figure below, comprises the following parameters:

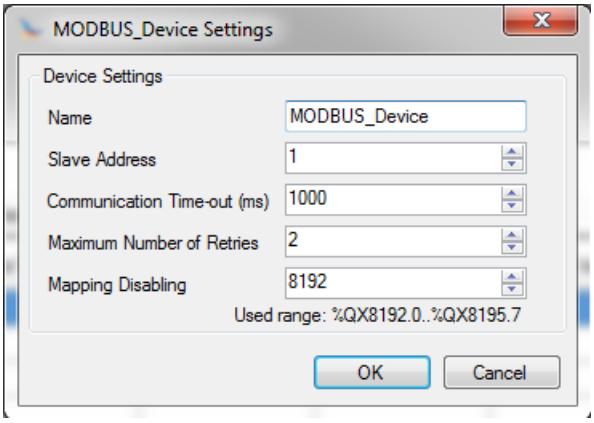


Figure 71: Device Configuration

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

Table 59: Device Configuration - MODBUS Master

Notes:

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

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

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

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

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

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

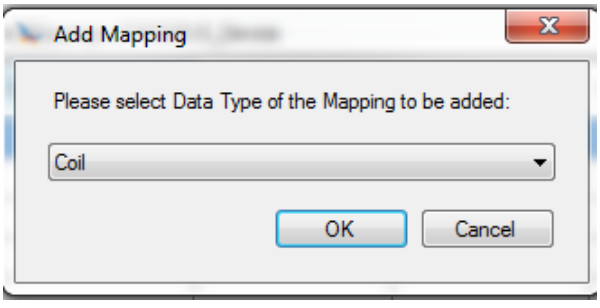


Figure 72: MODBUS Data Type

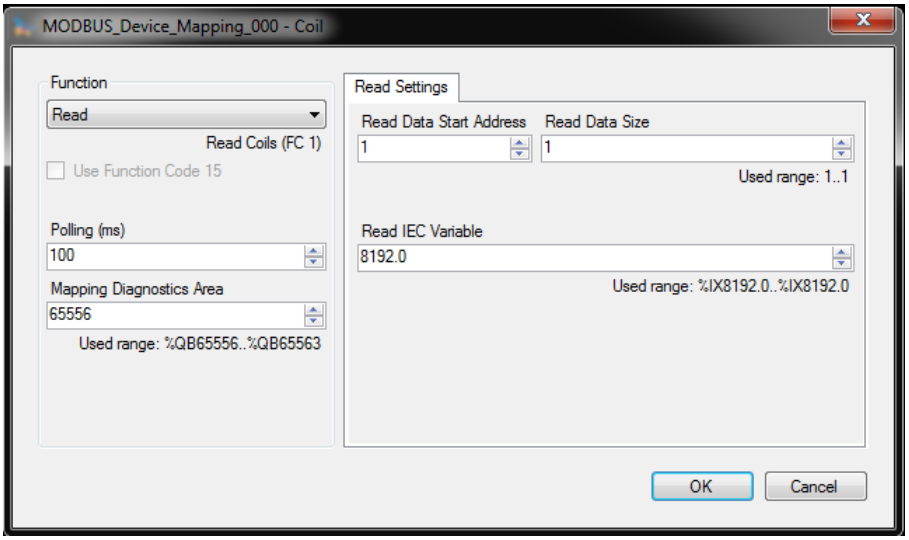


Figure 73: MODBUS Function

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

Configuration	Description	Default Value	Options
Function	MODBUS function type	Read	Read Write Read/Write Mask Write
Polling (ms)	Communication period (ms)	100	0 to 3600000
Mapping Diagnostics Area	Initial address of the MODBUS relation diagnostics (%Q)	-	0 to 2147483640
Read Data Start Address	Initial address of the MODBUS read data	1	1 to 65536
Read Data Size	Number of MODBUS read data	-	Depends on the function used
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
Mask Write IEC Variables	Initial address of the variables for the write mask (%Q)	-	0 to 2147483644

Table 60: Device Mapping

Notes:

Function: the available data types are detailed in the Table 79 and MODBUS functions (FC) are available in the Table 77.

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

Mapping Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in the section Memory. The configured MODBUS relations diagnostics are described in Table 56.

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

ATTENTION

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

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

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

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

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

ATTENTION

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

5.7.3. MODBUS RTU Slave

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

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 User Manual. The remaining configuration steps are described below for each mode:

- Add the MODBUS RTU Slave Protocol instance to the serial channel COM 1 or COM 2 (or both, in cases of two communication networks). To execute this procedure see Inserting a Protocol Instance section.
- Configure the serial interface, choosing the communication speed, the RTS/CTS signals behavior, the parity, the stop bits channel, among others. See Serial Interface Configuration section.

5.7.3.1. MODBUS Slave Protocol Configuration via Symbolic Mapping

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

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

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

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

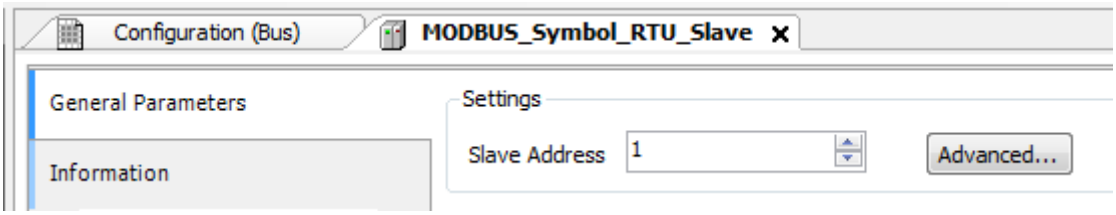


Figure 74: MODBUS RTU Slave Settings Screen

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

Table 61: Slave Settings

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

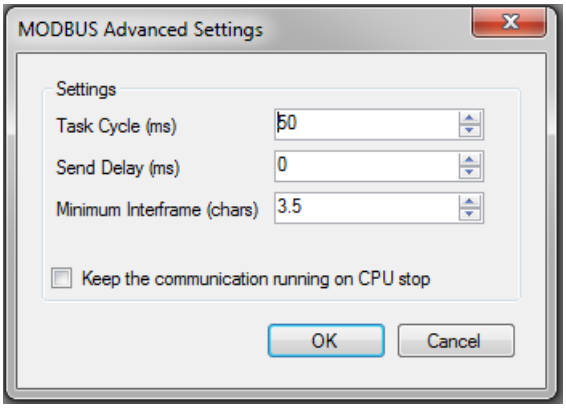


Figure 75: Modbus Slave Advanced Settings

Settings	Description	Default	Options
Task Cycle (ms)	Time for the instance execution within the cycle, without considering its own execution time	20	5 to 100
Send Delay (ms)	Delay for the transmission response	0	0 to 65535
Minimum Interframe (chars)	Minimum silence time between different frames	3.5	3.5 to 100.0
Keep the communication running on CPU stop	Enable the MODBUS Symbol Slave to run while the CPU is in STOP or standing in a breakpoint	Unchecked	Checked or unchecked

Table 62: Modbus Slave Advanced Settings

Notes:

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

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

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

Keep the communication running on CPU stop: This parameter is valid only for non-redundant PLCs and is not available for redundant PLCs. The MODBUS Slave protocol diagnostics and commands configured, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_RTU_SLAVE_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below:

T_DIAG_MODBUS_RTU_SLAVE_1.*	Size	Description
Diagnostic Bits:		
tDiag.bRunning	BIT	The slave is in execution mode.
tDiag.bNotRunning	BIT	The slave is not in execution (see bit: bInterruptedBy-Command).
tDiag.bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the slave was interrupted by the user through command bits.
tDiag.bConfigFailure	BIT	Configuration failure.
tDiag.bModuleFailure	BIT	Not implemented.
Error codes:		
eErrorCode	SERIAL_STATUS (BYTE)	0: there are no errors 1: invalid serial port 2: invalid serial port mode 3: invalid baud rate 4: invalid data bits 5: invalid parity 6: invalid stop bits 7: invalid modem signal parameter 8: invalid UART RX Threshold parameter 9: invalid time-out parameter 10: busy serial port 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid signal modem method 22: CTS time-out = true 23: CTS time-out = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from manual 33: invalid flow control for the configured serial port 34: data reception not allowed in normal mode 35: data reception not allowed in extended mode 36: DCD interruption not allowed 37: CTS interruption not allowed 38: DSR interruption not allowed 39: serial port not-configured 50: internal error in the serial port
Command bits, automatically initialized:		
tCommand.bStop	BIT	Stop slave.
tCommand.bRestart	BIT	Restart slave.
tCommand.bResetCounter	BIT	Restart diagnostics statistics (counters).
Communication Statistics:		
tStat.wRXRequests	WORD	Counter of normal requests received by the slave and answered normally. In case of a broadcast command, this counter is incremented, but it is not transmitted (0 to 65535).
tStat.wTXExceptionResponses	WORD	Counter of normal requests received by the slave and answered with exception code. In case of a broadcast command, this counter is incremented, but it isn't transmitted (0 to 65535).
tStat.wRXFrames	WORD	Counter of frames received by the slave. It's considered a frame something which is processed and it is followed by a Minimum Interframe Silence, in other words, an illegal message is also computed (0 to 65535).

T_DIAG_MODBUS_RTU_SLAVE_1.*	Size	Description
tStat.wRXIllegalRequests	WORD	Illegal request counter. These are frames which start with address 0 (broadcast) or with the MODBUS slave address, but aren't legal requests – invalid syntax, smaller frames, invalid CRC – (0 to 65535).
tStat.wRXOverrunErrors	WORD	Counter of frames with overrun errors during reception – UART FIFO or RX line – (0 to 65535).
tStat.wRXIncompleteFrames	WORD	Counter of frames with construction errors, parity or failure during reception (0 to 65535).
tStat.wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using the RTS/CTS handshake, during the transmission (0 to 65535).

Table 63: MODBUS RTU Slave Diagnostic

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

5.7.3.1.2. Configuration of the Relations – Symbolic Mapping Setting

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

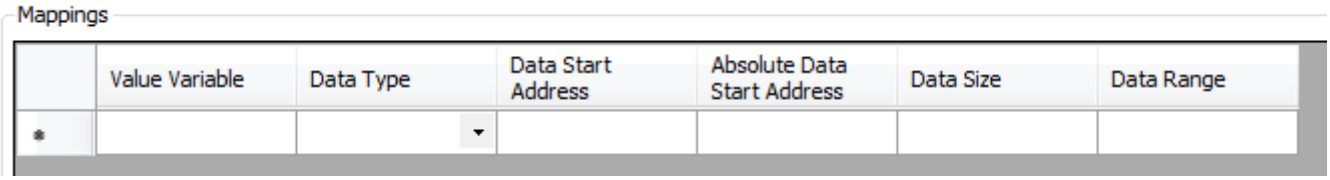


Figure 76: MODBUS Data Mappings Screen

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

Table 64: MODBUS Mappings Configurations

Notes:
Value Variable: this field is used to specify a symbolic variable in MODBUS relation.
Data Type: this field is used to specify the data type used in the MODBUS relation.

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

Table 65: MODBUS data types supported by Nexto CPUs

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

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

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

ATTENTION

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

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

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

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

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

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

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

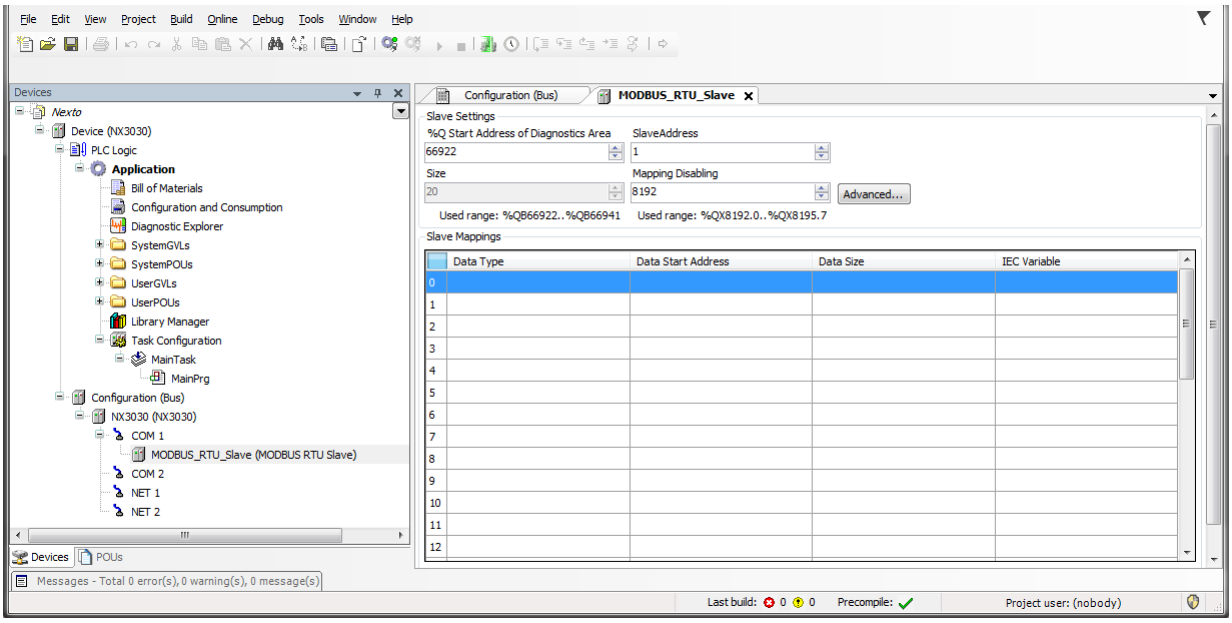


Figure 77: MODBUS RTU Slave Configuration Screen by Direct Representation

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

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

Table 66: Address and Direct Representation Variables Settings

Notes:

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

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

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

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

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

The MODBUS protocol diagnostics and commands are described in the Table 63.

The communication times of the MODBUS Slave protocol, found on the button *Advanced...* of the configuration screen, are described in Table 62.

5.7.3.2.2. *Mappings Configuration – Configuration via Direct Representation (%Q)*

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

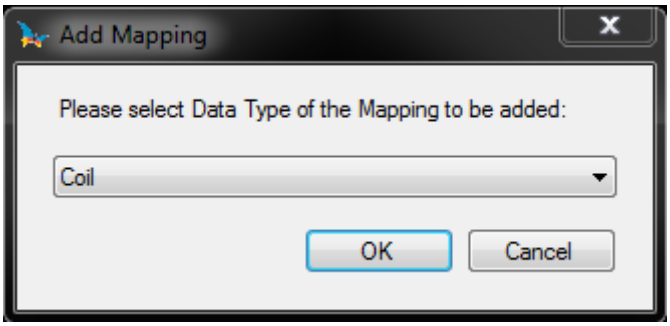


Figure 78: Adding MODBUS Relations

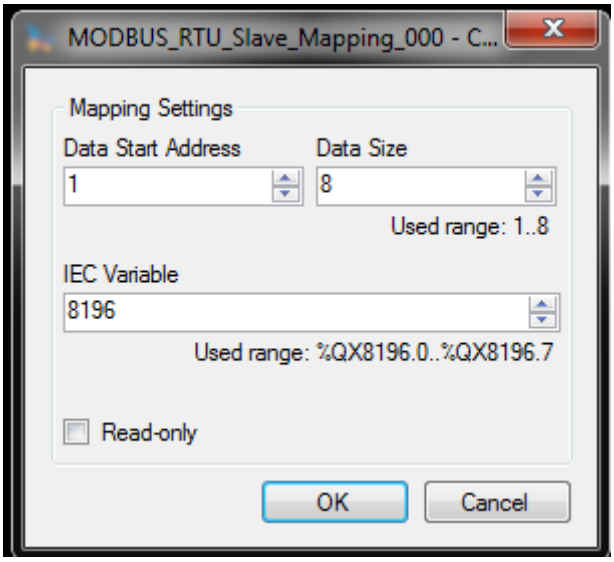


Figure 79: Configuring the MODBUS Relation

Configuration	Description	Default Value	Options
Data Type	MODBUS data type	Coil	Coil (1 bit) Holding Register (16 bits) Input Register (16 bits) Input Status (1 bit)
Data Start Address	Initial address of the 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 67: Slave Mappings

Notes:

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

Data Size: the value of *Data Size* defines the maximum amount of data that a MODBUS relation can access, from the initial address. Thus, to read a continuous address range, it is necessary that all addresses are declared in a single interface. This field varies with the MODBUS data type configured, i.e. when selected *Coil* or *Input Status*, the Data Size field must be a multiple of eight. Also, the maximum amount must not exceed the size of output addressable memory and not assign the same values used in the application.

ATTENTION

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

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

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

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

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

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

ATTENTION

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

5.7.4. MODBUS Ethernet

The multi-master communication allows the Nexto CPUs to read or write MODBUS variables in other controllers or HMIs compatible with the MODBUS TCP protocol or MODBUS RTU via TCP. The Nexto CPU can, at the same time, be client and server in the same communication network, or even have more instances associated to the Ethernet interface. It does not matter if they are MODBUS TCP or MODBUS RTU via TCP, as described on Table 45.

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

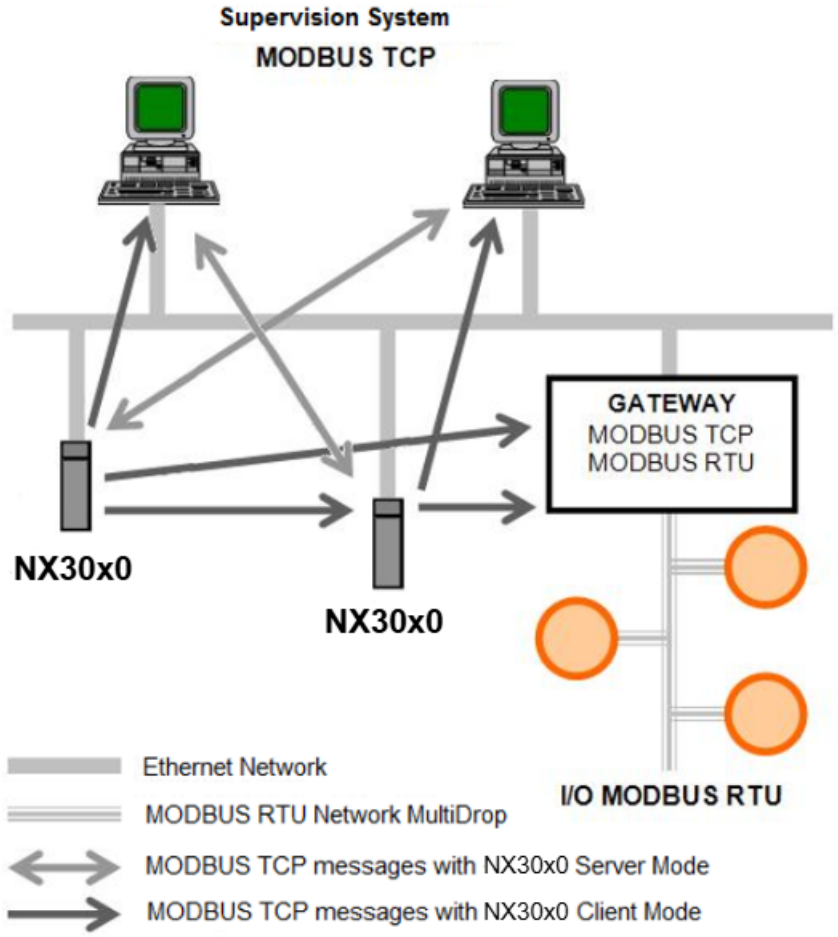


Figure 80: MODBUS TCP Communication Network

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

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

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

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

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

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

Table 68: MODBUS data types supported by Nexto CPUs

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

Table 69: MODBUS Functions Supported by Nexto CPUs

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

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

5.7.5. MODBUS Ethernet Client

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

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 User Manual or on Inserting a Protocol Instance section.

5.7.5.1. MODBUS Ethernet Client Configuration via Symbolic Mapping

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

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

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

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

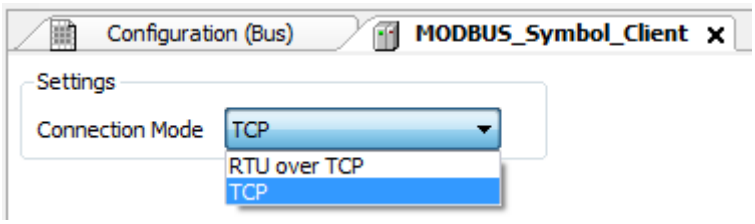


Figure 81: MODBUS Client General Parameters Settings Screen

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

Table 70: MODBUS Client General Settings

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

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

Table 71: MODBUS Client Protocol Diagnostics

Note:

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

5.7.5.1.2. Device Configuration – Configuration via Symbolic Mapping

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

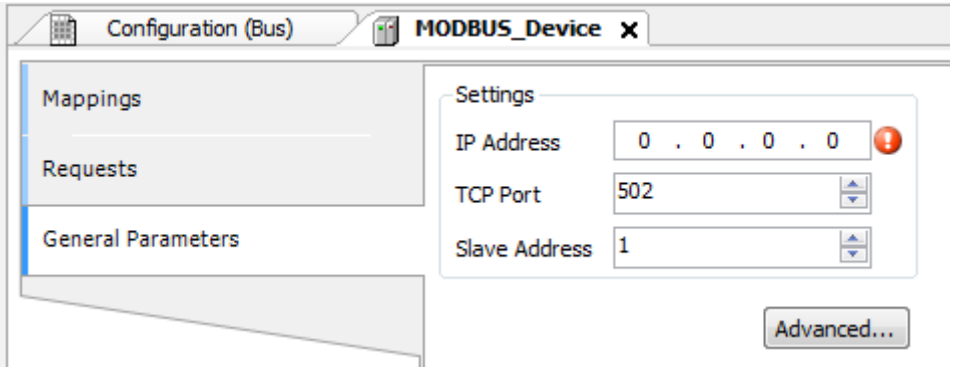


Figure 82: Device General Parameters Settings

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

Table 72: MODBUS Client General Parameters

Notes:

IP Address: IP address of Modbus Server Device.

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

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

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

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

Table 73: MODBUS Client Advanced Configurations

Notes:

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

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

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

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

Inactive Time: inactivity connection time.

5.7.5.1.3. Mappings Configuration – Configuration via Symbolic Mapping

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

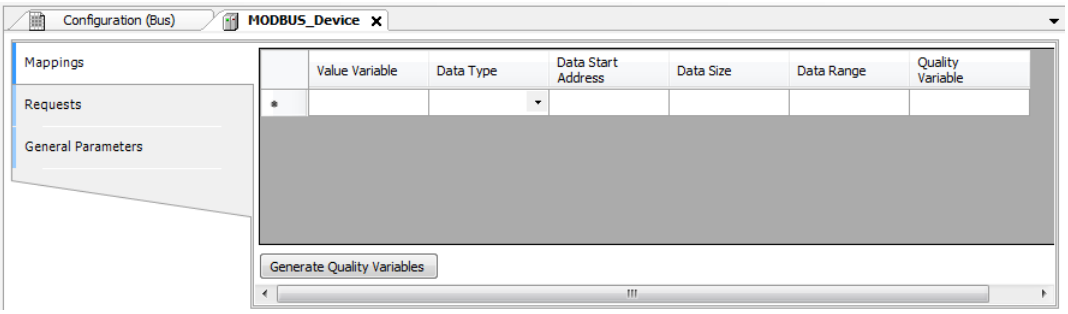


Figure 83: MODBUS Data Type

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

Table 74: MODBUS Mappings Settings

Notes:

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

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

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

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

Table 75: Data Types Supported in MODBUS

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

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

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

5.7.5.1.4. Requests Configuration – Configuration via Symbolic Mapping

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

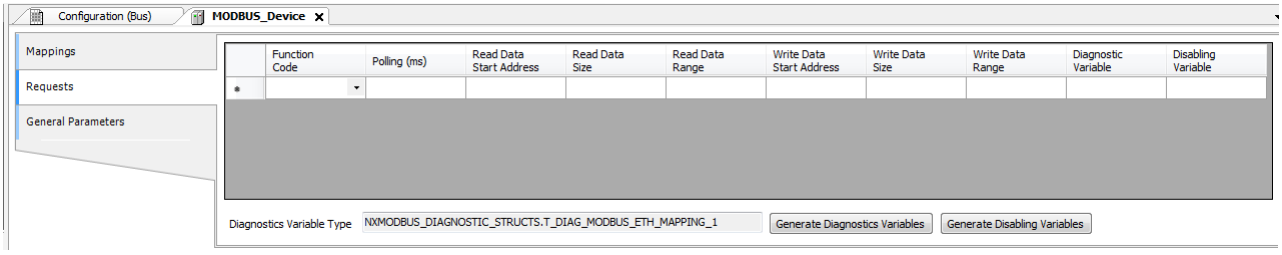


Figure 84: MODBUS Data Request Screen

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

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

Table 76: MODBUS Relations Configuration

Notes:

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

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

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

Table 77: MODBUS Functions Supported by Nexto CPUs

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

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

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

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

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

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

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

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

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

Diagnostic Variable: The MODBUS request diagnostics configured by symbolic mapping or by direct representation, are stored in variables of type *T_DIAG_MODBUS_RTU_MAPPING_1* for Master devices and *T_DIAG_MODBUS_ETH_CLIENT_1* for Client devices and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 56.

T_DIAG_MODBUS_ETH_MAPPING_1.*	Size	Description
Communication Status Bits:		
byStatus.bCommIdle	BIT	Communication idle (waiting to be executed).
byStatus.bCommExecuting	BIT	Active communication.
byStatus.bCommPostponed	BIT	Communication deferred, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of time-out. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true.
byStatus.bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
byStatus.bCommOk	BIT	Communication terminated previously was held successfully.
byStatus.bCommError	BIT	Communication terminated previously had an error. Check error code.
byStatus.bCommAborted	BIT	Previously terminated communication was interrupted due to connection failure.
Last error code (enabled when bCommError = true):		
eLastErrorCode	MASTER_ERROR_CODE (BYTE)	Informs the possible cause of the last error in the MODBUS mapping. Consult Table 79 for further details.
Last exception code received by master:		
eLastExceptionCode	MODBUS_EXCEPTION (BYTE)	NO_EXCEPTION (0) FUNCTION_NOT_SUPPORTED (1) MAPPING_NOT_FOUND (2) ILLEGAL_VALUE (3) ACCESS_DENIED (128)* MAPPING_DISABLED (129)* IGNORE_FRAME (255)*
Communication statistics:		
wCommCounter	WORD	Communications counter terminated, with or without errors. The user can test when communication has finished testing the variation of this counter. When the value 65535 is reached, the counter returns to zero.
wCommErrorCounter	WORD	Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero.

Table 78: MODBUS Client Relations Diagnostics

Notes:

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

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

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

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

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

Table 79: MODBUS Relations Error Codes

ATTENTION

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

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

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

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

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

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

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

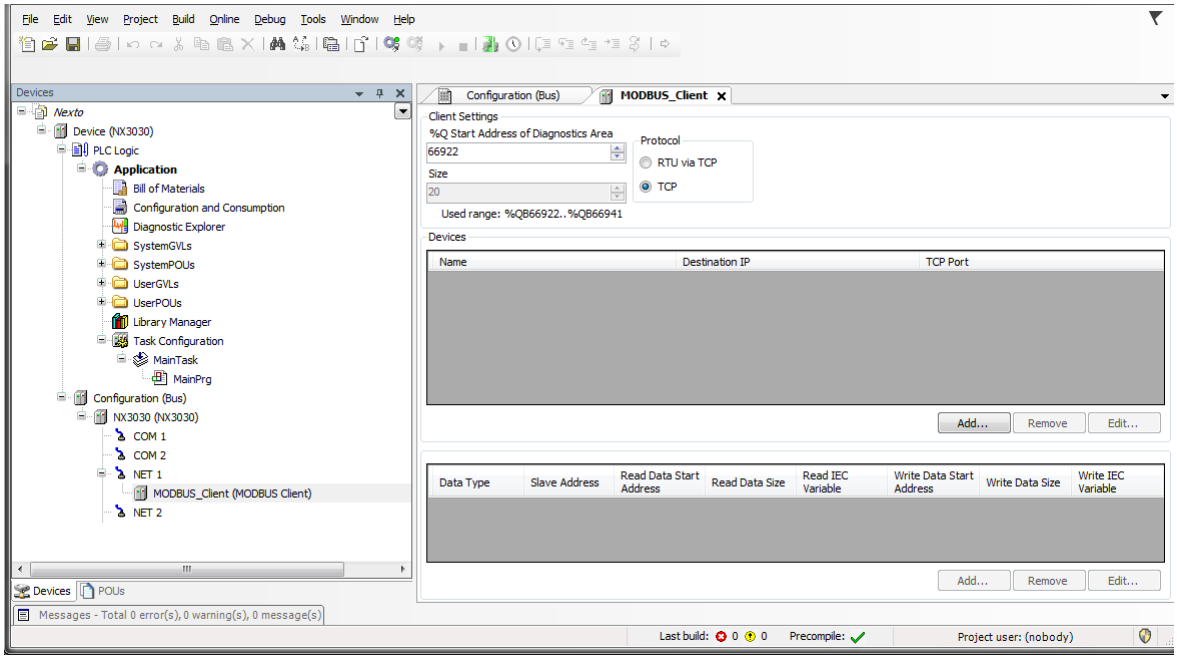


Figure 85: MODBUS Client Setup Screen

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 80: MODBUS Client settings

Notes:

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

Default Value: the default value cannot be defined for the *%Q Start Address of Diagnostics Area* field since the creation of a protocol instance can be made at any moment within the application development. The Mastertool 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 71.

5.7.5.2.2. Device Configuration – Configuration via Direct Representation (%Q)

The configuration of the devices, viewed in figure below, comprises the following parameters:

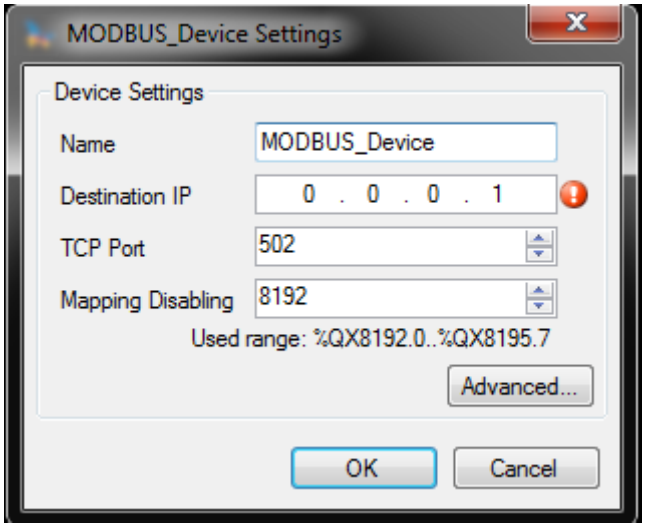


Figure 86: Configuring MODBUS Client

Settings	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 81: Settings of Client Devices

Notes:

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

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

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

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

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

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

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

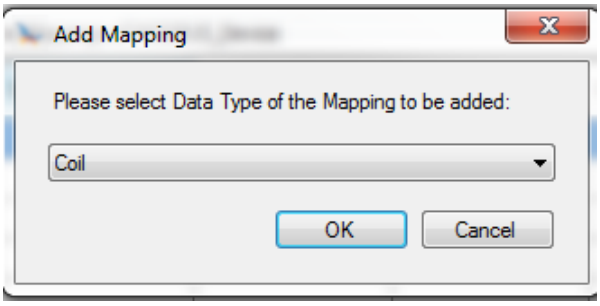


Figure 87: MODBUS Data Type

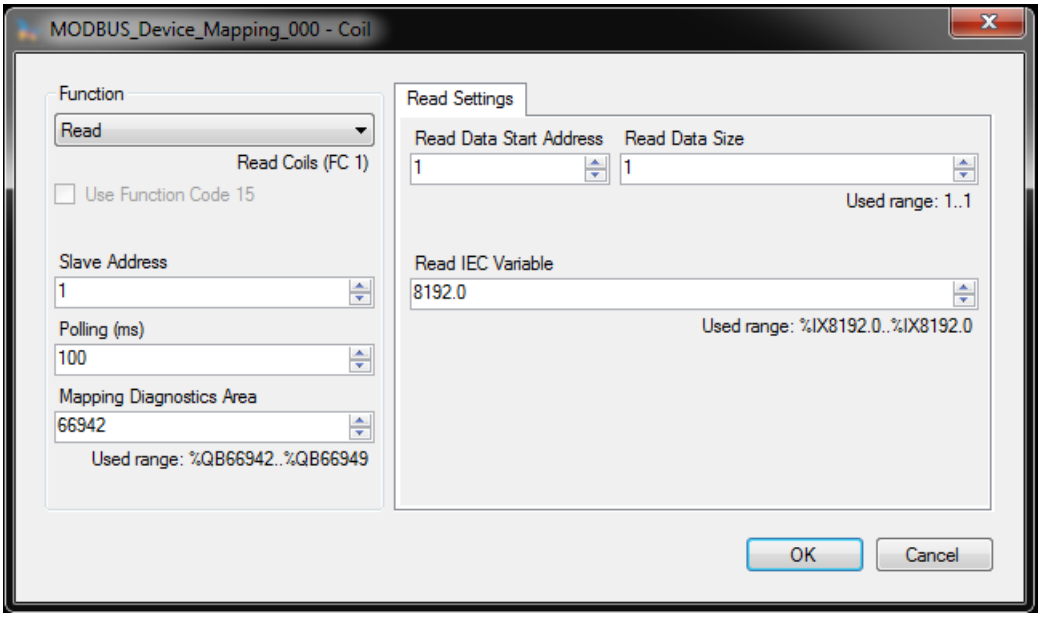


Figure 88: MODBUS Function

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

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

Configuration	Description	Default Value	Options
Mask Write IEC Variables	Starting address of variables for write mask (%Q)	-	0 to 2147483644

Table 82: Device Mapping

Notes:

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

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

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

Mapping Diagnostic Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in the section Memory . The configured MODBUS relations diagnostics are described in Table 56.

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

ATTENTION

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

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

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

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

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

ATTENTION

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

5.7.5.3. MODBUS Client Relation Start in Acyclic Form

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

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

5.7.6. MODBUS Ethernet Server

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

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 User Manual.

5.7.6.1. MODBUS Server Ethernet Protocol Configuration for Symbolic Mapping

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

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

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

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

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

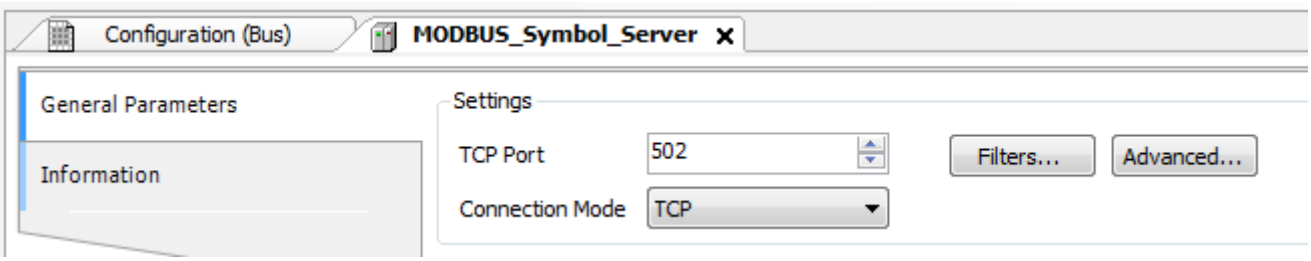


Figure 89: MODBUS Server General Parameters Settings Screen

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

Table 83: MODBUS Server General Parameters

Notes:

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

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

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

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

Table 84: IP Filters

Note:

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

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

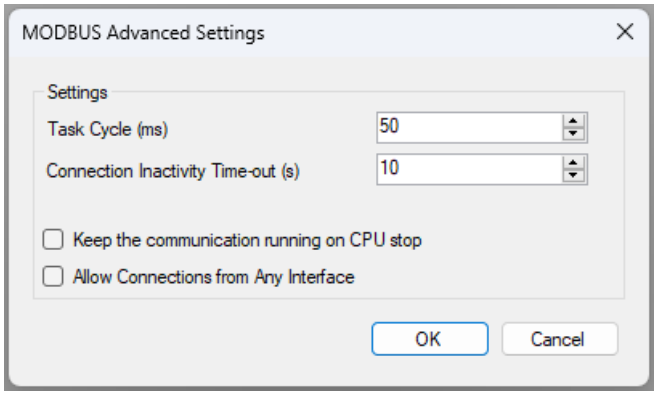


Figure 90: MODBUS Server Advanced Settings Screen

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

Table 85: MODBUS Server Advanced Settings

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.

Keep the communication running on CPU stop: This parameter is valid only for non-redundant PLCs and is not available for redundant PLCs.

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

5.7.6.1.2. MODBUS Server Diagnostics – Configuration via Symbolic Mapping

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

T_DIAG_MODBUS_ETH_SERVER_1.*	Size	Description
Diagnostic Bits:		
tDiag.bRunning	BIT	The server is running.
tDiag.bNotRunning	BIT	The server is not running (see bit bInterruptedBy-Command).
tDiag.bInterruptedByCommand	BIT	The bit bNotRunning was enabled, because the server was interrupted by the user through the command bit.
tDiag.bConfigFailure	BIT	Configuration failure.
tDiag.bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present.
Command bits, automatically initialized:		
tCommand.bStop	BIT	Stop the server.
tCommand.bRestart	BIT	Restart the server.
tCommand.bResetCounter	BIT	Reset diagnostics statistics (counters).
Communication Statistics:		
tStat.wActiveConnections	WORD	Number of established connections between client and server (0 to 64).
tStat.wTimeoutClosedConnections	WORD	Connections counter, between the client and server, interrupted after a period of inactivity - time-out (0 to 65535).
tStat.wClientClosedConnections	WORD	Connections counter interrupted due to customer request (0 to 65535).
tStat.wRXFrames	WORD	Ethernet frames counter received by the server. An Ethernet frame can contain more than one request (0 to 65535).
tStat.wRXRequests	WORD	Requests received by the server counter and answered normally (0 to 65535).
tStat.wTXExceptionResponses	WORD	Requests received by the server counter and answered with exception codes (0 to 65535).
tStat.wRXIllegalRequests	WORD	Illegal requests counter (0 to 65535).

Table 86: MODBUS Server Diagnostics

Note:

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

bModuleFailure: Diagnosis implemented only for symbolic MODBUS.

5.7.6.1.3. Mapping Configuration – Configuration via Symbolic Mapping

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

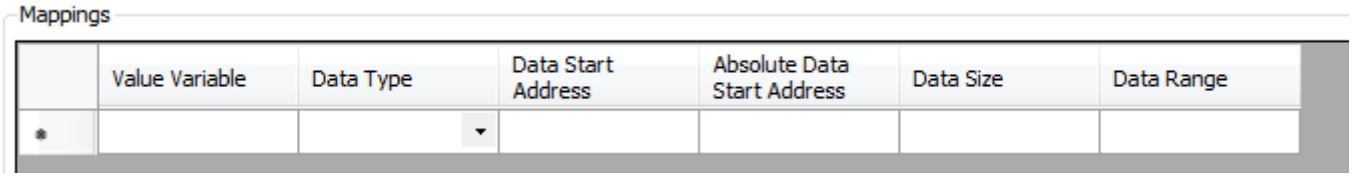


Figure 91: MODBUS Server Data Mappings Screen

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

Table 87: MODBUS Ethernet Mappings Configuration

Notes:

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

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

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

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

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

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

ATTENTION

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

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

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

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

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

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

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

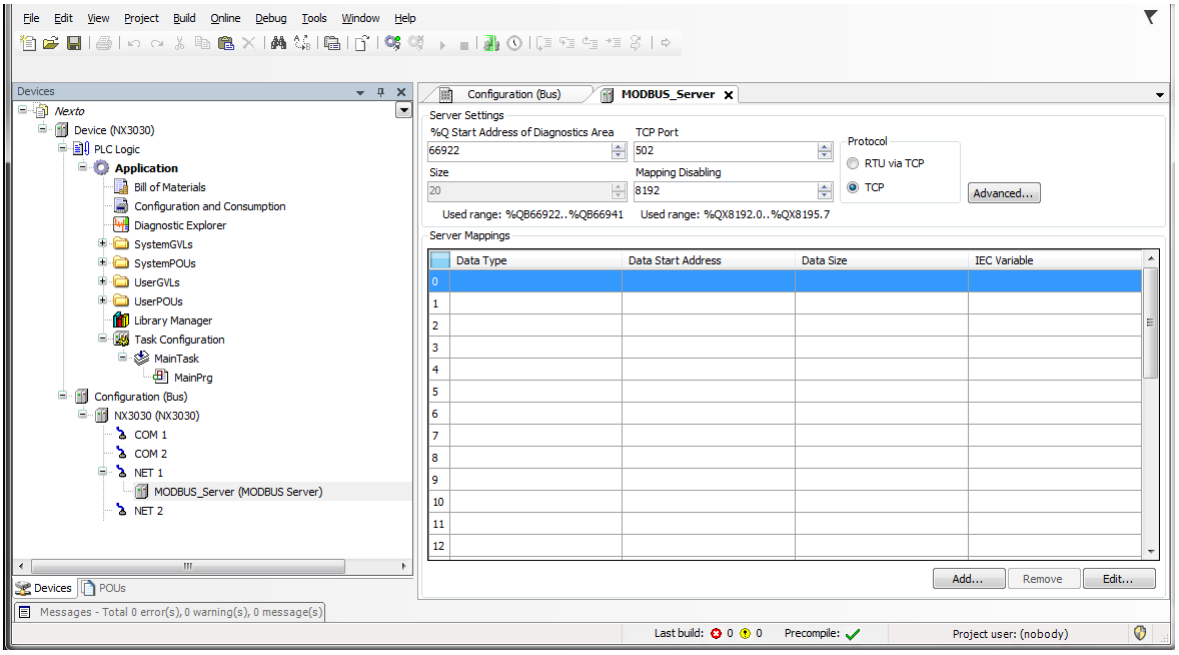


Figure 92: MODBUS Server Setup Screen

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 88: Settings to control relations and diagnostics

Notes:

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

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

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

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

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

The diagnostics and MODBUS commands are described in Table 86.

5.7.6.2.2. *Mapping Configuration – Configuration via Direct Representation (%Q)*

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

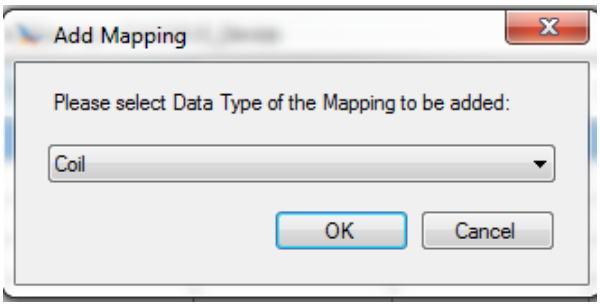


Figure 93: MODBUS Data Type

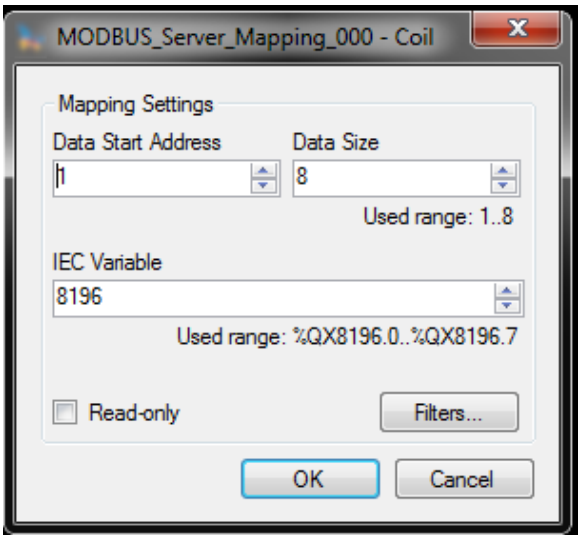


Figure 94: MODBUS Server Function

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

Table 89: Server Mappings

Notes:

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

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

ATTENTION

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

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

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

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

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

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

Table 90: IP Filters

Note:

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

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

- Read Coils (FC 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (FC 3): 125
- Read Input Registers (FC 4): 125
- Write Single Coil (FC 5): 1
- Write Single Holding register (FC 6): 1
- Force Multiple Coils (FC 15): 1968

- Write Holding Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Holding Registers (FC 23):
  - Read: 121
  - Write: 121

ATTENTION

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

5.7.7. OPC DA Server

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

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

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

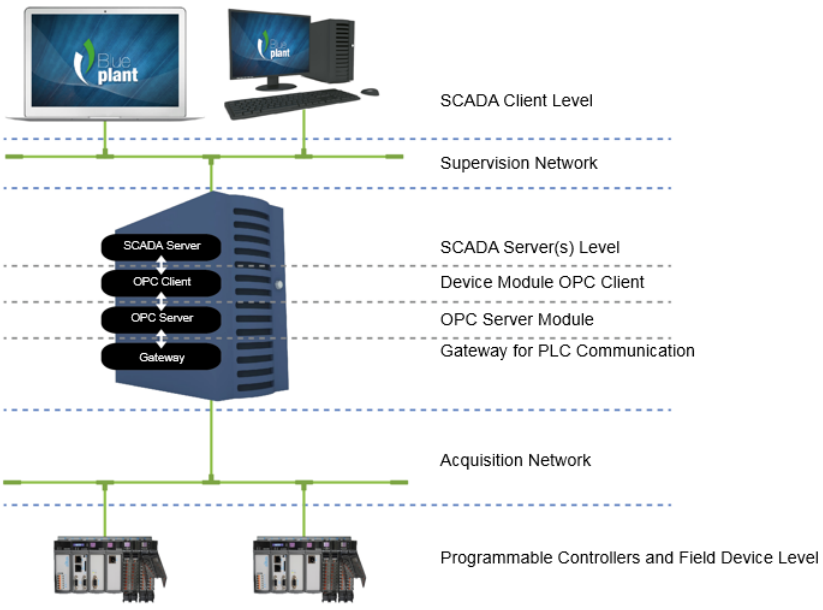


Figure 95: OPC DA Architecture

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

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

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

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

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

5.7.7.1. Creating a Project for OPC DA Communication

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

ATTENTION

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

ATTENTION

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

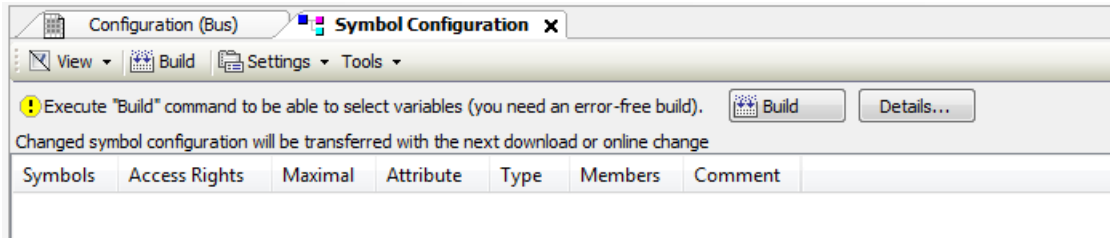


Figure 96: Symbol Configuration Object

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

Field	Description
Symbols	Variable identifier that will be provided to the OPC DA Server.
Access Rights	Indicates what the possible access right level are in the declared symbol. When not utilized, this column remains empty, and the access right level is maximum. Otherwise the access right level can be modified by clicking over this field. The possible options are: Read only Write only Read and Write
Maximal	Indicates the maximum access right level that is possible to assign to the variable. The symbols hold the same meanings from the ones in Access Rights. It's not possible to change it and it's indicated by the presence or not of the *attribute 'symbol'*
Attribute	Indicates if *attribute 'symbol'* is being used when the variable is declared. When not used, this column remains empty. For the cases in which the attribute is used, the behavior is the following: attribute 'symbol' := 'read' the column shows attribute 'symbol' := 'write' the column shows attribute 'symbol' := 'readwrite' the column shows
Type	Data type of the declared variable.
Members	When the data type is a Struct, a button is enabled in this column. Clicking on the button will allow the selection of which elements of that struct will be provided to the OPC DA Server.

Field	Description
Comment	Variable comment, inserted on the POU or GVL where the variable was declared. To show up as a variable comment here, the comment must be entered one line before the variable on the editor, while in text mode, or in the comment column when in tabular mode.

Table 92: 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 96 disappear. After this, all available variables in the project, whether they are declared on POUs, GVLs or diagnostics, will be shown here and can be selected. The selected variables will be available on the OPC DA Server to be accessed by the Clients.

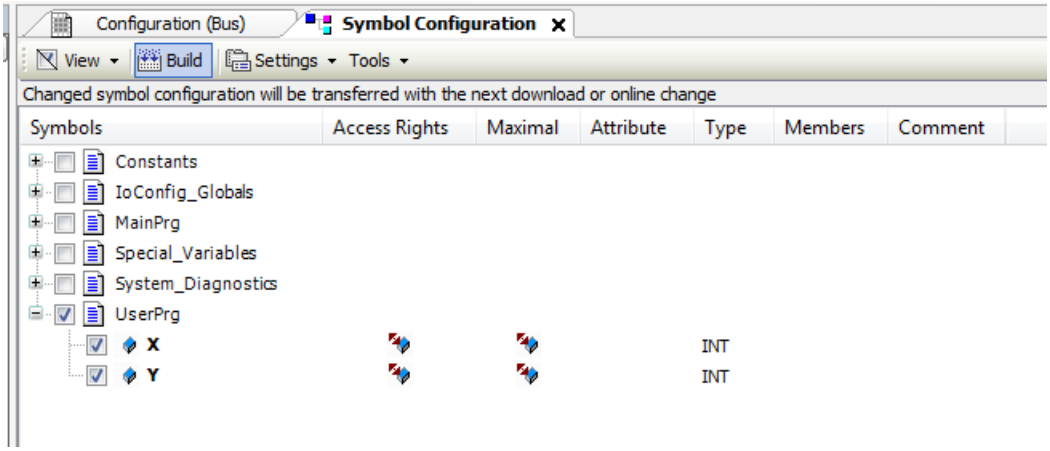


Figure 97: Selecting Variables on the Symbol Configuration

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

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

The valid syntaxes to use the attribute are:

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

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

```
{attribute 'symbol' := 'readwrite'}
PROGRAM UserPrg
VAR
```

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

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

ATTENTION

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

ATTENTION

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

5.7.7.2. Configuring a PLC on the OPC DA Server

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

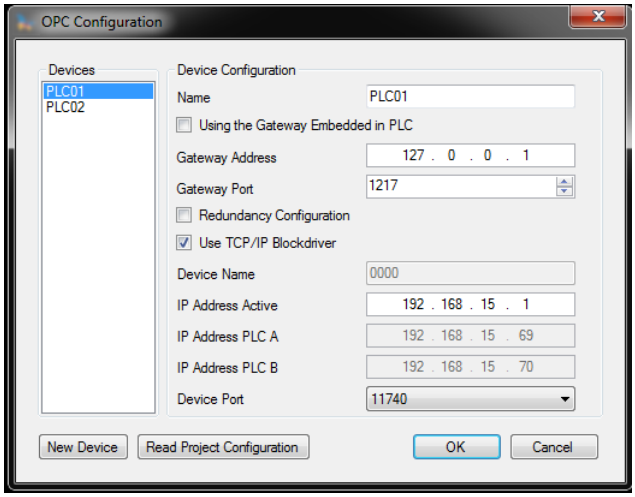


Figure 98: OPC DA Server Settings

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

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

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

The other option is to use the *IP Address* of the Ethernet Interfaces. The same address set on the configuration screens must be put in this field. Furthermore, when this method is used, the port number must be set to 11740 (the 11739 port is intended for simulation mode use only). The confirmation will save the OPC DA Server configurations.

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

Device Configuration	Description	Default Setting	Options
IP Address PLC B	IP address of the PLC B. Enabled only when the configuration is redundant. It is the secondary PLC address to which the server will communicate if a failure occurs.	192.168.15.70	0.0.0.0 to 255.255.255.255
Device Port	TCP Port. Enabled only when the checkbox *Use TCP/IP Blockdriver* is selected.	11740	11740 or 11739

Table 93: Configuration Parameter of each PLC for the OPC DA Server

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

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

ATTENTION

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

ATTENTION

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

5.7.7.2.1. Importing a Project Configuration

Using the button *Read Project Configuration*, as shown in Figure 98, you can assign the configuration of the open project to the PLC configuration that is being edited. For this option to work correctly, there must be an open project and an *Active Path* should be set as described in *Communication Settings*, present in Mastertool's user manual. If any of these conditions is not met an error message will be displayed and no data will be modified.

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

5.7.7.3. Configuration with the PLC on the OPC DA Server with Connection Redundancy

It's possible to configure the OPC DA Server for it to operate with connection redundancy. This way, the OPC DA 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 standby state, ready to take control of the process if some kind of failure occurs.

The project configuration in these cases is similar to that described in Creating a Project for OPC DA Communication. However, when a project is created with Half-Cluster Redundancy and communication with the supervisory system is carried

out through the OPC DA Server, it is necessary to add code in the *NonSkippedPrg* to enable communication with OPC DA connection redundancy, as shown in the example below.

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 DA 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 *attribute 'symbol' := 'read'* to allow the OPC DA 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 DA Server, it's necessary to enable the *Redundancy Configuration* option in the configuration screen as shown in Figure 98. When this option is selected, the option *Use TCP/IP Blockdriver* will always be used. In addition, the *IP Address PLC A* and *IP Address PLC B* fields will be enabled as described in Table 93. 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 DA 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 DA Server configuration. Refer to the SCADA system documentations to see which configurations are needed for the operation of the redundant architecture.

5.7.7.4. OPC DA Communication Status and Quality Variables

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

State	Value	Description
STATE_TERMINATE	-1	If the communication between the OPC DA Server and the OPC Client is terminated, this value will be returned. When there's more than one OPC Client simultaneously connected, this return will occur on the disconnection of the latter connected one. Besides the fact that this state is in the variable, it's value can't be visualized because it only changes when there's no longer a connection with the client.
STATE_PLC_NOT_CONNECTED	0	The PLC configured in the OPC DA Server is not connected. It can happen if the configuration is incorrect (wrong PLC and/or Gateway IP Address) or the PLC is unavailable in that moment.
STATE_PLC_CONNECTED	1	The PLC configured in the OPC DA Server is connected. This is a transitory state during the connection.
STATE_NO_SYMBOLS	2	There are no symbols (variables) available in the PLC configured in the OPC DA Server. It can happen when there are no symbols or there isn't a project loaded on the PLC.
STATE_SYMBOLS_LOADED	3	Finished the process of reading the symbols (variables) from the PLC configured in the OPC DA Server. This is a transitory state during the connection.
STATE_RUNNING	4	After the reading of the symbols (variables) the OPC DA Server is running the periodic update of the values of the available symbols in each configured PLC.
STATE_DISCONNECT	5	There has been a disconnection with the PLC configured in the OPC DA Server.
STATE_NO_CONFIGURATION	6	When the OPC configuration (stored in an OPCServer.ini file) has a wrong syntax, the variable value will be this. Generally, this behavior is not observed for Mastertool maintains this configuration valid.

Table 94: Description of the Communication states between OPC DA Server and the PLC

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

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

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

QQ	Bits values	Definition	Description
3	11SSSSLL	Good	The quality is good and the value read can be used.

Table 95: Description of the OPC Quality value

Table 95 presents the possible quality values. The OPC DA Server only returns *Good* and *Bad* Quality values. A OPC Client can maintain the quality as *Uncertain* due to failures in which it can't establish a connection to the Server. In case of monitoring of the 8 quality bits directly from the OPC DA Server, the *Substatus* and *Limit* fields shall be null and the *Good* Quality will be presented as the value 192 and the *Bad* Quality will be value 0.

5.7.7.5. Limits of Communication with OPC DA Server

The table below presents the OPC DA Server configuration limits.

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

Table 96: OPC DA Server Communication Limits

Note:

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

ATTENTION

The Maximum number of simultaneous connections of an OPC DA Server in a single PLC is shared with connections made with Mastertool. I.e. the sum of connections of OPC DA Server and Mastertool should not exceed the maximum quantity defined in Table 96.

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

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

5.7.7.6. Accessing Data Through an OPC DA Client

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

ATTENTION

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

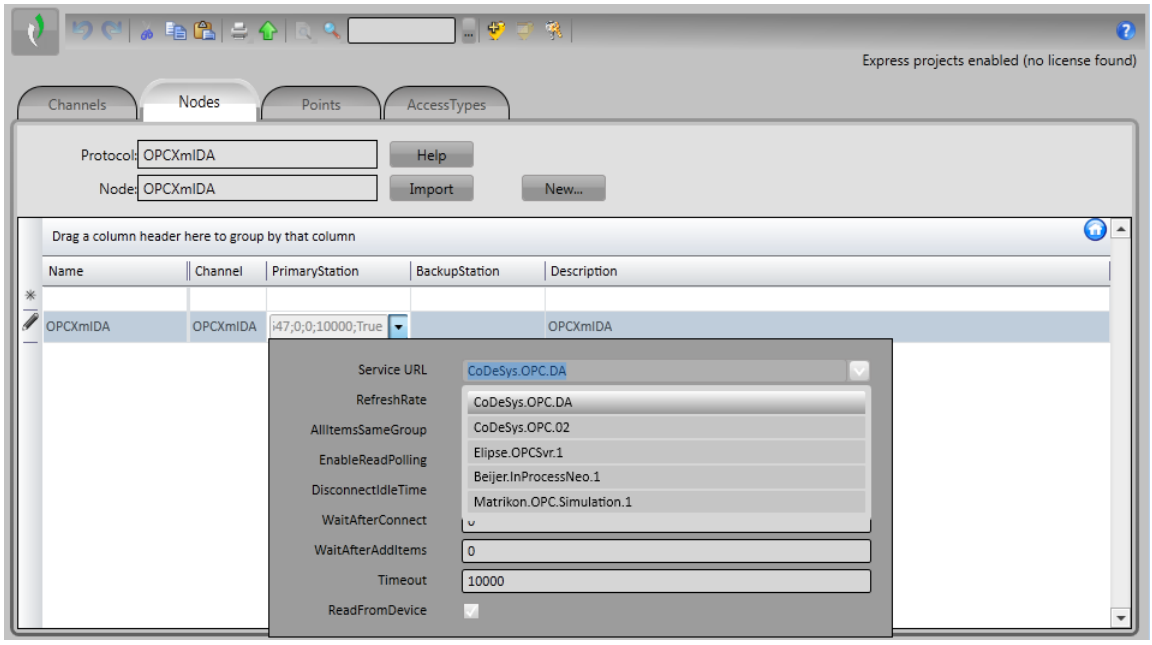


Figure 99: Selecting the OPC DA Server in the Client Configuration

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

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

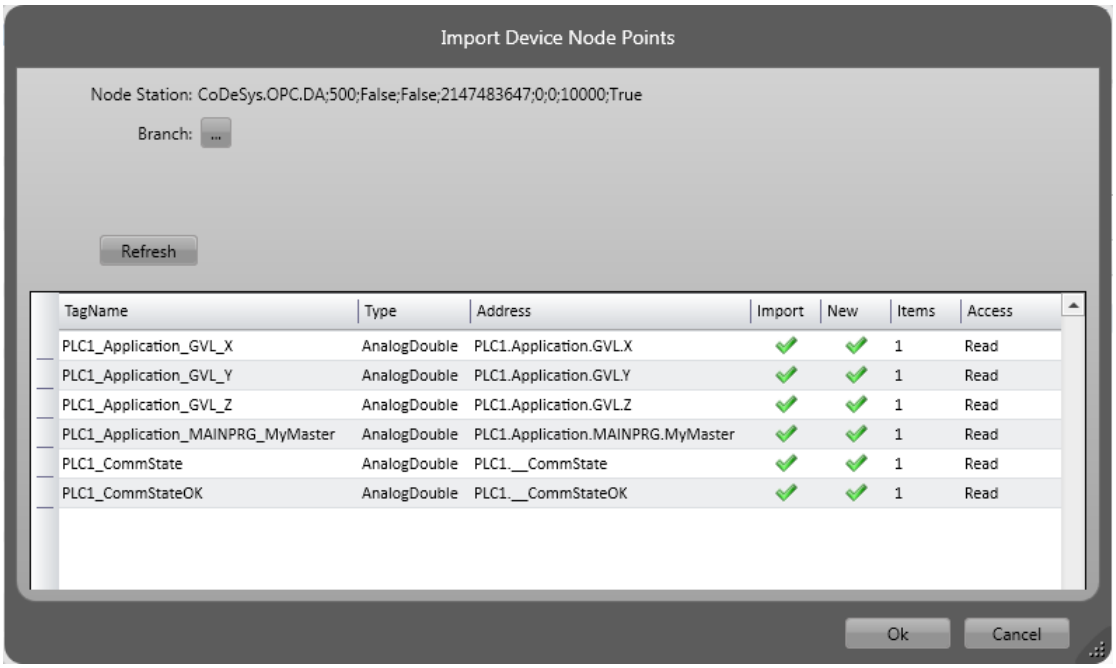


Figure 100: Symbols list consulted by the OPC Client

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

ATTENTION

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

5.7.8. OPC UA Server

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

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

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

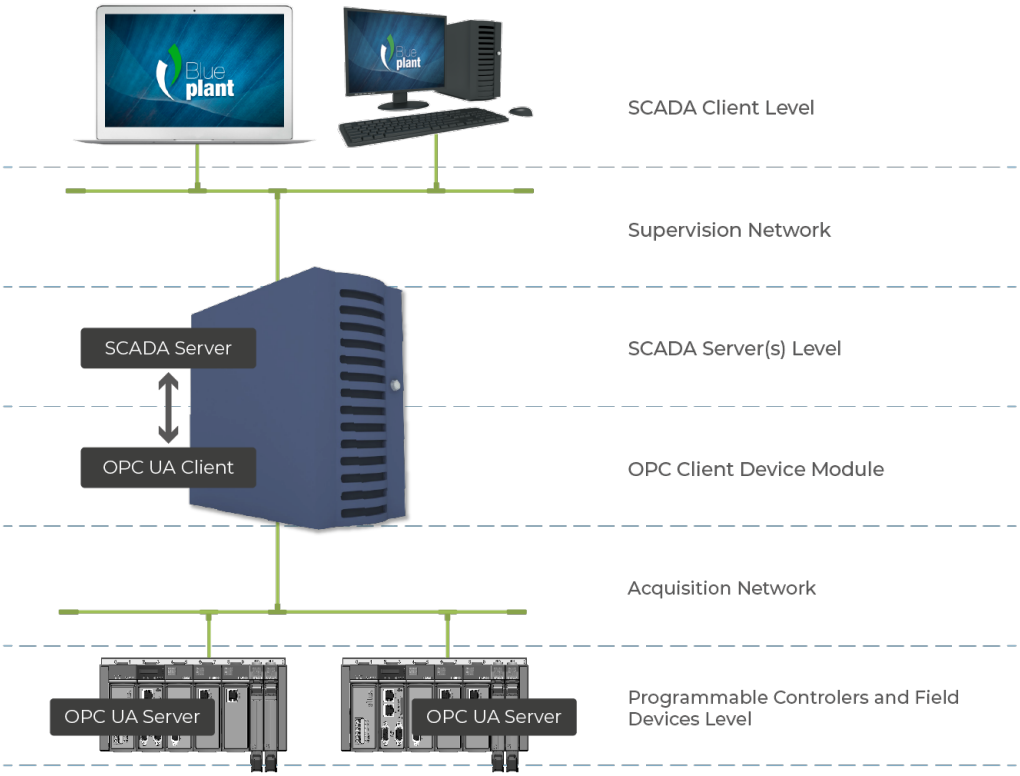


Figure 101: OPC UA Architecture

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

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

Table 97: Roles Description on an OPC UA Server Architecture

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

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

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

For detailed information about the configuration and operation of the OPC UA protocol, refer to OPC UA Server for Altus Controllers User Manual - MU214609.

5.7.9. EtherCAT Master

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

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

any order as the data sequence is independent of the physical order. It can perform Broadcast, Multicast and between slaves communications.

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

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

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

5.7.9.1. Installing and inserting EtherCAT Devices

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

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

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

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

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

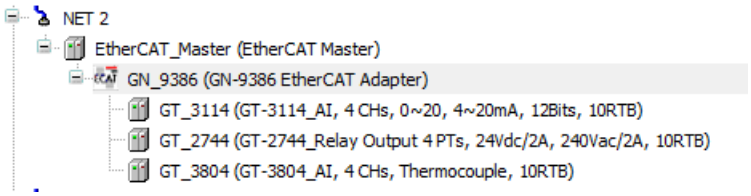


Figure 102: EtherCAT Configuration Example

ATTENTION

- Only one EtherCAT Master instance per project is allowed.
- Only available on the NET connectors of the PLC.
- It cannot be used when the NETs are set as redundant.
- It cannot be used when Project has cluster redundancy.
- Other drivers cannot be instanced in the same NET port as the EtherCAT Master.

5.7.9.1.1. EtherCAT - Scan Devices

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

It's noteworthy that, when the *Scan For Devices* command is selected, a connection with the PLC will be automatically established before the search begins and terminated when the search ends. So, for the first execution of this command, the Gateway connection must be configured and a program with the EtherCAT Master configured must be loaded into the PLC.

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

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

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

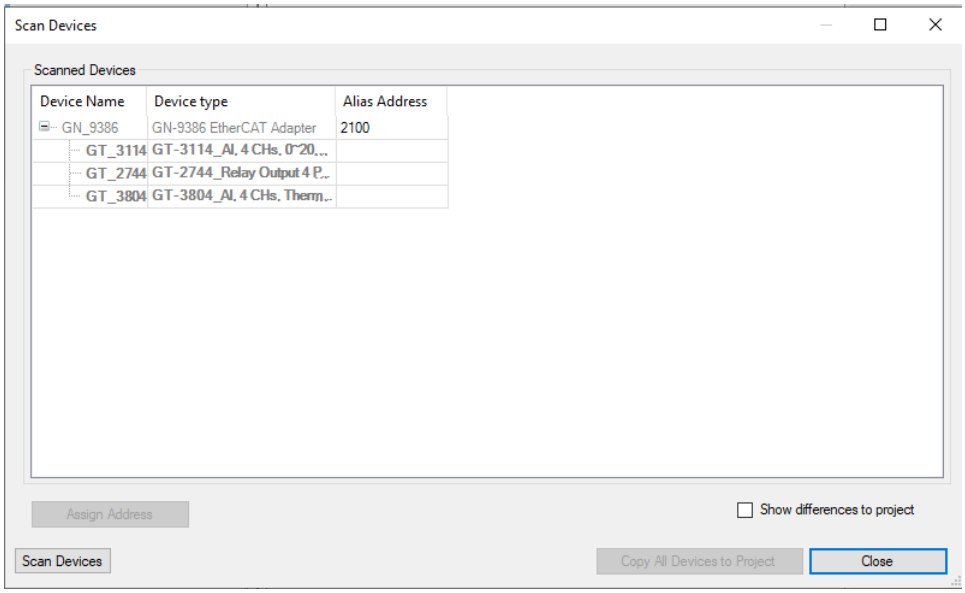


Figure 103: EtherCAT Devices Search Dialog

5.7.9.2. EtherCAT Master Settings

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

5.7.9.2.1. EtherCAT Master - General

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

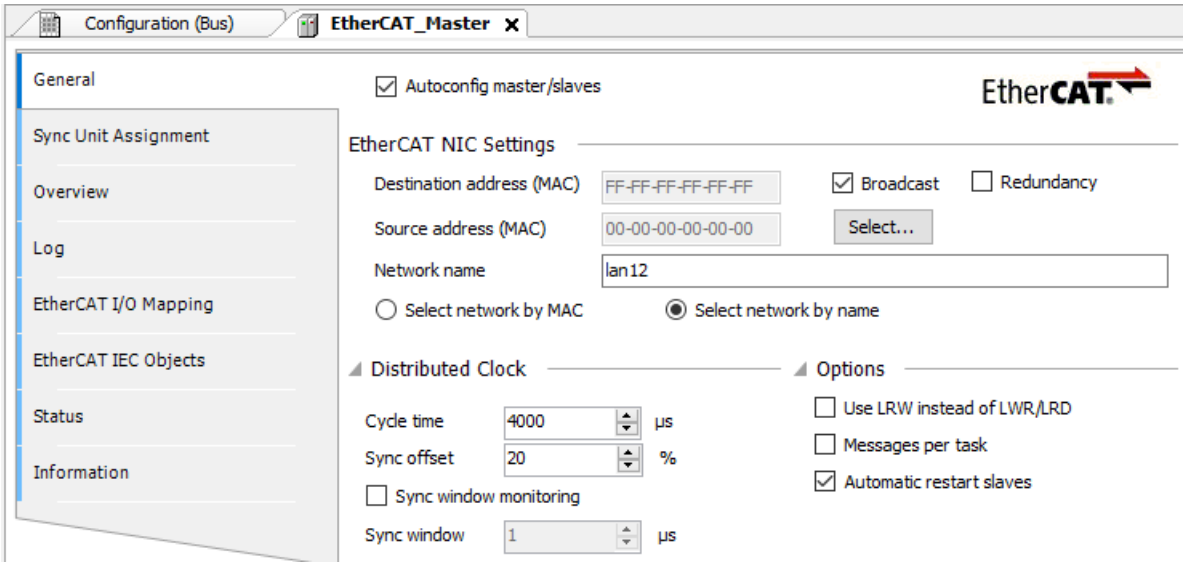


Figure 104: EtherCAT Master Configuration Dialog

Device Configuration	Description	Factory Default	Possible Values
Autoconfig master/slaves	Enable the Master and Slave automatic configuration.	Marked	Marked Unmarked
EtherCAT NIC Settings			
Network name	Configures the network name. It should not be changed.	-	-
Select network by name	The network is identified by its name, and the project is device-independent. It should not be changed.	Marked	-
Redundancy	Enables redundancy in a ring topology. The EtherCAT network remains operational even in the event of a cable failure. The redundancy parameters are configured in the Redundancy EtherCAT NIC Settings section.	Unmarked	Marked Unmarked
Distributed Clock			
Cycle time [μs]	Sets the time period in which a new data telegram must be send to the bus.	4000	2000 to 1000000
Sync Offset [%]	Adjust the offset, from the PLC cycle, of the EtherCAT Slave synchronization interrupt.	20	-50 to 50
Sync window monitoring	If enabled, this option allows monitoring the Slave synchronization.	Unmarked	Marked Unmarked
Sync window [μs]	Time for the Sync Window Monitoring.	1	1 to 32768
Options			
Use LRW instead of LWR/LRD	Enabling of the combined read and write commands.	Unmarked	Marked Unmarked
Messages per task	If enabled, the read and write commands that are dealing with input and output messages can be done in different tasks.	Unmarked	Marked Unmarked
Automatic restart slaves	Restart the devices when the communication is aborted.	Marked	Marked Unmarked
Mailbox Gateway			
Enable	Enables communication from an external device configuration tool, via the mailbox gateway, to the EtherCAT devices.	Unmarked	Marked Unmarked

Table 98: EtherCAT Master Configuration

Notes:

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

ATTENTION

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

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

Sync Offset: This value allows the adjustment of the offset of the EtherCAT Slave synchronization interrupt to the PLC cycle. Normally, the PLC task cycle begins 20% later than the Slaves synchronization interruption. This means that the PLC

task can be delayed by 80% of the cycle time and no message will be lost.

Sync Window: If the synchronization of all Slaves are inside this time window, the EtherCAT Master *bDistributedClockInSync* diagnostic will be set to TRUE, otherwise it will be set to FALSE. When Distributed Clock is used, it's highly recommended to use a dedicated task with high priority as the *Bus cycle task* of the EtherCAT Master. To do this, it's necessary to use Project Profiles that allows the creation of new tasks, then create a cyclic task with priority 0 (real time task) and link it to the master *Bus cycle task* on the EtherCAT Master - I/O Mapping tab of the EtherCAT Master. The user can also change the value of the *wDCInSyncWindow* variable, configuring the maximum jitter allowed on the synchronization between master and slaves.

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

Automatic Restart Slaves: By enabling this option, the Master will restart the Slaves as soon as the communication is aborted.

5.7.9.2.2. EtherCAT Master - Sync Unit Assignment

This tab of the EtherCAT Master configuration editor shows all slaves that are entered below a specific master with an assignment to the sync units.

With EtherCAT sync units, multiple slaves are configured into groups and subdivided into smaller units. For each group, the job counter can be monitored for better and more accurate error detection. As soon as a slave is missing from a group of synchronization units, the other slaves in the group are also shown as missing. Detection occurs immediately on the next bus cycle because the job counter is checked continuously. With device diagnostics, the missing group can be remedied as quickly as possible.

Unaffected groups remain operable without any interference.

5.7.9.2.3. EtherCAT Master - Overview

This tab of the EtherCAT Master configuration editor provides an overview of the states of all slaves, which are entered below this master and have an address. Modules are not displayed.

5.7.9.2.4. EtherCAT Master - I/O Mapping

This EtherCAT Master configuration editor tab offers the possibility to change the task that will be used for bus updates.

5.7.9.2.5. EtherCAT Master - IEC Objects

This tab of the EtherCAT Master configuration editor lists *objects* that allow access to the device from the IEC application. In online mode, this is used for monitoring.

5.7.9.2.6. EtherCAT Master - Status / Information Tabs

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

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

5.7.9.3. EtherCAT Slave Configuration

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

5.7.9.3.1. EtherCAT Slave - General

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

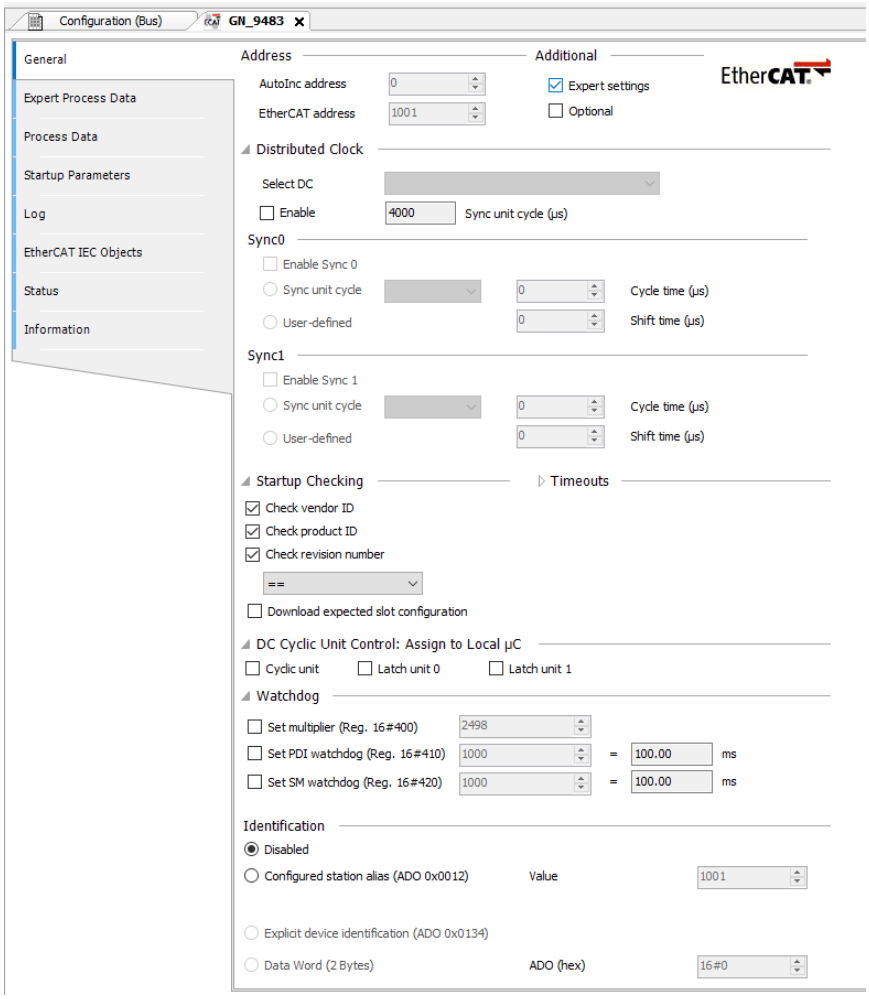


Figure 105: EtherCAT Slave Configuration Dialog

Device Configuration	Description	Default Value	Options
AutoInc Address	Auto incremental Address (16-bit) defined by the Slave position in the network.	-	-65535 to 0
EtherCAT Address	Slave final address, assign by the Master during startup. This address is independent from the position in the network.	-	1 to 65535
Expert settings	Enable the Slave advanced Settings options.	Unmarked	Marked Unmarked
Optional	Declare the Slave as Optional.	Unmarked	Marked Unmarked
Select DC	Show all Distributed Clock configurations provided by the device description file.	-	-
Enable Distributed Clock	Enable the Distributed Clock configuration options.	Unmarked	Marked Unmarked
Sync Unit Cycle [μs]	Show the Cycle Time set in Master.	100000	2000 to 1000000
Enable (Sync 0)	Enable the Sync 0 synchronization unit configurations.	Unmarked	Marked Unmarked
Sync Unit Cycle (Sync 0)	By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle.	Unmarked	Marked Unmarked

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

Table 99: EtherCAT Slave Configurations

Notes:

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

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

Enable Distributed Clock: If the *Distributed Clock* functionality is enabled, the data exchange cycle time, displayed in the *Sync Unit Cycle (μs)* field will be determined by the Master *Cycle Time*. Thus the master clock can synchronize the data exchange within the network. The settings for handling the synchronization unit(s) depend on the Slave.

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

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

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

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

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

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

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

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

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

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

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

5.7.9.3.2. EtherCAT Slave - Process Data

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

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

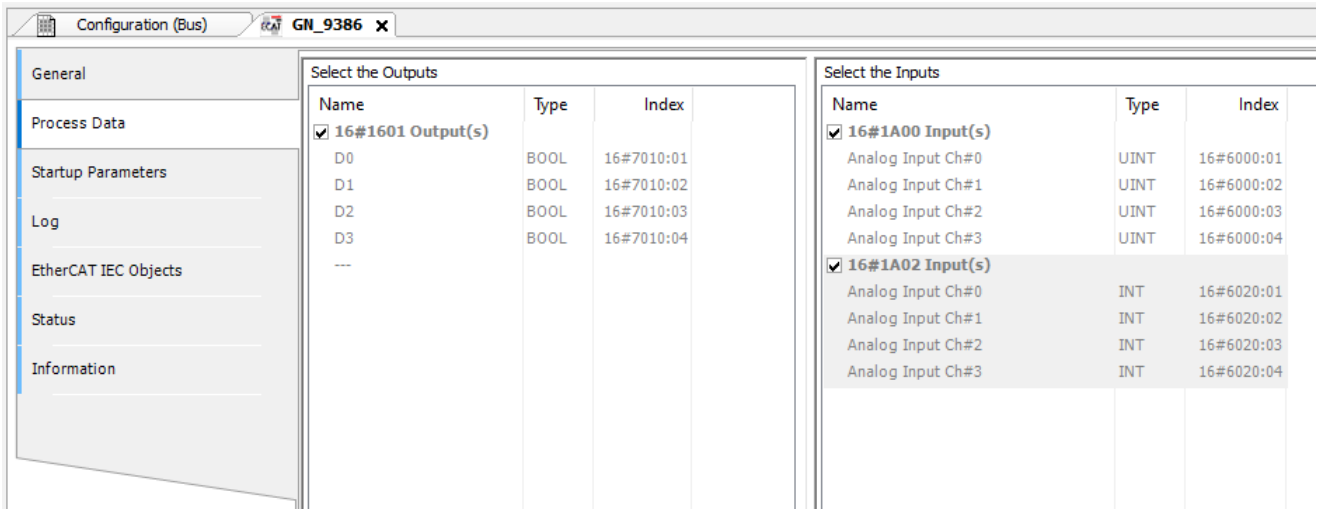


Figure 106: Process Data Dialog

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.

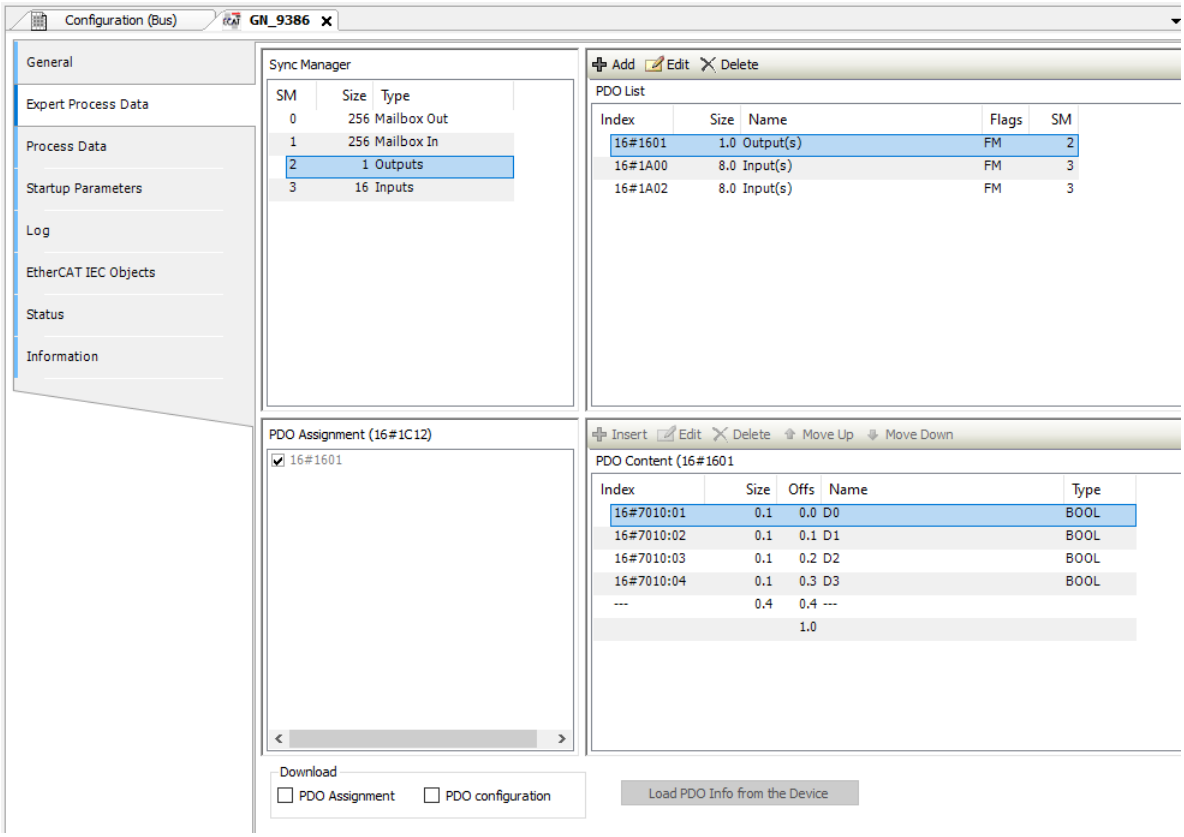


Figure 107: Expert Process Data Dialog

This dialog is divided in four sections and two options:

- *Sync Manager*: List of *Sync Manager* with data size and type of PDOs.
- *PDO Assignment*: List of PDOs assigned to the selected *Sync Manager*. The checkbox activates the PDO and I/O channels are created. It is similar to the simple PDO configuration windows. Here only PDOs can be enabled or disabled.
- *PDO List*: List of all PDOs defined in the device description file. Single PDOs can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Content*: Displays the content of the PDO selected in the section above. Entries can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Assignment*: If activated a CoE write command will be added to index 0x1CXX to write the PDO configuration 0x16XX or 0x1A00.
- *PDO Configuration*: If activated several CoE write commands will be added to write the PDO mapping to the slave.

ATTENTION

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

5.7.9.3.3. EtherCAT Slave - Edit PDO List

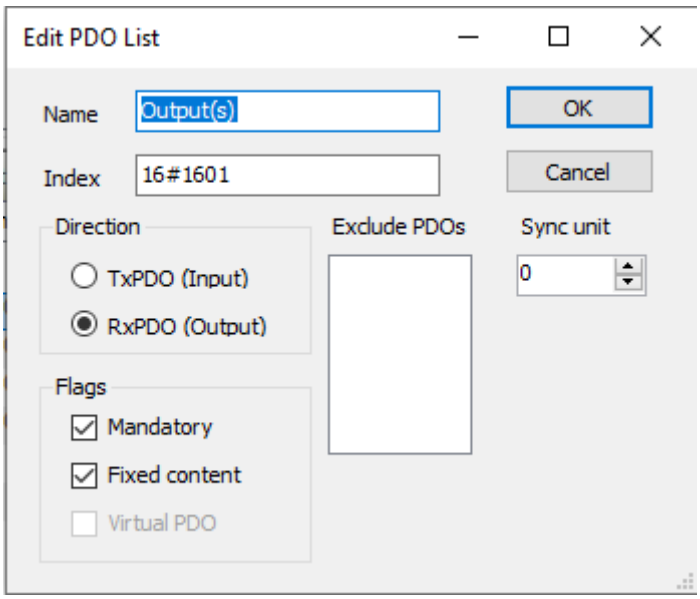


Figure 108: Edit PDO List Dialog

This dialog is opened through the context menu from the PDO List area, presented in Figure 107. Below are some explanations on the configuration options presented in this dialog.

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

5.7.9.3.4. EtherCAT Slave - Startup Parameters

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

5.7.9.3.5. EtherCAT Slave - Module 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*.

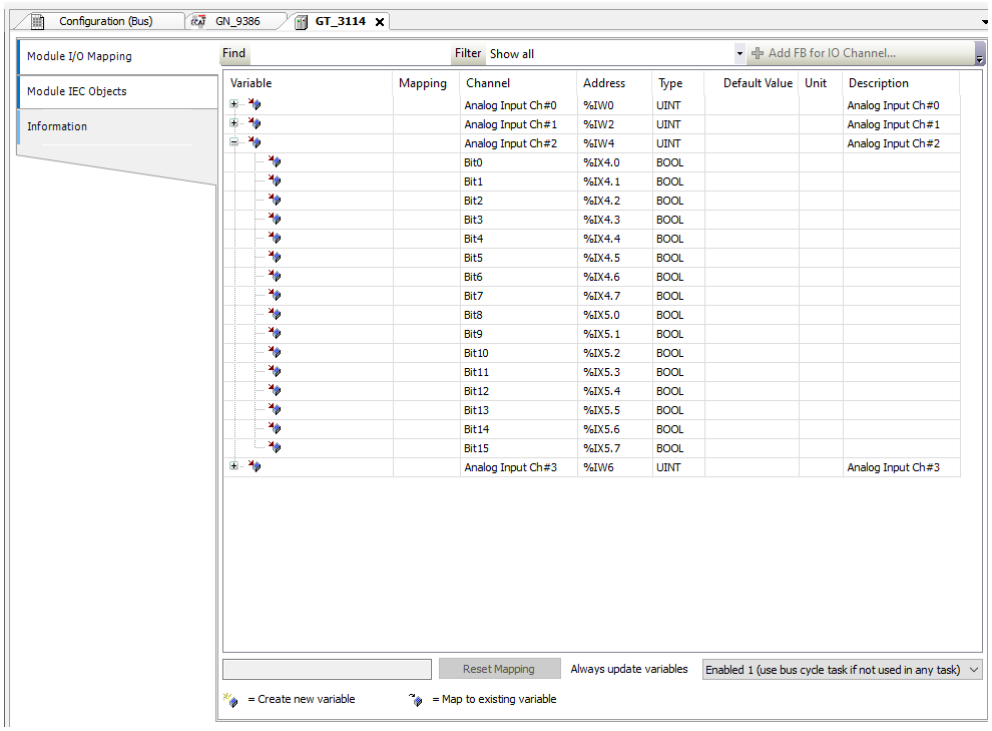


Figure 109: Slave I/O Mapping Dialog

5.7.9.3.6. EtherCAT Slave - Status and Information

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

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

5.7.10. EtherNet/IP

The EtherNet/IP is a master-slave architecture protocol, consisting of an EtherNet/IP Scanner (master) and one or more EtherNet/IP Adapters (slave).

The Ethernet/IP protocol is based on CIP (*Common Industrial Protocol*), which have two primary purposes: The transport of control-oriented data associated with I/O devices and other system-related information to be controlled, such as configuration parameters and diagnostics. The first one is done through implicit messages, while the second one is done through explicit messages.

Their runtime system can act as either Scanner or Adapter. Each CPU's NET interface support only one EtherNet/IP instance and it can't be instanced on an Ethernet expansion module.

An EtherNet/IP Adapter instance supports an unlimited number of modules or Input/Output bytes. In these modules, can be added variables of types: BYTE, BOOL, WORD, DWORD, LWORD, USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL and LREAL.

ATTENTION

EtherNet/IP cannot be used when Ethernet interfaces are configured in NIC Teaming Mode or Switch Mode, or in applications using Half-Cluster redundancy.

ATTENTION

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

5.7.10.1. Ethernet Adapter

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

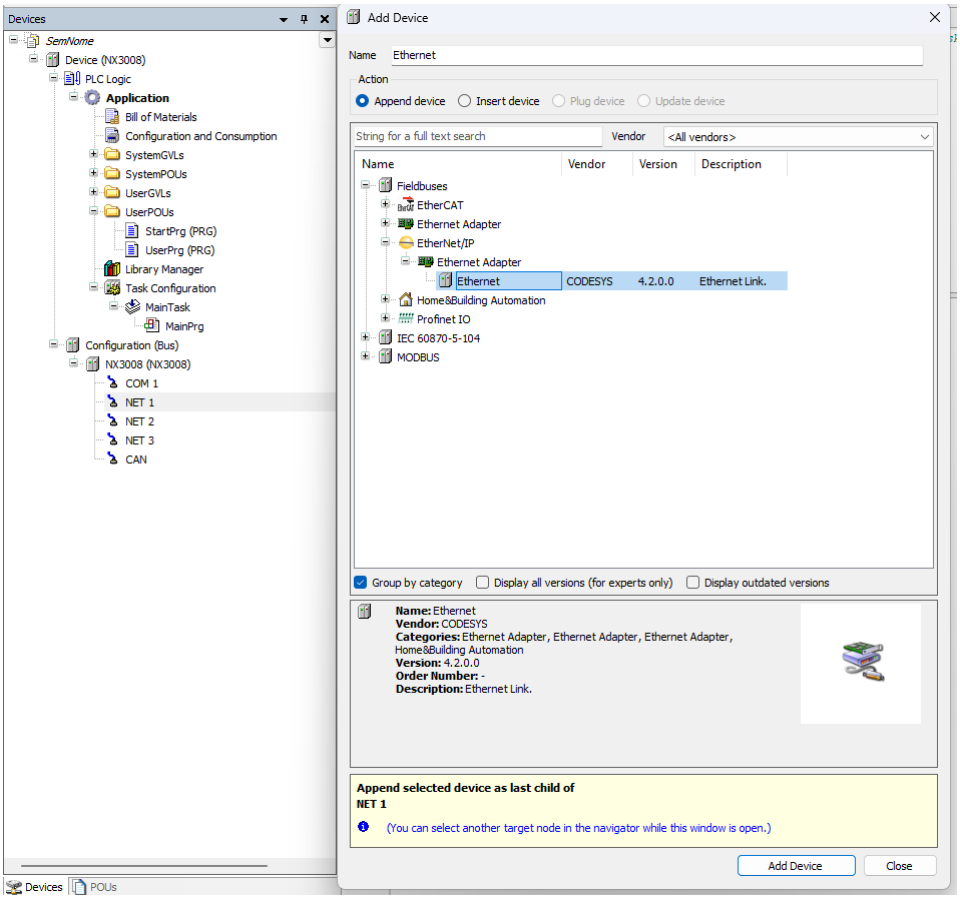


Figure 110: Adding an Ethernet Adapter

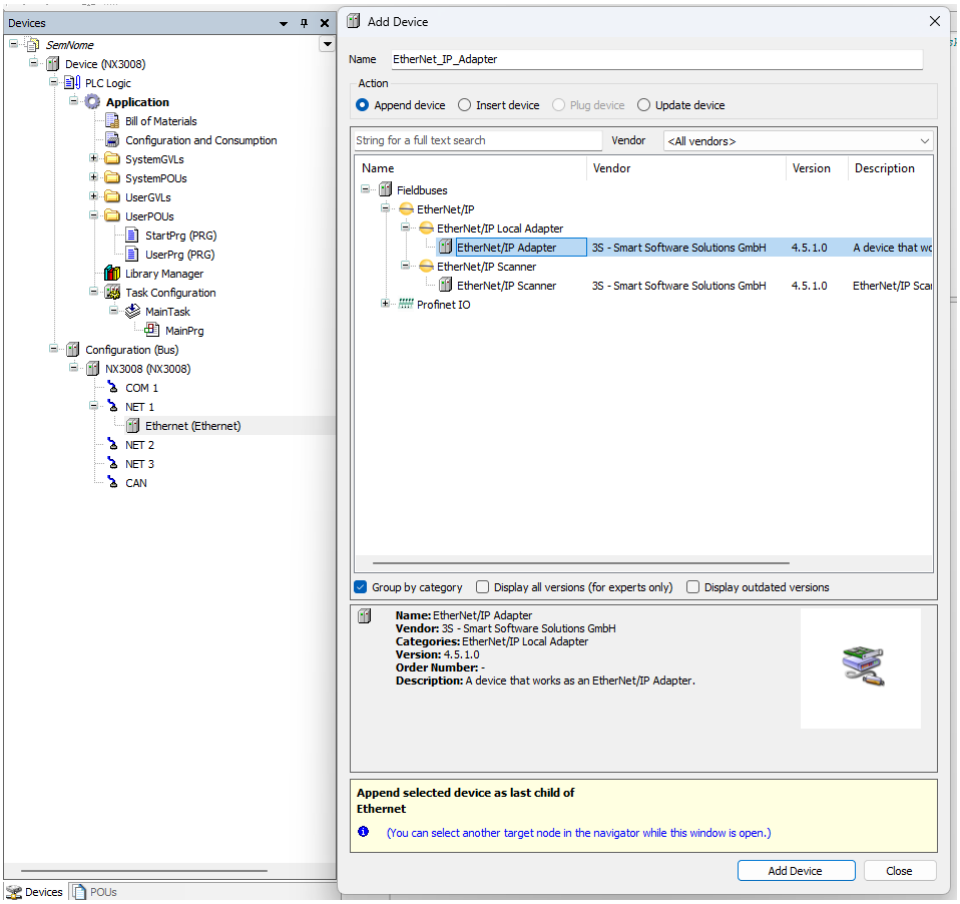


Figure 111: Adding an EtherNet/IP Adapter or Scanner

5.7.10.2. EtherNet/IP Scanner Configuration

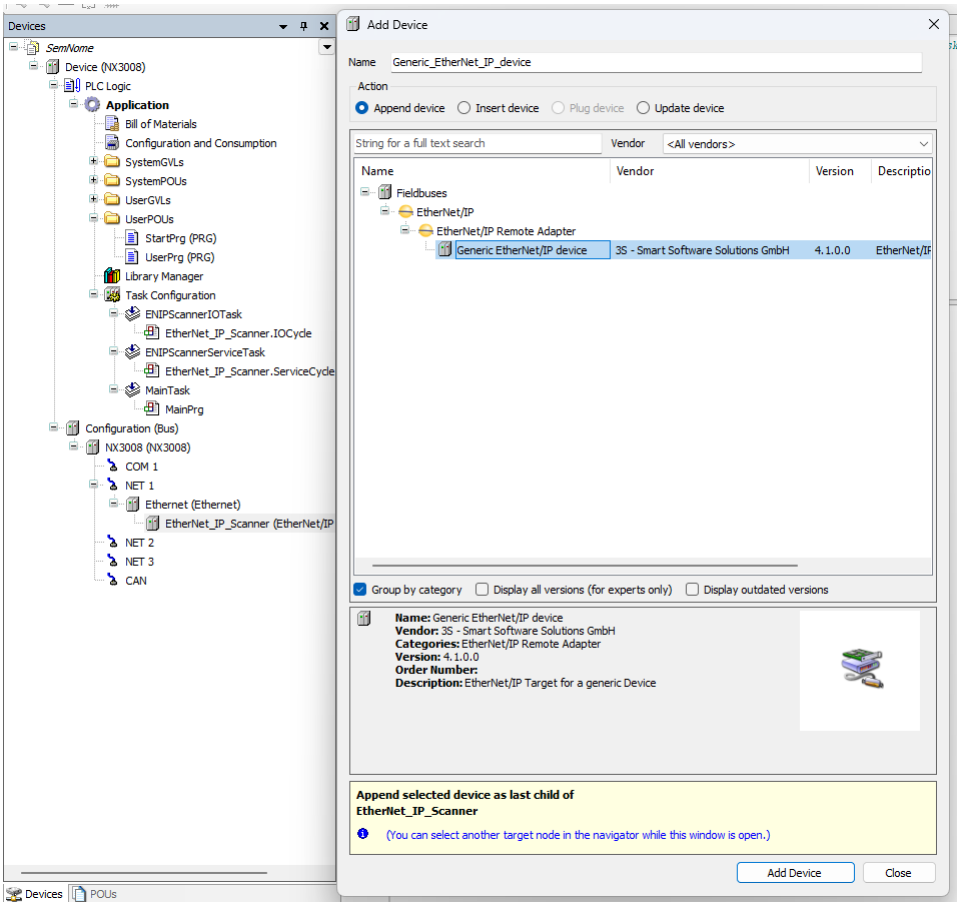


Figure 112: Adding an EtherNet/IP Adapter Under the Scanner

5.7.10.2.1. General

After open the Adapter declared under the Scanner it's possible to configure it as needed. The first Tab is *General*, on it is possible to configure the *IP address* and the *Electronic Keying* parameters. These parameters must be checked or unchecked if the adapter being used is installed on Mastertool. Otherwise, if the Adapter used is of type Generic. The Vendor ID, Device Type, Product Code, Large Revision, and Small Revision fields must be filled in with the correct vendor's information and the boxes checked as much as necessary. Altus, for its part, has its own ID, which is "1454".

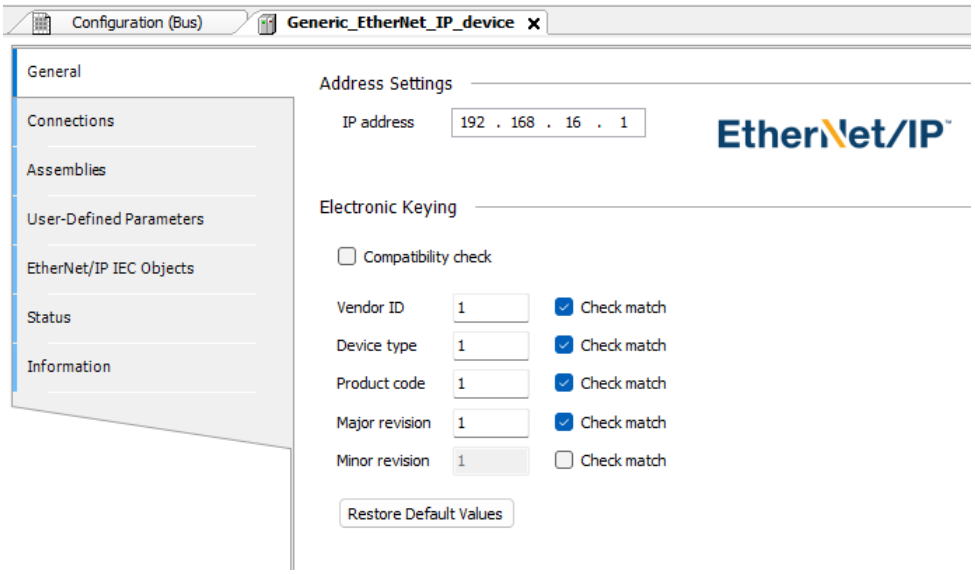


Figure 113: EtherNet/IP General Tab

5.7.10.2.2. *Connections*

The upper area of the *Connections* tab displays a list of all configured connections. When there is an *Exclusive Owner* connection in the EDS file, it is inserted automatically when the Adapter is added. The configuration data for these connections can be changed in the lower part of the view.

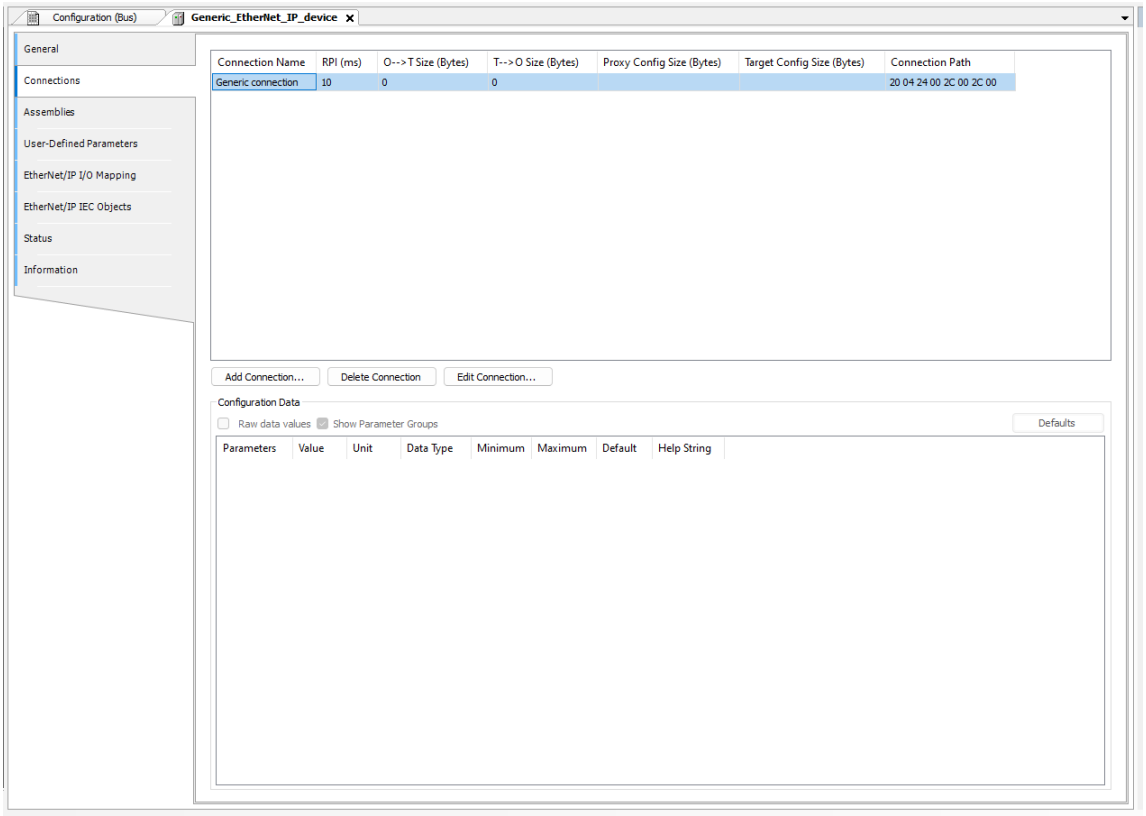


Figure 114: EtherNet/IP Connection Tab

Notes:

For two or more EtherNet/IP Scanners to connect to the same Remote Adapter:

1. Only one of the Scanners can establish an *Exclusive Owner* connection.
2. The same value of *RPI(ms)* must be configured for the Scanners.

The configuration data is defined in the EDS file. The data is transmitted to the remote adapter when the connection is opened.

Configuration	Description	Default Value	Options
RPI (ms)	Request Packet Interval: exchange interval of the input and output data.	10 ms	Multiple the Interval of the Bus Cycle Task to which it is associated
O -> T Size (Bytes)	Size of the producer data from the Scanner to the Adapter (O -> T)	0	0 - 65527
T -> O Size (Bytes)	Size of the consumer data from the Adapter to the Scanner (T -> O)	0	0 - 65531
Proxy Config Size (Bytes)	Proxy configuration data size	-	-
Device Config Size (Bytes)	Device configuration data size.	-	-
Connection Path	Address of the configuration objects - input objects - output objects.	Automatically generated path	Automatically generated path, User-defined path and Path defined by symbolic name

Table 100: EtherNet/IP Connection parameters

To *add* new connections there is the button *Add Connection...* which will open the *New connection* window. In this window, you can configure a new connection type from those predefined in the Adapter's EDS or a connection from zero when using a Generic device.

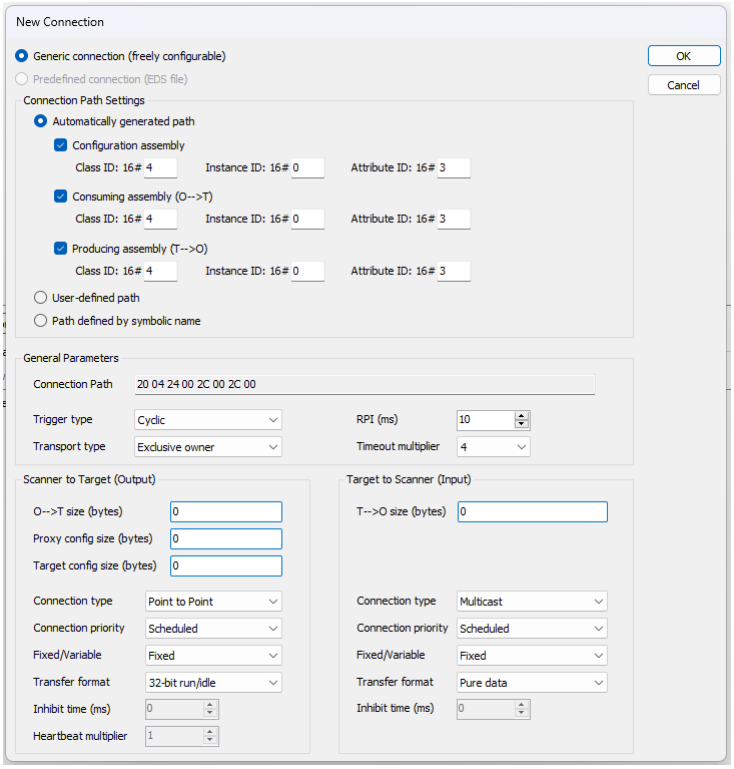


Figure 115: EtherNet/IP New Connection's Window

5.7.10.2.3. *Assemblies*

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

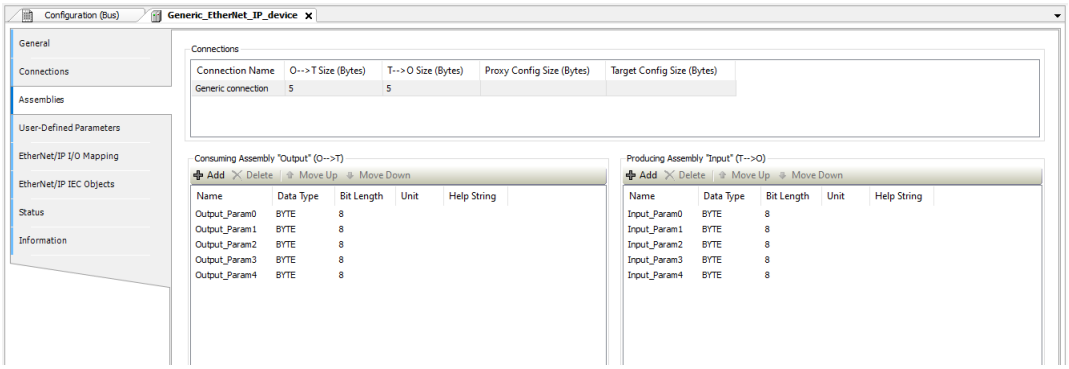


Figure 116: EtherNet/IP Assemblies Tab

Output Assembly and *Input Assembly*:

Configuration	Description
Add	Opens the dialog box “Add Input/Output”
Delete	Deletes all selected Inputs/Outputs.
Move Up	Moves the selected Input/Output within the list.
Move Down	The order in the list determines the order in the I/O mapping.

Table 101: EtherNet/IP Assemblies Tab

Dialog box *Add Input/Output*:

Configuration	Description
Name	Name of the input/output to be inserted.
Help String	
Data type	Type of the input/output to be inserted. This type also define its Bit Length.
Bit Length	This value must not be edited.

Table 102: EtherNet/IP “Add Input/Output” window

5.7.10.2.4. User-Defined Parameters

The *User-Defined Parameters* tab displays all additional parameters that are transmitted once only into the bus system during the phase of the starting procedure allotted to this. The parameters are transmitted to the remote adapter via acyclic services.

ATTENTION

The user parameters are also transmitted again when a connection is reestablished, for example after the failure of a remote adapter.

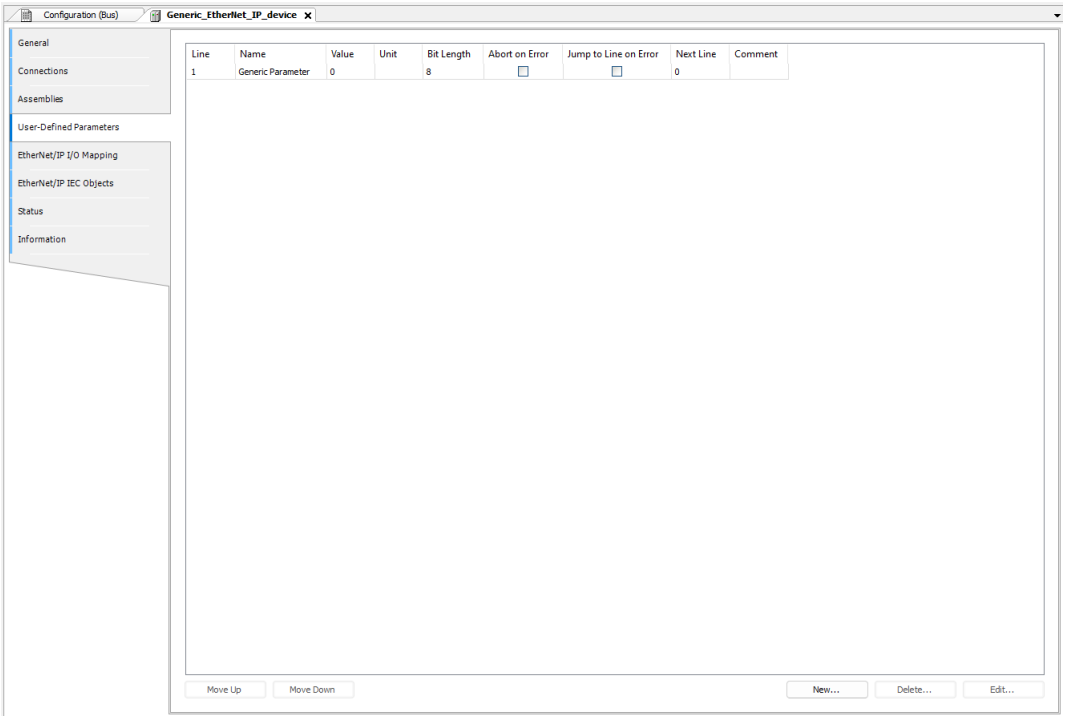


Figure 117: User-Defined Parameters Tab - EtherNet/IP

Configuration	Description
New	Opens the Select Parameters dialog for adding a new parameter.
Edit	Opens the Select Parameters dialog for changing an existing parameter.
Move up	Changes the order of the user parameters. The order of the parameters in the list corresponds to the order at the initialization.
Move down	
Value	The value of the respective parameter can be changed directly by double-clicking the value.
Abort If Error	In case of error, the entire transmission of the parameters is aborted.
Jump to Line If Error	In case of error, the program resumes with the line specified in the Next Line column. In this way, an entire block can be skipped during the initialization, or a return can be defined.

Table 103: User-Defined Parameters Tab - EtherNet/IP

Note:

Jump to Line If Error: A return can lead to an infinite loop if it is never possible to write a certain parameter.

5.7.10.2.5. EtherNet/IP I/O Mapping

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

5.7.10.3. EtherNet/IP Adapter Configuration

The EtherNet/IP Adapter requires Ethernet/IP Modules. The Modules will provide I/O mappings that can be manipulated by user application through %I or %Q addresses according to its configuration.

New Adapters can be installed on Mastertool with the EDS Files. The configuration options may differ depending on the device description file of the added Adapter.

5.7.10.3.1. General

The first tab of the EtherNet/IP Adapter is the *General* tab. Here you can set the parameters of the *Electronic Keying* used in the scanner to check compatibility. In this tab, you can also install the EDS of the device directly in the Mastertool device repository or export it.

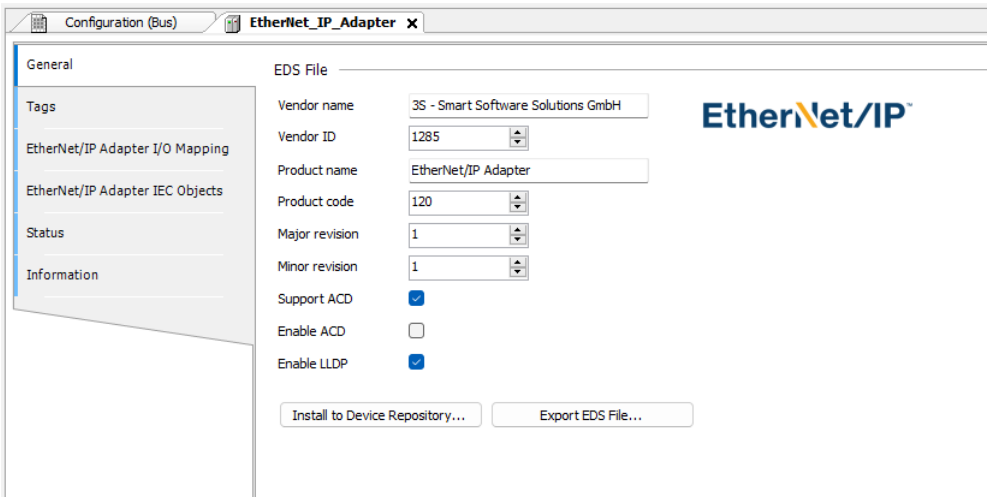


Figure 118: EtherNet/IP General Tab

5.7.10.3.2. EtherNet/IP Adapter: I/O Mapping

On the *EtherNet/IP I/O Mapping* tab, you can configure which bus cycle task the Adapter will execute.

5.7.10.4. EtherNet/IP Module Configuration

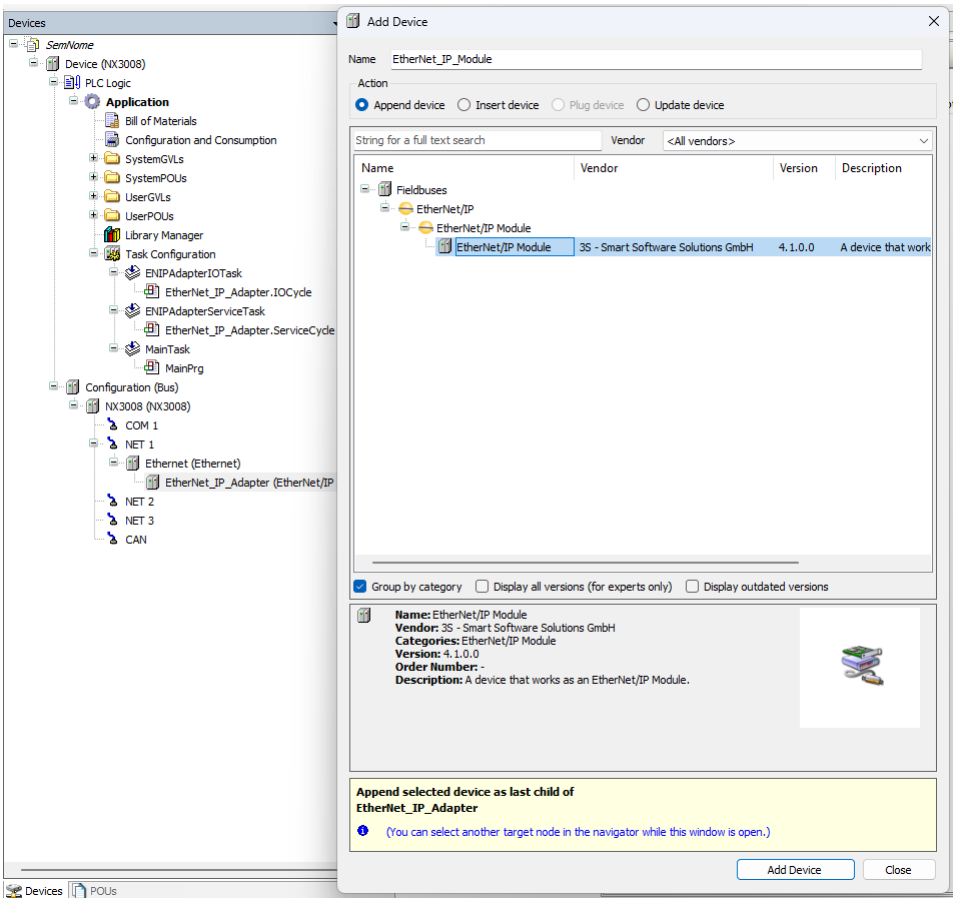


Figure 119: Adding an EtherNet/IP Module under the Adapter

5.7.10.4.1. Assemblies

The parameters of the module's *General* tab follow the same rules as described in the 101 and 102 tables.

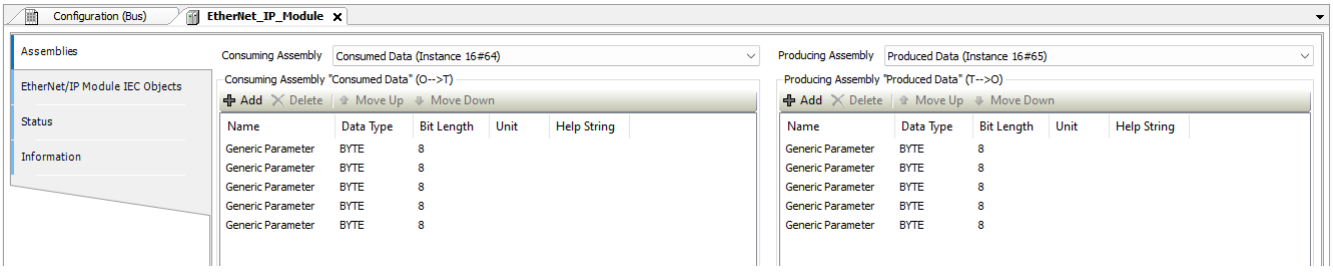


Figure 120: EtherNet/IP Module Assemblies Tab

5.7.10.4.2. EtherNet/IP Module: I/O Mapping

The I/O Mapping tab shows, in the *Variable* column, the name of the automatically generated Adapter instances. In this way, the instance can be accessed by the user application.

5.7.11. PROFINET Controller

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

5.8. Communication Performance

5.8.1. MODBUS Server

The MODBUS devices configurable in the Nexto CPU run in the background, with a priority below the user application and cyclically. Thus, their performance varies depending on the remaining time, taking into account the difference between the interval and time that the application takes to run. For example, a MODBUS device in an application that runs every 100 ms, with a running time of 50 ms, will have a lower performance than an application running every 50 ms to 200 ms of interval. It happens because in the latter case, the CPU will have a longer time between each MainTask cycle to perform the tasks with lower priority.

It also has to be taken into account the number of cycles that the device, slave or server takes to respond to a request. To process and transmit a response, a MODBUS RTU Slave will takes two cycles (cycle time of the MODBUS task), where as a MODBUS Ethernet Server task takes only one cycle. But this is the minimum time between receipt of a request and the reply. If the request is sent immediately after the execution of a task MODBUS cycle time may be equal to 2 or 3 times the cycle time for the MODBUS slave and from 1 to 2 times the cycle time for the MODBUS server.

In this case: Maximum Response Time = 3 * (cycle time) + (time of execution of the tasks) + (time interframe chars) + (send delay).

For example, for a MODBUS Ethernet Server task with a cycle of 50 ms, an application that runs for 60 ms every 100 ms, the server is able to run only one cycle between each cycle of the application. On the other hand, using the same application, running for 60 ms, but with an interval of 500 ms, the MODBUS performs better, because while the application is not running, it will be running every 50 ms and only each cycle of MainTask it will take longer to run. For these cases, the worst performance will be the sum of the Execution Time of the user application with the cycle time of the MODBUS task.

For the master and client devices the operating principle is exactly the same, but taking into account the polling time of the MODBUS relation and not the cycle time of the MODBUS task. For these cases, the worst performance of a relationship will be performed after the polling time, along with the user application Execution Time.

It is important to stress that the running MODBUS devices number also changes its performance. In an user application with Execution Time of 60 ms and interval of 100 ms, there are 40 ms left for the CPU to perform all tasks of lower priority. Therefore, a CPU with only one MODBUS Ethernet Server will have a higher performance than a CPU that uses four of these devices.

5.8.1.1. CPU's Integrated Interfaces

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

The table below exemplifies the number of requisitions that a MODBUS Server inserted in a integrated Ethernet interface is capable to answer, according to the cycle time configured for the MODBUS task and the number of active connections:

Number of Active Connections	Answered requisitions per second with the MODBUS task cycle at 5 ms	Answered requisitions per second with the MODBUS task cycle at 10 ms	Answered requisitions per second with the MODBUS task cycle at 20 ms
1 Connection	192	100	50
2 Connections	390	200	100
4 Connections	780	400	200
7 Connections	1354	700	350
10 Connections	1922	998	500

Table 104: Communication Rate of a MODBUS Server at Integrated Interface

ATTENTION

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

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

N = C x (1 / T)

Where:

N is the medium number of answers per second;

C is the number of active connections;

T is the MODBUS task interval in seconds.

As an example a MODBUS Server, with only one active connection and a cycle time of 50 ms we get:

C = 1; T = 0,05 s;

N = 1 x (1 / (0,05))

N = 20

That is, in this configuration the MODBUS Server answers, on average, 20 requisitions per second.

In case the obtained value is multiplied by the number of bytes in each requisition, we will obtain a transfer rate of n bytes per second.

5.8.1.2. Remote Interfaces

The performance of a device MODBUS Server in one remote Ethernet interface is similar to the performance in the CPU's integrated 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 integrated interfaces. 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 table below shows some calculated values:

Number of Connections	Requests responded per second with MODBUS task cycle time of 5 ms, 40 ms, and 55 ms (calculated)	Requests responded per second with MODBUS task cycle time of 70 ms (calculated)	Requests responded per second with MODBUS task cycle time of 100 ms (calculated)
1 Connection	17,85	14,29	9,99
2 Connections	35,69	28,58	19,98
4 Connections	71,4	57,16	39,96
7 Connections	124,95	100,03	69,93
10 Connections	178,5	142,9	99,9

Table 105: Communication Rate of a MODBUS Server on a NX5000 Interface

5.9. 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, the PLC consumes a certain amount of time for the task to be processed. At that time we

call it as *Execution Time*.

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

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

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

ATTENTION

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

ATTENTION

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

5.9.1. I/O Scan Time

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

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

Table 106: I/O Scanning Time

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

5.9.2. Memory Card

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

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

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

5.10. RTC Clock

The CPUs have an internal clock that can be used through the *NextoStandard.lib* library. This library is automatically loaded during the creation of a new project (to perform the library insertion procedure, see Libraries section). The figure below shows how to include the blocks in the project:

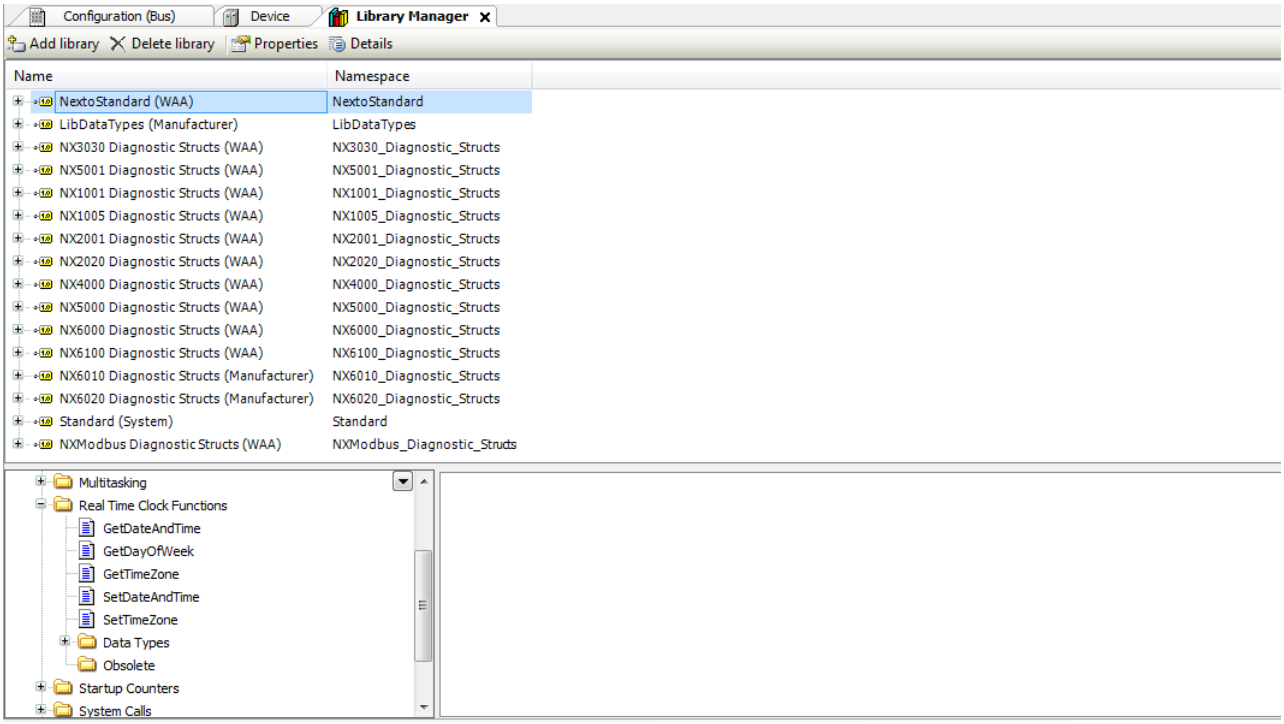


Figure 121: Clock Reading and Writing Blocks

ATTENTION

Function blocks of RTC Reading and Writing, previously available in 2.00 Mastertool or older become obsolete from 2.00 or newer, the following blocks are no longer used: *NextoGetDateAndTime*, *NextoGetDateAndTimeMs*, *NextoGetTimeZone*, *NextoSetDateAndTime*, *NextoSetDateAndTimeMs* and *NextoSetTimeZone*.

5.10.1. Function Blocks for RTC Reading and Writing

Among other function blocks, there are some very important used for clock reading (*GetDateAndTime*, *GetDayOfWeek* and *GetTimeZone*) and for date and time new data configuring (*SetDateAndTime* and *SetTimeZone*). These functions always use the local time, that is, take into account the value defined by the *Time Zone*.

The proceedings to configure these two blocks are described below.

ATTENTION

The obsolete functional blocks for reading and writing the RTC (*NextoGetDateAndTime*, *NextoGetDateAndTimeMs*, *NextoSetDateAndTime*, and *NextoSetDateAndTimeMs*) cannot be used in the redundant data area in redundant projects. They must be used in non-redundant POUs, such as the *NonSkippedPrg* POU. More details on how the *NonSkippedPrg* POU works can be found in NonSkippedPrg Program.

5.10.1.1. Function Blocks for RTC Reading

The clock reading can be made through the following functions:

5.10.1.1.1. GetDateAndTime

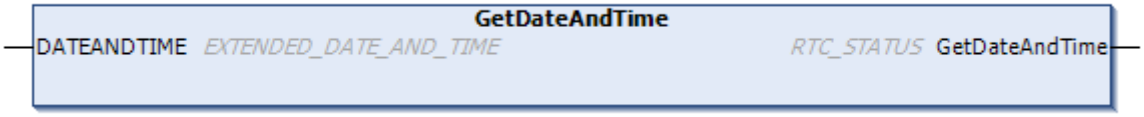


Figure 122: Date and Hour Reading

Input Parameters	Type	Description
DATEANDTIME	EXTENDED_DATE _AND_TIME	This variable returns the value of date and hour of RTC in the format shown at Table 116.

Table 107: Input Parameters of GetDateAndTime

Output Parameters	Type	Description
GETDATEANDTIME	RTC_STATUS	Returns the function error state, see Table 118.

Table 108: Output Parameters of GetDateAndTime

Utilization example in ST language:

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

5.10.1.1.2. GetTimeZone

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

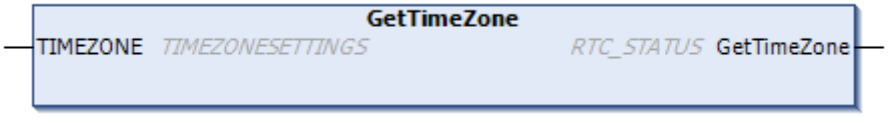


Figure 123: Configuration Reading of Time Zone

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

Table 109: Input Parameters of GetTimeZone

Output Parameters	Type	Description
GetTimeZone	RTC_STATUS	Returns the function error state, see Table 118.

Table 110: Output Parameters of GetTimeZone

Utilization example in ST language:

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

5.10.1.1.3. *GetDayOfWeek*

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



Figure 124: Day of Week Reading

Output Parameters	Type	Description
GetDayOfWeek	DAYS_OF_WEEK	Returns the day of the week. See Section 117.

Table 111: Output Parameters of GetDayOfWeek

When called, the function will read the day of the week and fill the structure *DAYS_OF_WEEK*.
Utilization example in ST language:

```
PROGRAM UserPrg
VAR
DayOfWeek : DAYS_OF_WEEK;
END_VAR
-------------------------------------------------------------------------
DayOfWeek := GetDayOfWeek();
```

5.10.1.2. RTC Writing Functions

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

5.10.1.2.1. SetDateAndTime

SetDateAndTime function is used to write the settings on the clock. Typically the precision is on the order of hundreds of milliseconds.

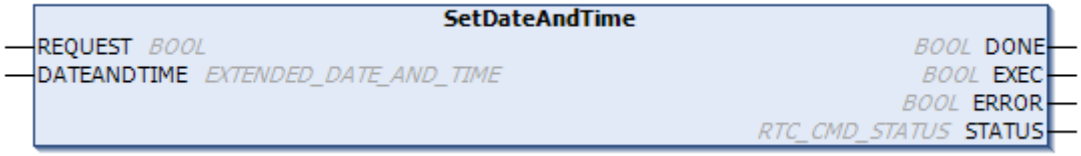


Figure 125: Set Date And Time

Input parameters	Type	Description
REQUEST	BOOL	This variable, when receives a rising edge, enables the clock writing.
DATEANDTIME	EXTENDED_DATE_AND_TIME	Receives the values of date and hour with milliseconds. See section 116.

Table 112: Input Parameters of SetDateAndTime

Output parameters	Type	Description
DONE	BOOL	This variable, when true, indicates that the action was successfully completed.
EXEC	BOOL	This variable, when true, indicates that the function is processing the values.
ERROR	BOOL	This variable, when true, indicates an error during the Writing.
STATUS	RTC_STATUS	Returns the error occurred during the configuration. See Table 118.

Table 113: Output Parameters of SetDateAndTime

When a rising edge occurs at the *REQUEST* input, the function block will write the new *DATEANDTIME* values on the clock. If the writing is successfully done, the *DONE* output will be equal to *TRUE*. Otherwise, the *ERROR* output will be equal to *TRUE* and the error will appear in the *STATUS* variable.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
SetDateAndTime : SetDateAndTime;
xRequest : BOOL;
DateAndTime : EXTENDED_DATE_AND_TIME;
xDone : BOOL;
xExec : BOOL;
xError : BOOL;
xStatus : RTC_STATUS;
END_VAR
-------------------------------------------------------------------------
IF xRequest THEN
  SetDateAndTime.REQUEST:=TRUE;
  SetDateAndTime.DATEANDTIME:=DateAndTime;
  xRequest:= FALSE;
END_IF
SetDateAndTime();
SetDateAndTime.REQUEST:=FALSE;
IF SetDateAndTime.DONE THEN
  xExec:=SetDateAndTime.EXEC;
  xError:=SetDateAndTime.ERROR;
  xStatus:=SetDateAndTime.STATUS;
END_IF
```

ATTENTION

If the user attempts to write time values outside the RTC range, the values will be converted to valid ones, as long as they do not exceed the range described in section RTC Operating Limits. For example, if the user tries to write the value 2000 ms, it will be converted to 2 seconds; if 100 seconds are written, it will be converted to 1 minute and 40 seconds; if 30 hours are written, it will be converted to 1 day and 6 hours, and so on.

5.10.1.2.2. SetTimeZone

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

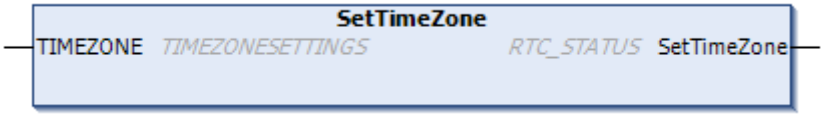


Figure 126: Writing of the Time zone Settings

Input parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	Structure with time zone to be configured. See Table 119.

Input parameters	Type	Description

Table 114: SetTimeZone Input Parameters

Output parameters	Type	Description
SetTimeZone	RTC_STATUS	Returns the error occurred during the reading/setting. See Table 118.

Table 115: SetTimeZone Output Parameters

When called, the function will configure the *TIMEZONE* with the new system time zone configuration. The configuration results is returned by the function.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Status : RTC_STATUS;
TimeZone : TIMEZONESETTINGS;
xWrite : BOOL;
END_VAR
-------------------------------------------------------------------------------
//FB SetTimeZone
IF (xWrite = TRUE) THEN
Status := SetTimeZone(TimeZone);
  IF Status = RTC_STATUS.NO_ERROR THEN
    xWrite := FALSE;
  END_IF
END_IF
```

ATTENTION

To adjust the clock, time and date values must be within the range described in section RTC Operating Limits; otherwise, an error will be reported through the *STATUS* output parameter. For more details about the *STATUS* output parameter, refer to section RTC_STATUS.

5.10.2. RTC Data Structures

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

5.10.2.1. EXTENDED_DATE_AND_TIME

This structure is used to store the RTC date when used the function blocks for date reading/setting within milliseconds of accuracy. It is described in the table below:

Structure	Type	Variable	Description
EXTENDED_DATE_AND_TIME	BYTE	byDayOfMonth	Stores the day of the set date.
	BYTE	ByMonth	Stores the month of the set date.
	WORD	wYear	Stores the year of the set date.
	BYTE	byHours	Stores the hour of the set date.
	BYTE	byMinutes	Stores the minutes of the set date.
	BYTE	bySeconds	Stores the seconds of the set date.

Structure	Type	Variable	Description
	WORD	wMilliseconds	Stores the milliseconds of the set date.

Table 116: EXTENDED_DATE_AND_TIME

5.10.2.2. DAYS_OF_WEEK

This structure is used to store the day of week:

Enumerable	Value	Description
DAYS_OF_WEEK	0	INVALID_DAY
	1	SUNDAY
	2	MONDAY
	3	TUESDAY
	4	WEDNESDAY
	5	THURSDAY
	6	FRIDAY
	7	SATURDAY

Table 117: DAYS_OF_WEEK Structure

5.10.2.3. RTC_STATUS

This enumerator is used to return the type of error in the RTC setting or reading and it is described in the table below:

Enumerator	Value	Description
RTC_STATUS	NO_ERROR (0)	There is no error.
	UNKNOWN_COMMAND (1)	Unknown command.
	DEVICE_BUSY (2)	Device is busy.
	DEVICE_ERROR (3)	Device with error.
	ERROR_READING_OSF (4)	Error in the reading of the valid date and hour flag.
	ERROR_READING_RTC (5)	Error in the date and hour reading.
	ERROR_WRITING_RTC (6)	Error in the date and hour writing.
	ERROR_UPDATING_SYSTEM _TIME (7)	Error in the update of the system's date and hour.
	INTERNAL_ERROR (8)	Internal error.
	INVALID_TIME (9)	Invalid date and hour.
	INPUT_OUT_OF_RANGE (10)	Out of the limit of valid date and hour for the system.
	SNTP_NOT_ENABLE (11)	Error generated when the SNTP service is not enabled and it is done an attempt for modifying the time zone.

Table 118: RTC_STATUS

5.10.2.4. TIMEZONESETTINGS

This structure is used to store the time zone value in the reading/setting requests of the RTC's function blocks and it is described in table below:

Structure	Type	Variable	Description
TIMEZONESETTINGS	INT	iHour	Set time zone hour.
	INT	iMinutes	Set time zone minute.

Table 119: TIMEZONESETTINGS

Note:

Function Blocks of Writing and Reading of Date and Hour: different libraries of *NextoStandard*, which have function blocks or functions that may perform access of reading and writing of date and hour in the system, are not indicated. The *NextoStandard* library has the appropriate interfaces for writing and reading the system's date and hour accordingly and for informing the correct diagnostics.

5.10.3. RTC Operating Limits

The values presented in Table 120 indicate the earliest and latest time instants that the RTC clock can represent. The minimum value corresponds to the earliest date and time that can be set, while the maximum value represents the latest valid future date and time. These limits must be respected when setting or reading the product's clock, ensuring that all date and time functions operate correctly.

Parameter	Minimum	Maximum
Day	1	31
Month	1	12
Year	2000	2099
Hour	0	23
Minute	0	59
Second	0	59
Millisecond	0	999

Table 120: Valid value range for the RTC Clock

5.11. User Files Memory

Nexto Series CPUs have a memory area destined to the general data storage, in other words, the user can store several project files of any format in the CPU memory. This memory area varies according to the CPU model used (check Memory section).

In order to use this area, the user must access a project in Mastertool and click on the *Devices Tree*, placed at the program left. Double click on the *Device* item and, after selecting the CPU in the *Communication Settings* tab which will be open, select the *Files* tab and click on *Refresh*, both in the computer files column (left) and in the CPU files column (right) as shown on figure below.

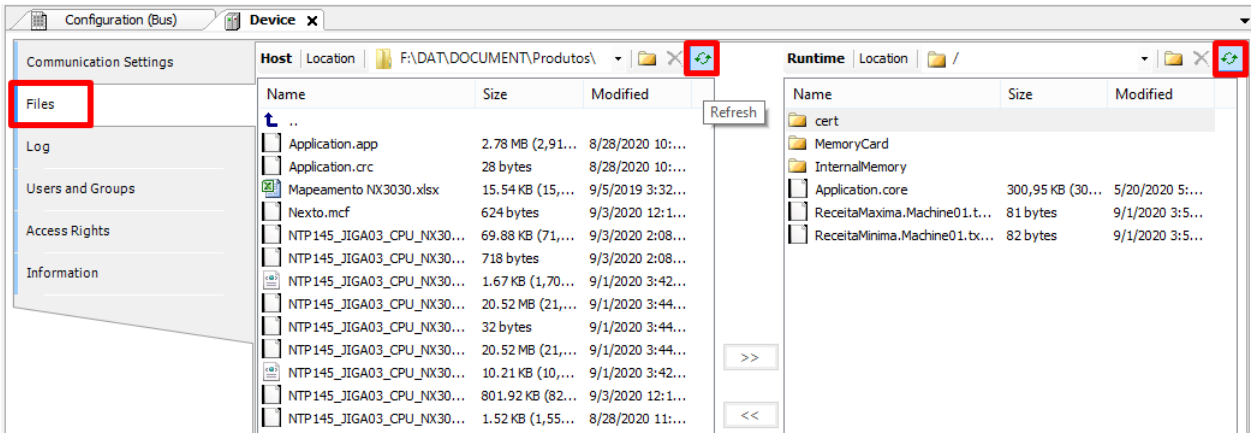


Figure 127: User Files Access

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

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

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

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

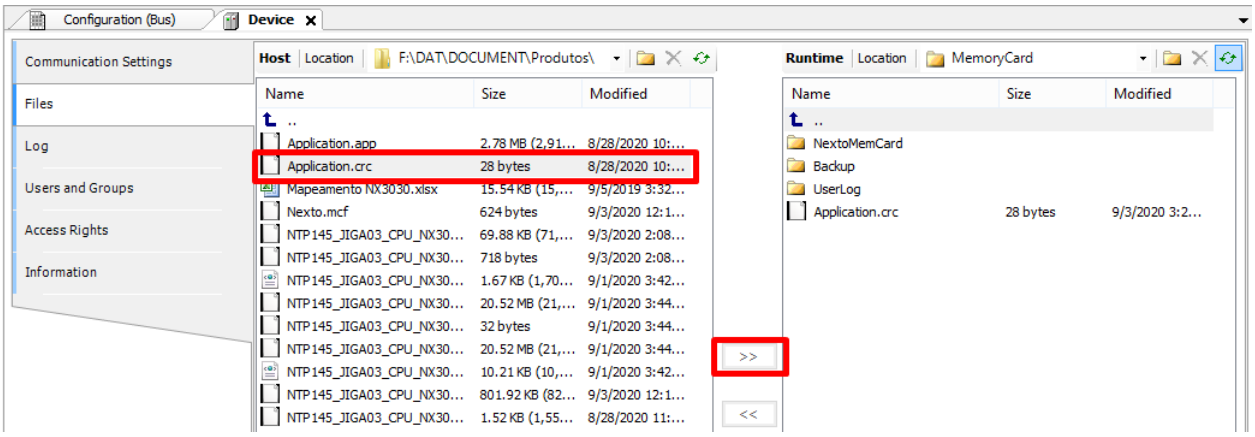


Figure 128: Files Transference

ATTENTION

The files contained in the folder of a project created by Mastertool have special names reserved by the system in this way cannot be transferred through the *Files* tab. If the user wishes to transfer a project to the user memory, you must compact the folder and then download the compressed file (*.zip for example).

In case it is necessary to transfer documents from the CPU to the PC in which Mastertool is installed, the user must follow a very similar procedure to the previously described, as the file must be selected from the right column and the button "«" pressed, placed on the center of the screen.

Furthermore, the user has some operation options in the storing files area, which are the following:

- New directory : allows the creation of a new folder in the user memory area.
- Delete item : allows the files excluding in the folders in the user memory area.
- Refresh : allows the file updating, on Mastertool screen, of the files in the user memory area and in the computer.

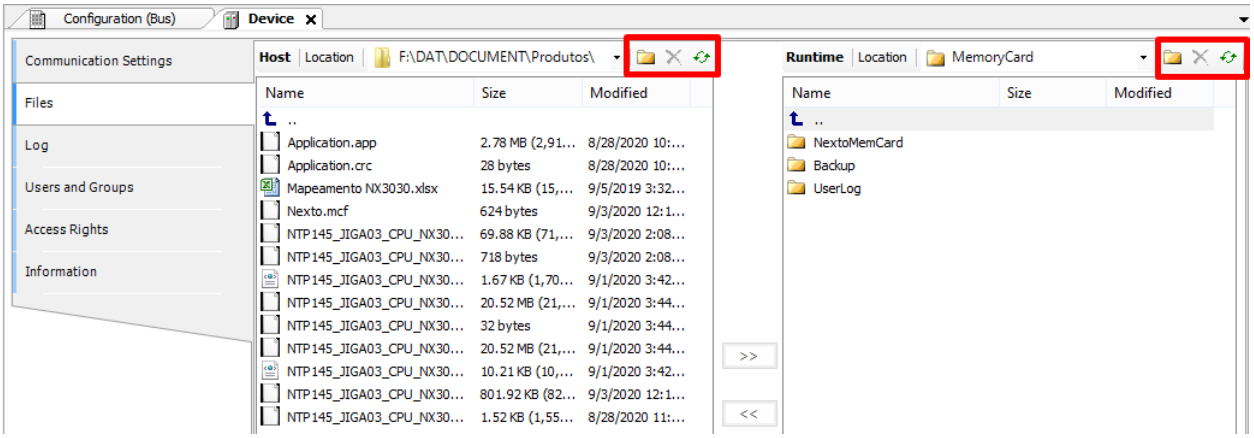


Figure 129: Utilization Options

ATTENTION

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

5.12. CPU's Informative and Configuration Menu

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

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

Level 1	Level 2	Level 3	Type
HARDWARE	TEMPERATURE	-	Informational
	CONTRAST	CONTRAST LEVEL	Configurable
	DATE AND TIME	-	Informational
	BACK	-	Return Level
LANGUAGES	ENGLISH	>ENGLISH	Configurable
	PORTUGUESE	>PORTUGUESE	Configurable
	BACK	-	Return Level
	IP ADDR. NET 1	-	Informational
	SUBNET MASK NET 1		Informational
	IP ADDR. NET 2		Informational
	SUBNET MASK NET 2		Informational
	IP ADDR. NET 3		Informational

Level 1	Level 2	Level 3	Type
NETWORK	SUBNET MASK NET 3		Informational
	IP ADDR. NET 4		Informational
	SUBNET MASK NET 4		Informational
	IP ADDR. NET 5		Informational
	SUBNET MASK NET 5		Informational
	IP ADDR. NET 6		Informational
	SUBNET MASK NET 6		Informational
	BACK		Return Level
REDUNDANCY	CPU IDENT.	-	Informational
	REMOTE STATE	-	Informational
	PROJECT SYNC.	-	Informational
	CHANGE STATE	CONFIRM?	Configurable
	BACK	-	Return Level
MEM. CARD	UNMOUNT	-	Configurable
	FORMAT	CONFIRM?	Configurable
	BACK	-	Return level
SOFTWARE	FIRMWARE	-	Informational
	BOOTLOADER		Informational
	COPROC		Informational
	BACK		Return Level

Table 121: CPU Menu Levels

Notes:

Memory Card: The memory card will only be available in the menu if it is inserted in the Nexto CPU.

Redundancy: The "*REDUNDANCY*" menu will only be available if the NX3035 CPU is identified as Redundant. As shown in Table 121, among the options available for viewing and changing are the main data required by the user, such as:

- Information about hardware resources:
  - TEMPERATURE – Internal temperature of the CPU (e.g., 36 C 97 F)
  - CONTRAST – Contrast adjustment of the front display of the CPU
  - DATE AND TIME – Date and time set on the CPU (e.g., 2001.01.31 00:00)
- Changing the CPU menu language:
  - PORTUGUESE – Changes the language to Portuguese
  - ENGLISH – Changes the language to English
- Viewing information about the network configured on the device:
  - END. IP NET 1 – IP address (e.g., 192.168.0.1)
  - MASK NET 1 – Subnet mask (e.g., 255.255.255.0)
  - IP ADDRESS NET 2 – IP address (e.g., 192.168.0.2)
  - NET MASK 2 – Subnet mask (e.g., 255.255.255.0)
  - IP ADDRESS NET 3 – IP address (e.g., 192.168.0.3)
  - NET MASK 3 – Subnet mask (e.g., 255.255.255.0)
  - IP ADDRESS NET 4 – IP address (e.g., 192.168.0.4)

  - NET MASK 4 – Subnet mask (e.g., 255.255.255.0)
  - IP ADDRESS NET 5 – IP address (e.g., 192.168.0.5)
  - NET MASK 5 – Subnet mask (e.g., 255.255.255.0)
  - NET IP ADDRESS 6 – IP address (e.g., 192.168.0.6)
  - NET MASK 6 – Subnet mask (e.g., 255.255.255.0)
- Access to PLC redundancy information:
  - PLC IDENT. – Provides the PLC identification in the redundancy. Possible information:
    - PLCA
    - PLCB
  - REMOTE STATE – Provides the state of the remote redundant PLC. Possible states:
    - ACTIVE
    - STANDBY
    - INACTIVE
    - NOT-CONFIGURED
    - STARTING
    - UNKNOWN
  - PROJECT SYNC – Reports the status of project synchronization.
    - CONNECTED – Synchronism enabled and redundancy link operational
    - NOT CONNECTED – Synchronism enabled and redundancy link inoperative
    - DISABLED – Synchronism disabled
    - SYNCHRONIZING – Synchronism in progress
    - SYNCHRONIZED – Synchronism successfully completed
  - CHANGE STATE – Allows the local PLC state to be changed. Possible options:
    - STANDBY – Available when the local PLC is Active/Inactive, and requests that the local PLC change to the Standby state.
    - INACTIVE – Available when the local PLC is Standby, and requests that the local PLC change to the Inactive state.
- Information about software versions:
  - FIRMWARE – CPU software version (e.g., 1.0.0.0)
  - BOOTLOADER – CPU bootloader version (e.g., 1.0.0.0)
  - COPROC – CPU coprocessor version (e.g., 1.0.0.0)

The figure below describes an example of how to operate the Nexto CPU menu, using the contrast adjustment procedure from the *Status* screen. In addition to facilitating configuration, it is possible to identify all screen levels and the type of press required to navigate between them. To modify other parameters, such as Language and commands in Redundancy, simply follow the same access logic. A short press shows that the contrast is being increased (brighter), and the next press after its maximum value returns it to the minimum value (darker). A long press confirms the desired contrast and returns to the previous level.

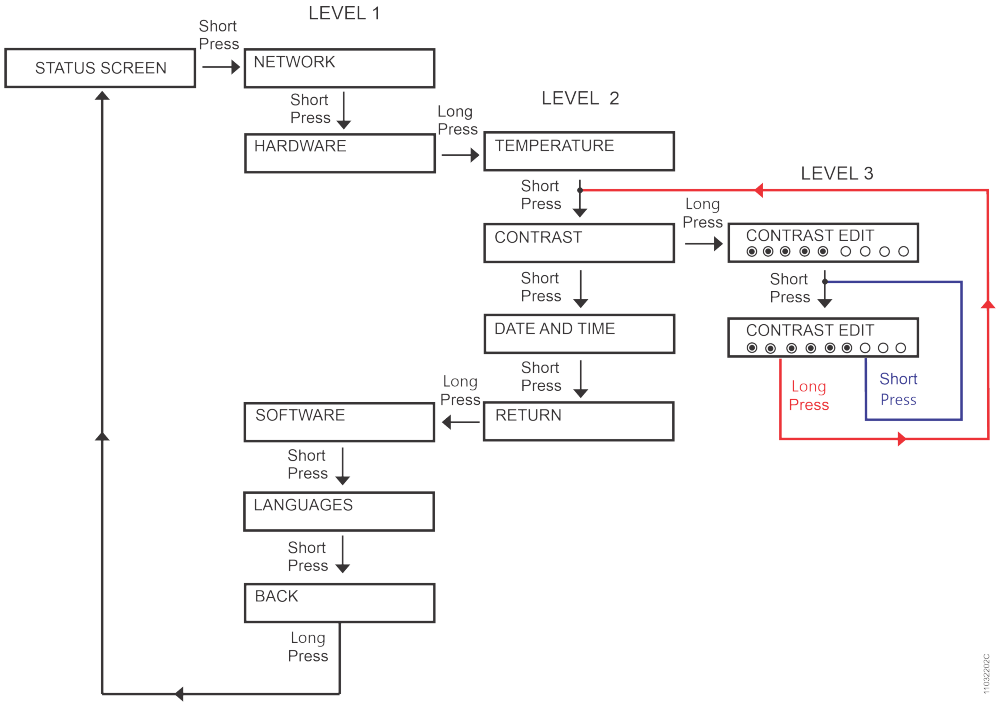


Figure 130: Contrast Adjust

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

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

5.13. Function Blocks and Functions

5.13.1. Special Function Blocks for Serial Interfaces

The special function blocks for serial interfaces make possible the local access (COM 1 and COM 2 - if available) and also access to remote serial ports (expansion modules). Therefore, the user can create his own protocols and handle the serial ports as he wishes, following the IEC 61131-3 languages available in Mastertool. The blocks are available inside the *NextoSerial* library which must be added to the project so it's possible to use them (to execute the library insertion procedure, see programming manual in section Library).

The special function blocks for serial interfaces can take several cycles (consecutive calls) to complete the task execution. Sometimes a block can be completed in a single cycle, but in the general case, needs several cycles. The task execution associated to a block can have many steps which some depend on external events, that can be significantly delayed. The function block cannot implement routines to occupy the time while waits for these events, because it would make the CPU busy. The solution could be the creation of blocking function blocks, but this is not advisable because it would increase the user application complexity, as normally, the multitask programming is not available. Therefore, when an external event is waited, the serial function blocks are finished and the control is returned to the main program. The task treatment continues in the next cycle, in other words, on the next time the block is called.

Before describing the special function blocks for serial interfaces, it is important to know the *Data types*, it means, the data type used by the blocks.

Data type	Options	Description
SERIAL_BAUDRATE	BAUD9600	Lists all baud rate possibilities (bits per second)
	BAUD19200	
	BAUD38400	
	BAUD57600	
	BAUD115200	

Data type	Options	Description
SERIAL_DATABITS	DATABITS_8	Lists all data bits possibilities.
	DATABITS_7	
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 modem signal possibilities for the configurations:	
	BAUDRATE	Defined in SERIAL_BAUDRATE.
	DATABITS	Defined in SERIAL_DATABITS.
	STOPBITS	Defined in SERIAL_STOPBITS.
	PARITY	Defined in SERIAL_PARITY.
	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. Valid values are: 1, 4, 8, or 14.
	MODE	Defined in SERIAL_MODE.
	ENABLE_RX_ON_TX	When true, all the received byte during the transmission will be discharged instead going to the RX line. Used to disable the full-duplex operation in the RS-422 interface.
SERIAL_PARITY	PARITY_NONE	List all parity possibilities.
	PARITY_ODD	
	PARITY_EVEN	
SERIAL_PORT	COM 1	List all available serial ports.
SERIAL_RX_CHAR_ EXTENDED	Defines a character in the RX queue in extended mode.	
	RX_CHAR	Data byte.
	RX_ERROR	Error code.
	RX_TIMESTAMP	Silence due to the previous character or due to another event which has happen before this character (serial port configuration, transmission ending).
SERIAL_RX_QUEUE_ STATUS	It has some fields which deliver information regarding RX queue status/error, used when the normal format is utilized (no error and timestamp information):	
	RX_FRAMING_ERRORS	Frame errors counter: character incorrect formation – no stop bit, incorrect baud rate, among other – since the serial port configuration. Returns to zero when it reaches the maximum value (65535).
	RX_PARITY_ERRORS	Parity errors counter, since the serial port configuration. Returns to zero when it reaches the maximum value (65535).
	RX_BREAK_ERRORS	Interruption errors counter, since the serial port configuration, in other words, active line higher than the character time. Returns to zero when it reaches the maximum value (65535).
	RX_FIFO_OVERRUN_ ERRORS	FIFO RX overrun errors counter, since the serial port configuration, in other words, error in the FIFO RX configured threshold. Returns to zero when it reaches the maximum value (65535).
	RX_QUEUE_OVERRUN_ ERRORS	RX queue overrun errors counter, in other words, the maximum characters number (1024) was overflowed and the data are being overwritten. Returns to zero when it reaches the maximum value (65535).

Data type	Options	Description
	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 - SERIAL_MODE - BAUDRATE - DATA_BITS - PARITY - STOP_BITS - UART_RX_THRESHOLD - TIMEOUT - TX_BUFF_LENGTH - 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.
	TX_TIMEOUT_ERROR	Time-out while waiting for the transmission ending in the SERIAL_TX.
	RX_TIMEOUT_ERROR	Time-out while waiting for all characters in the SERIAL_RX block or the SERIAL_RX_EXTENDED block.
	FB_SERIAL_RX_ NOT_ALLOWED	The SERIAL_RX isn't available for the RX queue, extended mode.
	FB_SERIAL_RX_ EXTENDED_NOT_ALLOWED	The SERIAL_RX_EXTENDED isn't available for the RX queue, normal mode.
	NOT_CONFIGURED	The function block can't be used before the serial port configuration.
	INTERNAL_ERROR	Indicates that an internal problem has ocurred in the serial port.
SERIAL_STOPBITS	STOPBITS_1	List all Stop Bits possibilities.
	STOPBITS_2	

Table 122: Serial Function Blocks Data types

Notes:

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

- One byte for the received data (RX_CHAR : BYTE): Store the six, seven or eight bits from the data received, depending on the serial communication configuration.
- One byte for the signal errors (RX_ERROR : BYTE): It has the format described below:
  - Bit 0: 0 - the character in bits 0 to 7 is valid. 1 - the character in bits 0 to 7 is not valid (or it cannot be valid), due to problems indicated in bits 10 to 15.
  - Bit 1: Not used.
  - Bit 2: Not used.
  - Bit 3: UART interruption error. The serial input remained in logic 0 (space) for a time greater than a character (start bit + data bits + parity bit + stop bits).
  - Bit 4: UART frame error. The logic 0 (space) was read when the first stop bit was expected and it should be logic 1 (mark).
  - Bit 5: UART parity error. The parity bit read is not correct according to the calculated one.
  - Bit 6: UART overrun error. Data was lost during the FIFO UART reading. New characters were received before the later ones were removed. This error will only be indicated in the first character read after the overrun error indication. This means some old data were lost.

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

- Two bytes for the timestamp signal (RX_TIMESTAMP : WORD): Indicates the silence time, within the 0 to 65535 interval, using 10 us as base. It saturates in 655.35 ms if the silence time is higher than 65535 units. The RX_TIMESTAMP of a character measures the time from a reference which can be any of the three options below:
  - On most of the cases, the end of the later character.
  - Serial port configuration.
  - The end of serial communication using the SERIAL_TX FB, in other words, when the last character is sent on line.

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

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

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

Table 123: Specific Configurations

5.13.1.1. SERIAL_CFG

This function block is used to configure and initialize the desired serial port. After the block is called, every RX and TX queue associated to the serial ports and the RX and TX FIFO are restarted.



Figure 131: 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 124: 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 125: SERIAL_CFG Output Parameters

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

```
PROGRAM UserPrg
VAR
Config: SERIAL_CFG;
Port: SERIAL_PORT := COM1;
Parameters: SERIAL_PARAMETERS := (BAUDRATE := BAUD9600,
DATABITS := DATABITS_8,
STOPBITS := STOPBITS_1,
PARITY := PARITY_NONE,
UART_RX_THRESHOLD := 8,
MODE :=NORMAL_MODE,
ENABLE_RX_ON_TX := FALSE);
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Config.REQUEST := TRUE;
Config.PORT := Port;
Config.PARAMETERS := Parameters;
//FUNCTION:
Config();
//OUTPUTS:
Config.DONE;
Config.EXEC;
Config.ERROR;
Status := Config.STATUS;    //If it is necessary to treat the error.
```

5.13.1.2. SERIAL_GET_CFG

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



Figure 132: 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 126: 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 127: SERIAL_GET_CFG Output Parameters

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

```
PROGRAM UserPrg
VAR
GetConfig: SERIAL_GET_CFG;
Port: SERIAL_PORT := COM1;
Parameters: SERIAL_PARAMETERS;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
GetConfig.REQUEST := TRUE;
GetConfig.PORT := Port;
```

```
//FUNCTION:
GetConfig();
//OUTPUTS:
GetConfig.DONE;
GetConfig.EXEC;
GetConfig.ERROR;
Status := GetConfig.STATUS; //If it is necessary to treat the error.
Parameters := GetConfig.PARAMETERS; //Receive the parameters of desired serial
   port.
```

5.13.1.3. SERIAL_GET_CTRL

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



Figure 133: 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 128: 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

Output parameters	Type	Description
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 129: SERIAL_GET_CTRL Output Parameters

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

```
PROGRAM UserPrg
VAR
Get_Control: SERIAL_GET_CTRL;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Get_Control.REQUEST := TRUE;
Get_Control.PORT := Port;
//FUNCTION:
Get_Control();
//OUTPUTS:
Get_Control.DONE;
Get_Control.EXEC;
Get_Control.ERROR;
Status := Get_Control.STATUS;   //If it is necessary to treat the error.
Get_Control.CTS_VALUE;
Get_Control.DSR_VALUE;
Get_Control.DCD_VALUE;
```

5.13.1.4. SERIAL_GET_RX_QUEUE_STATUS

This block is used to read some status information regarding the RX queue, specially developed for the normal mode, but it can also be used in the extended mode.



Figure 134: 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 130: 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 131: SERIAL_GET_RX_QUEUE_STATUS Output Parameters

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

```
PROGRAM UserPrg
VAR
Get_Status: SERIAL_GET_RX_QUEUE_STATUS;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
Status_RX: SERIAL_RX_QUEUE_STATUS;
END_VAR
//INPUTS:
Get_Status.REQUEST := TRUE;
Get_Status.PORT := Port;
//FUNCTION:
Get_Status();
//OUTPUTS:
Get_Status.DONE;
Get_Status.EXEC;
Get_Status.ERROR;
Status := Get_Status.STATUS;     //If it is necessary to treat the error.
Status_RX := Get_Status.RXQ_STATUS; //If it is necessary to treat the error of
    the RX.
```

5.13.1.5. SERIAL_PURGE_RX_QUEUE

This function block is used to clean the serial port RX queue, local and remote. The UART RX FIFO is restarted too.



Figure 135: 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 132: 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 133: SERIAL_PURGE_RX_QUEUE Output Parameters

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

```
PROGRAM UserPrg
VAR
Purge_Queue: SERIAL_PURGE_RX_QUEUE;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Purge_Queue.REQUEST := TRUE;
Purge_Queue.PORT := Port;
//FUNCTION:
Purge_Queue();
//OUTPUTS:
Purge_Queue.DONE;
Purge_Queue.EXEC;
Purge_Queue.ERROR;
Status := Purge_Queue.STATUS; //If it is necessary to treat the error.
```

5.13.1.6. SERIAL_RX

This function block is used to receive a serial port buffer, using the RX queue normal mode. In this mode, each character in the RX queue occupy a single byte which has the received data, storing 5, 6, 7 or 8 bits, according to the serial interface configuration.



Figure 136: 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_TIMEOUT	UINT	Specify the time-out to receive the expected character quantity. In case it is smaller than the necessary to receive the characters, the RX_TIME-OUT_ERROR output from the STATUS parameter will be indicated. When the specified value, in ms, is equal to zero, the function will return the data within the buffer.

Table 134: 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.

Output parameters	Type	Description
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_RX_BUFF_LENGTH - RX_TIMEOUT_ERROR - FB_SERIAL_RX_NOT_ALLOWED - NOT_CONFIGURED
RX_RECEIVED	UINT	Returns the received characters number. This number can be within zero and the configured value in RX_BUFFER_LENGTH. In case it is smaller, an error will be indicated by the function block.
RX_REMAINING	UINT	Returns the number of characters which are still in the RX queue after the function block execution.

Table 135: SERIAL_RX Output Parameters

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

```
PROGRAM UserPrg
VAR
Receive: SERIAL_RX;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..1023] OF BYTE;    //Max size.
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Receive.REQUEST := TRUE;
Receive.PORT := Port;
Receive.RX_BUFFER_POINTER := ADR(Buffer_Pointer);
Receive.RX_BUFFER_LENGTH := 1024;     //Max size.
Receive.RX_TIMEOUT := 10000;
//FUNCTION:
Receive();
//OUTPUTS:
Receive.DONE;
Receive.EXEC;
Receive.ERROR;
Status := Receive.STATUS;   //If it is necessary to treat the error.
Receive.RX_RECEIVED;
Receive.RX_REMAINING;
```

5.13.1.7. SERIAL_RX_EXTENDED

This function block is used to receive a serial port buffer using the RX queue extended mode as shown in the Serial Interface Configuration section.

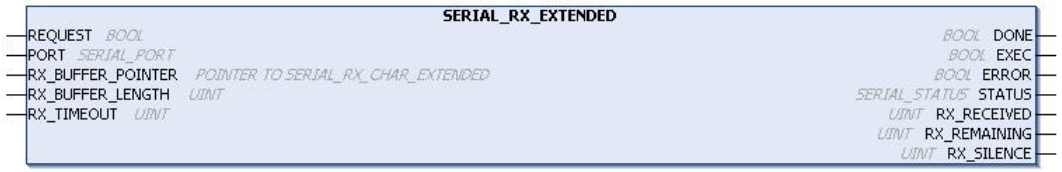


Figure 137: Block Used for Reception Buffer Reading

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_TIMEOUT	UINT	Specify the time-out to receive the expected character quantity. In case it is smaller than the necessary to receive the characters, the RX_TIMEOUT_ERROR output from the STATUS parameter will be indicated. When the specified value, in ms, is equal to zero, the function will return the data within the buffer.

Table 136: SERIAL_RX_EXTENDED Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_RX_BUFF_LENGTH - RX_TIMEOUT_ERROR - FB_SERIAL_RX_EXTENDED_NOT _ALLOWED - NOT_CONFIGURED
RX_RECEIVED	UINT	Returns the received characters number. This number can be within zero and the configured value in RX_BUFFER_LENGTH. In case it is smaller, an error will be indicated by the function block.

Output parameters	Type	Description
RX_REMAINING	UINT	Returns the number of characters which are still in the RX queue after the function block execution.
RX_SILENCE	UINT	Returns the silence time in the RX queue, measured since the last received character is finished. The time unit is 10 μs. This output parameter type is important to detect the silence time in protocols as MODBUS RTU. It might not be the silence time after the last received character by this function block, as it is only true if RX_REMAINING = 0.

Table 137: SERIAL_RX_EXTENDED Output Parameters

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

```
PROGRAM UserPrg
VAR
Receive_Ex: SERIAL_RX_EXTENDED;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..1023] OF SERIAL_RX_CHAR_EXTENDED;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Receive_Ex.REQUEST := TRUE;
Receive_Ex.PORT := Port;
Receive_Ex.RX_BUFFER_POINTER := ADR(Buffer_Pointer);
Receive_Ex.RX_BUFFER_LENGTH := 1024;      //Max size.
Receive_Ex.RX_TIMEOUT := 10000;
//FUNCTION:
Receive_Ex();
//OUTPUTS:
Receive_Ex.DONE;
Receive_Ex.EXEC;
Receive_Ex.ERROR;
Status := Receive_Ex.STATUS;  //If it is necessary to treat the error.
Receive_Ex.RX_RECEIVED;
Receive_Ex.RX_REMAINING;
Receive_Ex.RX_SILENCE;
```

5.13.1.8. SERIAL_TX

This function block is used to transmit a data buffer through serial port. It is finalized after all bytes were copied to internal TX queue or after time-out (generating errors).



Figure 138: 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_TIMEOUT	UINT	Specify the time-out to complete the transmission including the handshake phase. The specified value, in ms, must be positive and different than zero.
DELAY_BEFORE_ TX	UINT	Specify the delay, in ms, between the function block call and the transmission beginning. This variable can be used in communications with some modems.
CLEAR_RX_ BEFORE_TX	BOOL	When true, the RX queue and the UART FIFO RX are erased before the transmission beginning. This behavior is typical in half-duplex master/slave protocols.

Table 138: SERIAL_TX Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_TX_BUFF_LENGTH - ILLEGAL_TIMEOUT - CTS_TIMEOUT_ON - CTS_TIMEOUT_OFF - TX_TIMEOUT_ERROR - NOT_CONFIGURED
TX_TRANSMITTED	UINT	Returns the transmitted byte number which must be equal to TX_BUFFER_LENGTH, but can be smaller in case some error has occurred during transmission.

Table 139: SERIAL_TX Output Parameters

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

```
PROGRAM UserPrg
VAR
Transmit: SERIAL_TX;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..9] OF BYTE := [0,1,2,3,4,5,6,7,8,9];
```

```
Status: SERIAL_STATUS;
END_VAR

//INPUTS:
Transmit.REQUEST := TRUE;
Transmit.PORT := Port;
Transmit.TX_BUFFER_POINTER := ADR(Buffer_Pointer);
Transmit.TX_BUFFER_LENGTH := 10;
Transmit.TX_TIMEOUT := 10000;
Transmit.DELAY_BEFORE_TX := 1000;
Transmit.CLEAR_RX_BEFORE_TX := TRUE;
//FUNCTION:
Transmit();
//OUTPUTS:
Transmit.DONE;
Transmit.EXEC;
Transmit.ERROR;
Status := Transmit.STATUS;    //If it is necessary to treat the error.
Transmit.TX_TRANSMITTED;
```

5.13.2. Inputs and Outputs Update

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

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

ATTENTION

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

5.13.2.1. REFRESH_INPUT

This function block is used to update the specified module inputs without the necessity to wait for the cycle to be completed. It is important to notice that the filters configured in the Mastertool 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.

ATTENTION

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

ATTENTION

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

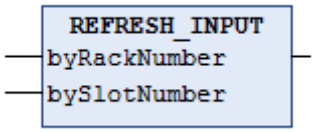


Figure 139: Block for Input Updating

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

Table 140: REFRESH_INPUT Input Parameters

Utilization example in ST language:

```
PROGRAM UserPrg
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.
```

5.13.2.2. REFRESH_OUTPUT

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

ATTENTION

The *REFRESH_OUTPUT* function must only be used in MainTask.

ATTENTION

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

ATTENTION

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

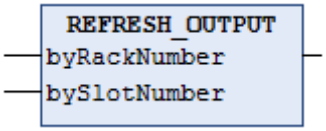


Figure 140: Block for Output Updating

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

Table 141: REFRESH_OUTPUT Input Parameters

Utilization example in ST language:

```
PROGRAM UserPrg
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.
```

5.13.3. Timer Retain

The timer retain is a function block developed for applications as production line clocks, that need to store its value and restart the counting from the same point in case of power supply failure. The values stored by the function block, are only zero in case of a *Reset Cold*, *Reset Origin* or a new application *Download* (see Mastertool's user manual), when the counters keep working, even when the application is stopped (Stop Mode).

ATTENTION

It is important to stress that, for the correct functioning of the Timer Retain blocks, the variables must be declared as Retain (*VAR RETAIN*). It's also important to notice that in simulation mode, the Timer Retain function blocks do not run properly due to need the Nexto CPU for correct behavior.

The three blocks already available in Mastertool *NextoStandard* library are described below (for the library insertion proceeding, see programming manual, section Library).

5.13.3.1. TOF_RET

The function block *TOF_RET* implements a time delay to disable an output. When the input *IN* has its state changed from (TRUE) to (FALSE), or a falling edge, the specified time *PT* will be counted and the *Q* output will be driven to (FALSE) at the end of it. When the input *IN* is in logic level 1 (TRUE), the output *Q* remain in the same state (TRUE), even if this happened in the middle of the counting process. The *PT* time can be changed during the counting as the block assumes the new value if the counting hasn't finished. Figure 141 depicts the *TOF_RET* block and Figure 142 shows its graphic behavior.



Figure 141: 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 142: 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 143: TOF_RET Output Parameters

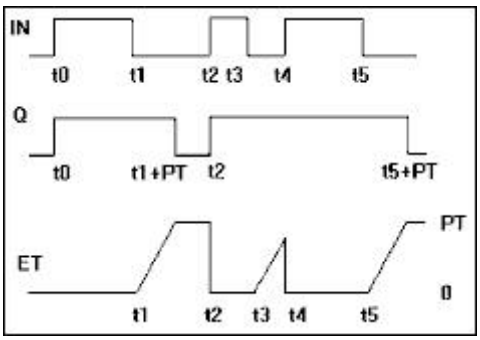


Figure 142: TOF_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL := TRUE;
TOF_RET : TOF_RET;
END_VAR
```

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

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

5.13.3.2. TON_RET

The *TON_RET* implements a time delay to enable an output. When the input *IN* has its state changed from (FALSE) to (TRUE), or a rising edge, the specified time *PT* will be counted and the *Q* output will be driven to (TRUE) at the end of it. When the input *IN* is in logic level 0 (FALSE), the output *Q* remain in the same state (FALSE), even if it happens in the middle of the counting process. The *PT* time can be changed during the counting as the block assumes the new value if the counting hasn't finished. Figure 143 depicts the *TON_RET* block and Figure 144 shows its graphic behavior.



Figure 143: 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 144: 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 145: TON_RET Output Parameters

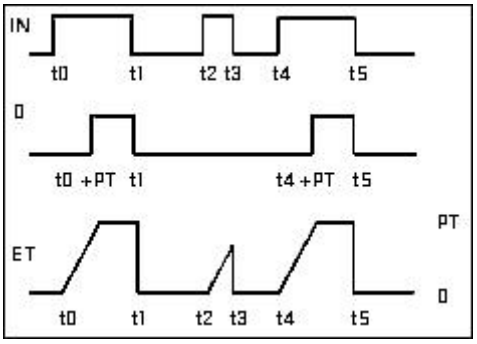


Figure 144: TON_RET Block Graphic Behavior

Utilization example in ST language:

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

// Quando bStart=TRUE starts counting
TON_RET( IN := bStart,
PT := T#20S);

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

5.13.3.3. TP_RET

The *TP_RET* function block works as a trigger. The timer which starts when the *IN* input has its state changed from (FALSE) to (TRUE), that is, a rising edge, it is increased until the *PT* time limit is reached. During the counting, the *Q* output is (TRUE), otherwise it is (FALSE). The *PT* time can be changed during the counting as the block assumes the new value if the counting has not finished. Figure 145 depicts the *TP_RET* and Figure 146 shows its graphic behavior.



Figure 145: 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 146: 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 147: TP_RET Output Parameters

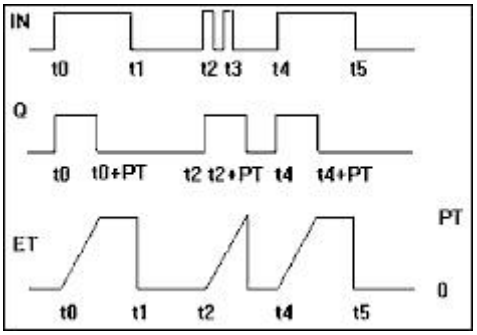


Figure 146: TP_RET Block Graphic Behavior

Utilization example in ST language:

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

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

bStart := FALSE;

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

5.13.4. Non-Redundant Timer

The non-redundant timer is used in applications for redundant CPUs that require 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 standby half-cluster assumes the active status and the active one goes for standby.

The three types of blocks already available in the *NextoStandard* library of the Mastertool software are describe as follows (for doing the procedure of library's inclusion, check IEC 61131 Programming Manual, section Library).

5.13.4.1. 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 147: TOF_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedPrg
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
```

5.13.4.2. 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.

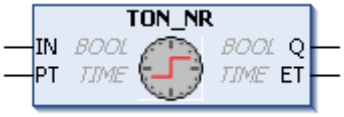


Figure 148: TON_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedPrg
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
```

5.13.4.3. 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.

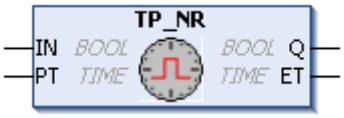


Figure 149: TP_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedPrg
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
```

5.13.5. User Log

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

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

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

Table 148: Data Type for User Log

The following are described the two functions available in the *LibLogs* library on Mastertool. To perform the procedure of inserting a library, see the IEC 61131 Programming Manual, section Libraries.

ATTENTION

The User Logs are available only until version 1.3.0.20 of Nexto Series CPUs. In the same way to use this feature is necessary version 1.40 or higher of Mastertool.

5.13.5.1. UserLogAdd

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

The *UserLogAdd* function can be used to enter multiple messages within a single task and also in different application tasks. However independent of each execution of the function in the application, being on the same task or on different tasks, all use the same feature to record the desired messages. For this reason it is recommended that the addition of messages using the *UserLogAdd* function in the application be held every 50 ms to prevent the return of buffer overload. If the function is performed in periods shorter than the indicated, but respect the average time of 50 ms between each message addition at the end of the interval for the task, also prevents the return of buffer overload. So that the logs are added correctly, it is important to respect time limits when the card is inserted and at startup of the CPU, mentioned in section Memory Card. After the operation the function returns the options for the given type *USER_LOG_ERROR_CODES* as Table 148. 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 below represents the function *UserLogAdd* and table below the input parameters:

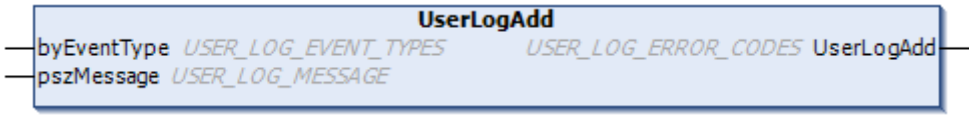


Figure 150: UserLogAdd Function

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 149: UserLogAdd Input Parameters

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

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

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

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
  eLogError :  LibLogs.USER_LOG_ERROR_CODES;
  sMessage  :  LibLogs.USER_LOG_MESSAGE;
  m_rTemperature : REAL;
  MAX_TEMPERATURE_ACCEPT : REAL;
END_VAR

IF (m_rTemperature > MAX_TEMPERATURE_ACCEPT) THEN
  sMessage := 'Temperature above expected: ';
  sMessage := Standard.CONCAT(sMessage, REAL_TO_STRING(m_rTemperature));
  sMessage := Standard.CONCAT(sMessage, '°');
  eLogError := LibLogs.UserLogAdd(LibLogs.USER_LOG_EVENT_WARN, sMessage);
  //The variable eLogError receives possible errors from the function.
END_IF
```

Log file content example: (*UserLog-20130827150624573 8.csv*)

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

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

5.13.5.2. UserLogDeleteAll

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

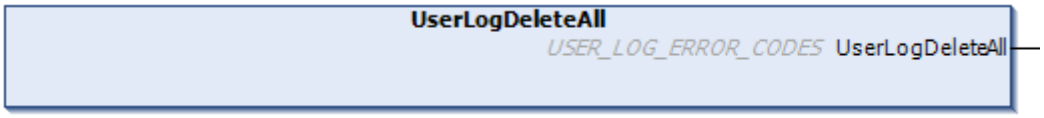


Figure 151: UserLogDeleteAll Function

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
  eLogError : LibLogs.USER_LOG_ERROR_CODES;
  m_DeleteLogs: BOOL;
END_VAR

IF (m_DeleteLogs = TRUE) THEN
  //The variable eLogError receives possible errors from the function.
  eLogError := LibLogs.UserLogDeleteAll();
  m_DeleteLogs := FALSE;
END_IF
```

ATTENTION

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

5.14. System Web Page

The *System Web Page* of the Nexto Series controllers provides status information and controller settings, which will be described in the following sections.

5.14.1. General Overview

To access the System Web Page of the desired CPU, simply use a desktop browser and enter the IP address corresponding to the CPU in the address bar (e.g., http://192.168.15.1). Initially, the general information of the CPU will be displayed, as shown in the figure below.



Figure 152: Initial Screen

5.14.2. Bus Information

Developed to display information and diagnostics related to the bus used by the controller. The Bus Information tab allows the user to view the status of the modules, identify connected devices, and access diagnostic and status data for each module. The features available in this tab vary according to the type of bus and the functionalities supported by the controller in use.

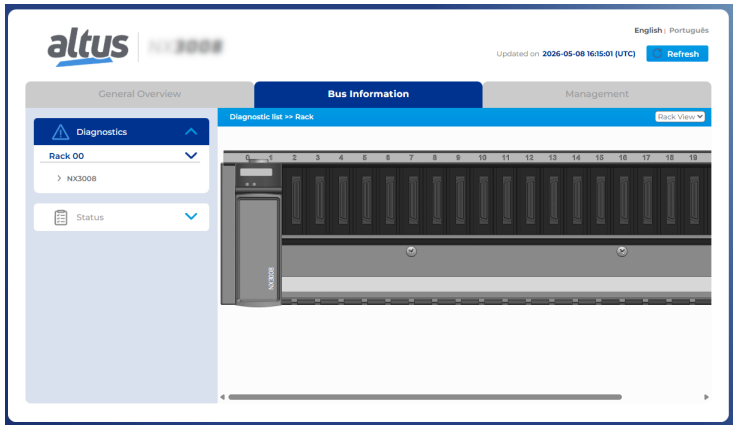


Figure 153: Bus Information

When there is no application in the CPU, no image will be displayed. When the Rack view is used, modules with diagnostics will blink and turn red. Otherwise, a list of the modules present in the system, the corresponding Tags, and the number of active diagnostics will be displayed.

When the module with diagnostics is pressed, the module active(s) diagnostic(s) are shown, as shown in the figure below:

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.

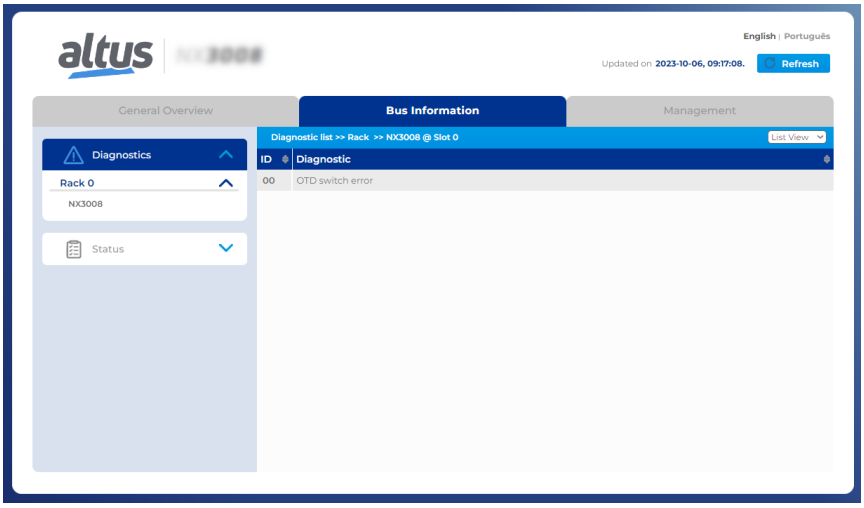


Figure 154: System Diagnostics

In case the Status tab is selected, the state of all detailed diagnostics is shown on the screen, as shown in the figure below:

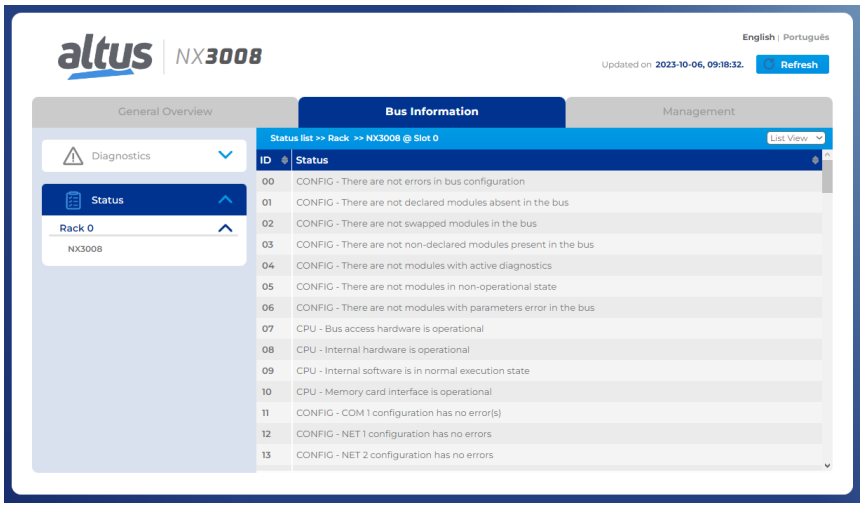


Figure 155: System Status

The user can choose to visualize two language options: Portuguese and English. Simply change in the upper right part of the screen to the desired language.

5.14.3. Management

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

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

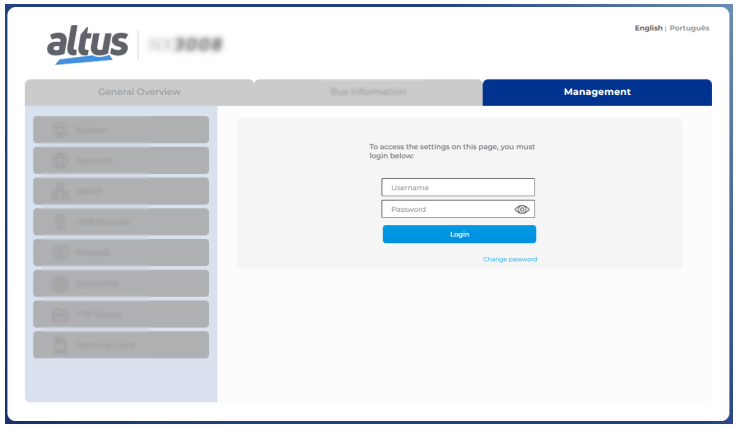


Figure 156: Management Tab Access

5.14.3.1. System

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

5.14.3.1.1. Firmware Update

On the *System* page, it is possible to update the controller's firmware, allowing the user to install the desired version. The *Firmware Update* section displays information about the current status of the controller application, the installed firmware version, and the status of the update process.

To perform the update, the user must select the corresponding firmware file. Optionally, the user can choose whether the application should be erased after the update by enabling the *Erase Application* option. After configuring the desired settings, the user must click the *Start Update* button. The system will then request confirmation of the operation through authentication using the configured username and password.

The latest firmware version is available on the Altus website. Access to the firmware update feature requires prior user authentication, as illustrated in the figure below.

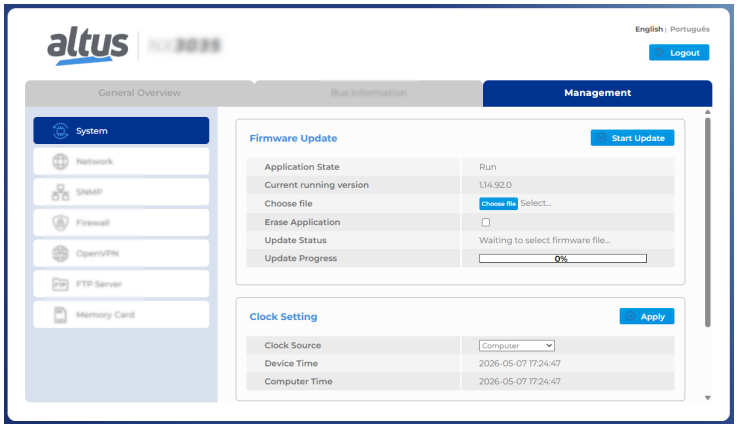


Figure 157: Firmware Update

ATTENTION

For redundant CPUs, this feature is available only when the CPU is in the INACTIVE state or in STOP (Not Configured). The feature is not available when the CPU is in the STANDBY or ACTIVE state.

5.14.3.1.2. Clock Setting

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

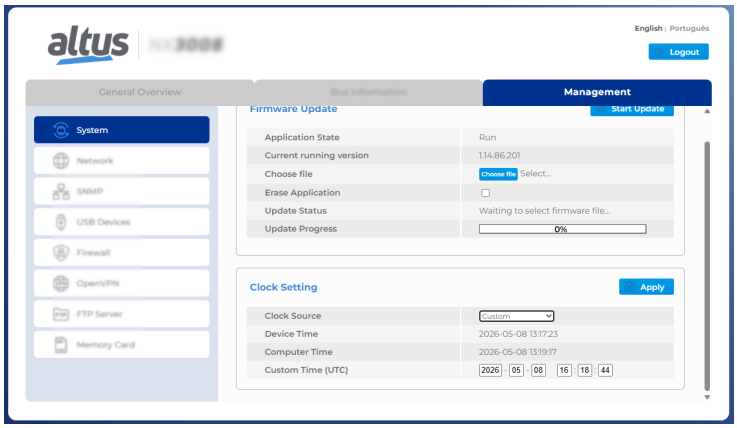


Figure 158: Clock Setting

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

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

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

5.14.3.1.3. SysLog

Syslog (System Logging Protocol) is a protocol that allows devices to send log messages to a centralized log collection and storage system. These messages can include information about system events, errors, warnings, and other activities relevant to system administration and security.

The CPU can be configured as a SysLog Client from the System section in the Management tab of the CPU System web page. As shown in the figure below.

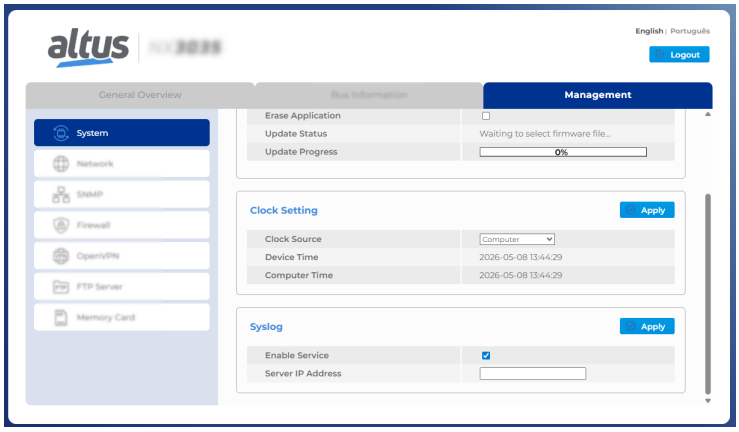


Figure 159: SysLog Configuration

The SysLog configuration consists of only two fields. The *"Enable Service"* checkbox is disabled by default and must be checked to activate the service. The IP address of the SysLog server to be used is configured in the *"Server IP Address"* field.

After configuring the server IP address and enabling the service, the settings are applied by clicking the *"Apply"* button. It is important that the Syslog server is properly configured to communicate througth UDP protocol, using port 514.

ATTENTION

The CPU implements Syslog messages compliant with RFC 5424 and UDP transport according to RFC 5426. Verify the compatibility of the Syslog Server solution before deployment.

When configured, SysLog will send all logs present in Mastertool's System Logs to the server, as detailed in the System Log section. The SysLog service only sends logs with a priority of WARNING or higher, as indicated in the classification below:

Mastertool Category	Syslog Category	Priority
Exception	Emergency	0
Error	Error	3
Warning	Warning	4

Table 150: Syslog Message Priority

5.14.3.1.4. System Control

On the System Web Page, it is possible to perform system control operations directly through the web interface. The System Control option provides a dedicated environment for device management actions, allowing the user to carry out the device reboot procedure. For this, it is necessary to click the *Reboot* button. If the operator wants to erase the application that exists, it is necessary to check the *Erase Application* option before performing the operation. The feature requires user authentication before any system control operation can be executed, as shown in the figure below.

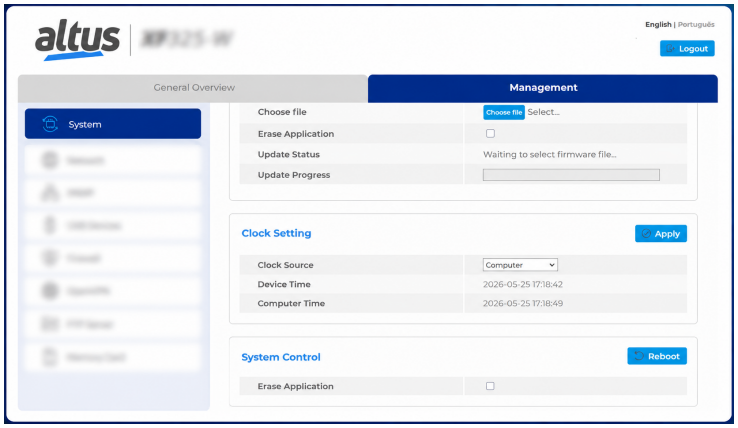


Figure 160: System Control

ATTENTION

For redundant CPUs, this feature is available only when the CPU is in the INACTIVE state or in STOP (Not Configured). The feature is not available when the CPU is in the STANDBY or ACTIVE state.

5.14.3.2. Network

Designed to improve controller usability, the Network section allows changing network addresses and executing the network Sniffer. The number of Ethernet interfaces displayed depends on the specifications of each controller.

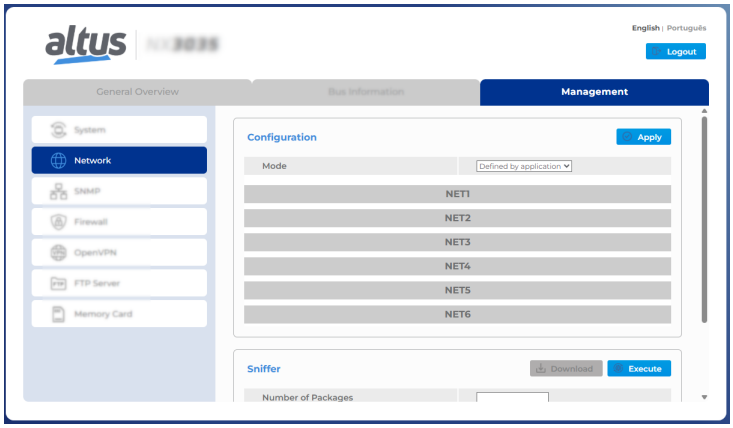


Figure 161: Network Web Page

5.14.3.2.1. Configuration

ATTENTION

In a redundant PLC, the configuration is defined by the application and the Mode field is disabled.

1. **Defined by application**
   The Mode field defines which configuration the controller should load for its interfaces. This field can be configured as *Defined By Web Page* or *Defined By Application*.
   When set to *Defined by Application*, the interface table is disabled, not allowing changes, as shown in the figure below. In this mode, the settings applied to the controller are those defined by the application.
   Below is an image with *Defined by Application* mode selected, showing the interface table disabled.

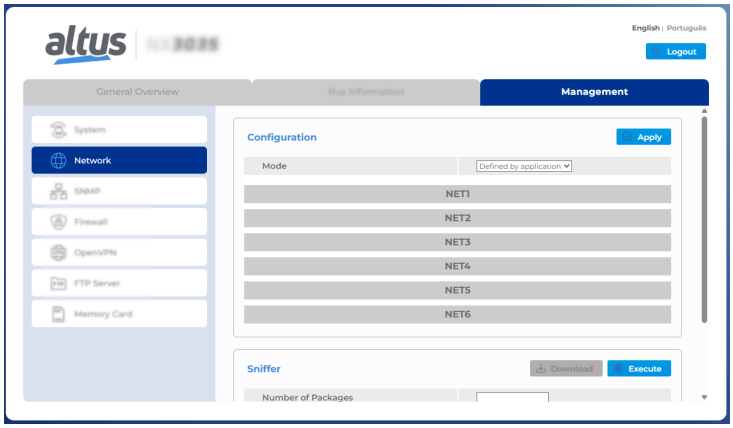


Figure 162: Interfaces Table - Application Mode

2. **Defined by web page**
For *Defined by Web Page* mode, the interface table remains enabled as shown in the figure below.
In this mode, the user can set the IP Address, Network Mask, and Gateway for each of the available Ethernet interfaces, as well as enable or disable it when more than one Ethernet interface is available.

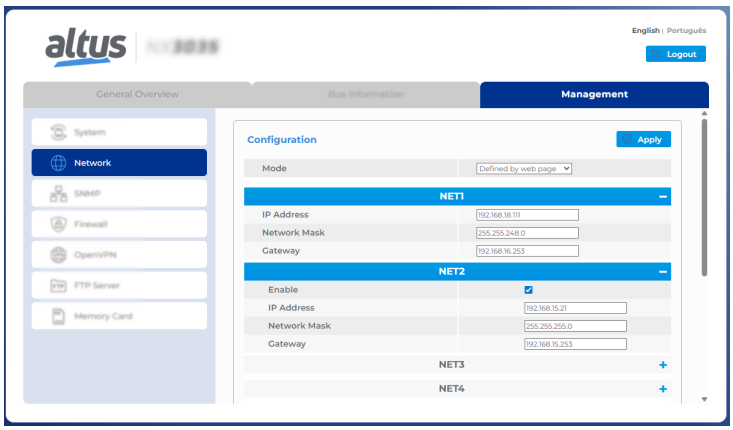


Figure 163: Interfaces Table - Web Mode

The *Enable* checkbox allows the user to enable and disable the Ethernet interfaces. This is only available to be checked or unchecked when the configuration is *Defined by web page*.
When this checkbox is unchecked, it indicates that the corresponding NET interface is disabled, meaning it will not receive configuration and will remain inactive. When the interface is enabled (with the checkbox checked), the settings become available for editing.
To have the settings applied to the controller, simply click the *Apply* button. This process checks if there were any errors in the configuration made and, if so, displays a message on the browser screen indicating the error. If the settings are correct, after clicking *Apply*, a confirmation window appears in the browser to apply the new settings. By clicking *OK*, the settings are sent to the controller and applied.

ATTENTION

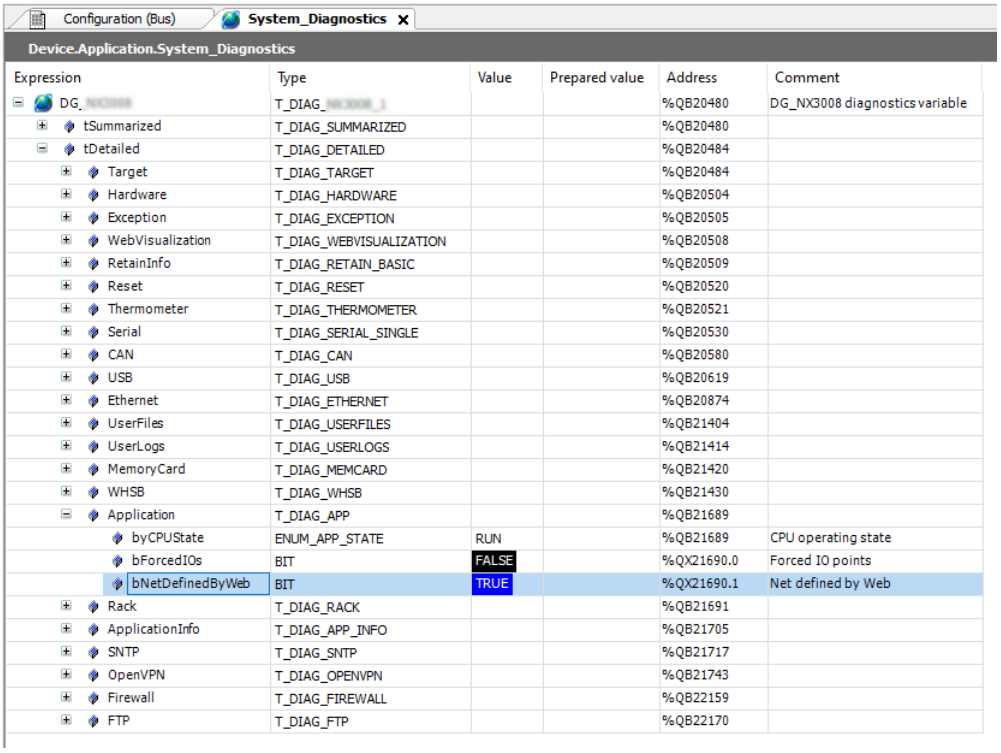
When making network changes in the controller, the interfaces will be restarted, which may cause a communication loss. Especially when changing the IP address value.

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

ATTENTION

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

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



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

Figure 164: Diagnostics - IP defined by the Web Page

5.14.3.2.2. Sniffer

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

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

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

Using the *Interfaces* table, you can select which interfaces you want the Sniffer to run on, i.e., perform network analysis. This screen displays the Ethernet interfaces specific to each controller. You can select all available interfaces and run them on all simultaneously. For disabled interfaces, it is not possible to run Sniffer. If the selected option is disabled, an error will be displayed in the browser.

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

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

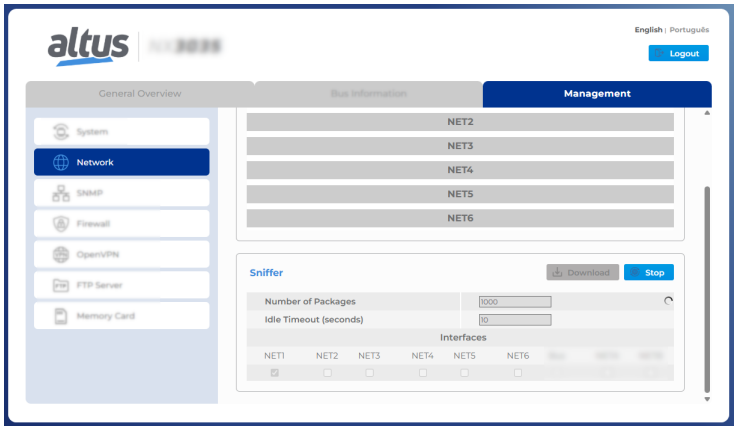


Figure 165: Network Sniffer Running

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

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

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

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

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

5.14.3.3. SNMP

SNMP (*Simple Network Management Protocol*) is a protocol widely used by network administrators to provide important information and diagnostic equipment present in a given Ethernet network.

This protocol uses the concept of agent and manager, in which the manager sends read requests or write certain objects to the agent. Through a MIB (*Management Information Base*) the manager is aware of existing objects in the agent, and thus can make requests of these objects, respecting the read permissions or writing the same.

MIB is a collection of information organized hierarchically with each object of this tree is called OID (*Object Identifier*). For all equipment with SNMP, it is mandatory to support MIB-II. In this MIB are described key information for managing Ethernet networks.

5.14.3.3.1. SNMP on Nexto CPUs

The Nexto Series controllers 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 the access, with support for the SNMPv1, SNMPv2c, and SNMPv3 protocols, as well as support for MIB-II, where objects are described in RFC-1213. The table below contains the available objects.

OID	Name	Description
1.3.6.1.2.1.1	System	Contains name, description, location and other equipment identification information.
1.3.6.1.2.1.2	Interfaces	Contains information of the machine's network interfaces.
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.

OID	Name	Description
1.3.6.1.2.1.7	UDP	Contains statistical connections using UDP protocol.
1.3.6.1.2.1.11	SNMP	Contains statistical connections using SNMP protocol.

Table 151: MIB II Objects – Nexto Series SNMP Agent

By default, the SNMP agent is enabled, meaning that the service is started when the controller boots up. Access to the agent information is provided through the integrated Ethernet interfaces on UDP port 161. Therefore, when the service is active, the agent information can be accessed through any of the integrated Ethernet interfaces, depending on the availability of the controller being used. The figure below shows an example of an SNMP manager reading some values.

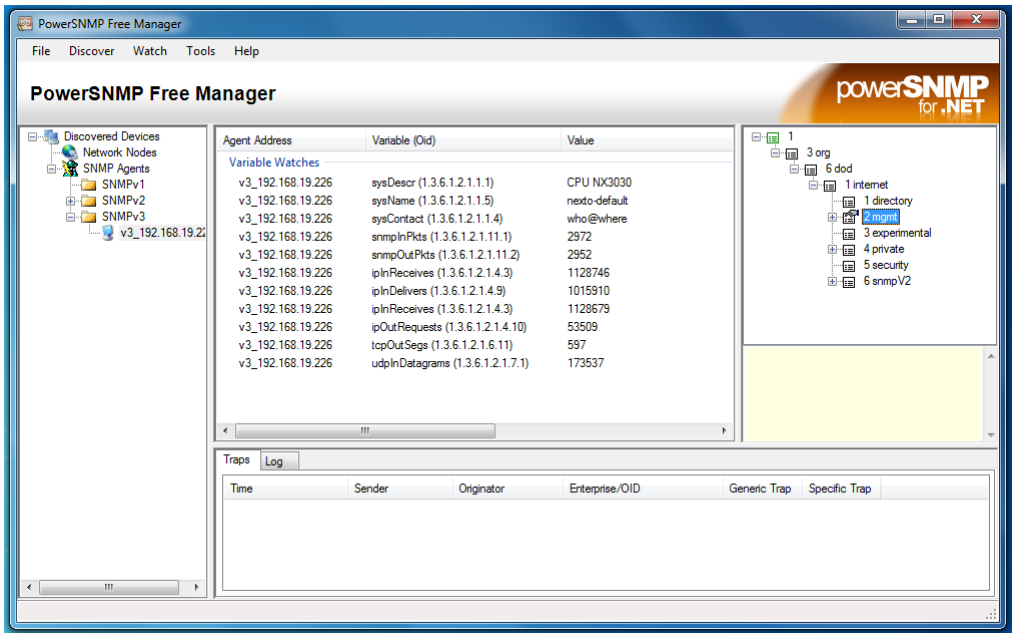


Figure 166: SNMP Manager Example

For SNMPv3, in which there is user authentication and password to requests via SNMP protocol, is provided a standard user described in the User and SNMP Communities section.

If you want to disable the service, change the SNMPv3 user or communities for SNMPv1 / v2c predefined, you must access the controller System Web Page. For details, see the SNMP Configuration section.

5.14.3.3.2. SNMP Configuration

SNMP settings can be changed through the System Web Page, in the *CPU Management* tab, under the *SNMP* menu. After successful login, the current state of the service (enabled or disabled) as well as the user information SNMPv3 and communities for SNMPv1 / v2c can be viewed.

The user can enable or disable the service via a checkbox at the top of the screen.

It's also possible to change the SNMPv3 information by clicking the *Change* button just below the user information. This action will open a form where the user must fill the old username and old password, also the new username and the new password. Any other information can not be changed.

To change the information from the SNMPv1/v2c *Communities* group data, the process is similar, just click the *Change* button below the group information. In opened screen, new data can be filled in the *rocommunity* and *rwcommunity* fields. If any field is left blank, its correspondent *community* will be disabled. If both fields are left blank, the access to the SNMP agent will only be possible through SNMPv3.

If the user wants to return to the default settings, it must be manually reconfigured according to the available information in the User and SNMP Communities section. Therefore, all current SNMP configurations will be kept in the firmware update process. These options are shown in figure below.

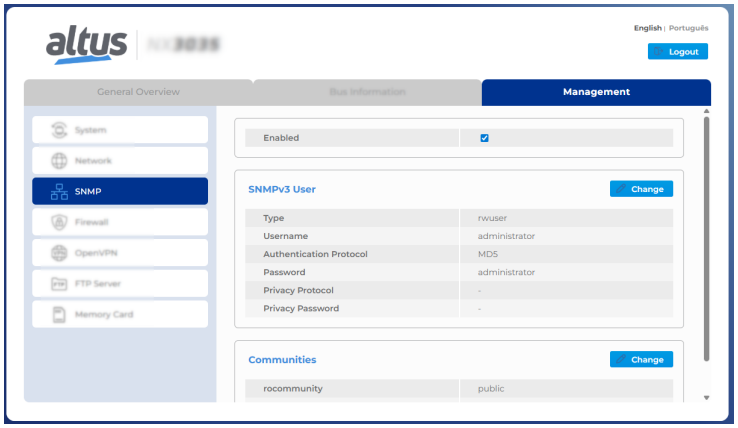


Figure 167: SNMP status configuration screen

5.14.3.3.3. User and SNMP Communities

Prior to access the SNMPv1 / v2c protocol informations, there are two *communities* settings to be configured, according to table below.

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

Table 152: SNMPv1/v2c Default Communities info

It's possible to access SNMPv3 using default user, see table below:

Username	Type	Authentication Protocol	Password	Privacy Protocol	Privacy Password
administrator	rwuser	MD5	administrator	-	-

Table 153: SNMPv3 Default User info

For *communities* settings, user and password, some limits must be respected, as described on the following table:

Configurable item	Minimum Size	Max Size	Allowed Characters
rocommunity	-	30	[0-9][a-z][A-Z]@$*_
rwcommunity	-	30	[0-9][a-z][A-Z]@$*_
V3 User	1	30	[0-9][a-z][A-Z]@$*_
V3 Password	8	30	[0-9][a-z][A-Z]@$*_

Table 154: SNMP settings limits

5.14.3.4. Firewall

The Firewall is designed to increase the security of the device while it is in use. The main function of the Firewall is to filter data packets coming into and leaving the device. The implemented filter uses information from each data packet to decide whether that packet is allowed. The main parameters used are the input/output interfaces, the port, the transport layer protocol, and the source and destination addresses.

ATTENTION

When the device is turned on with its button pressed, the Firewall is disabled and its rules are not applied.

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

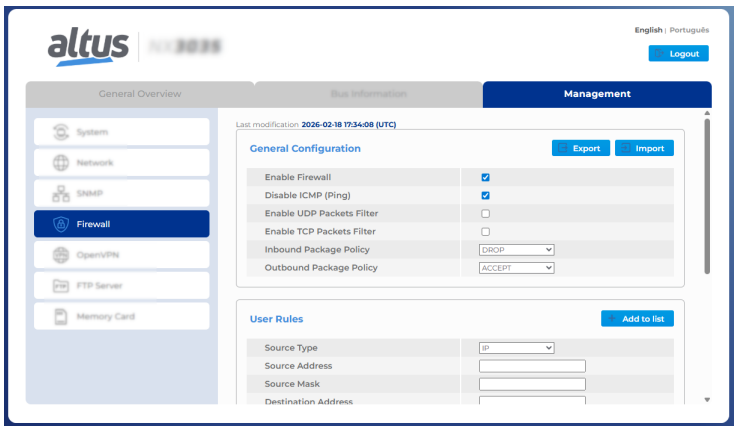


Figure 168: Firewall Configuration Screen

The Firewall is a separate feature from Mastertool, that is it doesn't require any interaction with Mastertool. Settings applied on the *Firewall* section take effect when confirmed with the *Apply* button, and are automatically saved in the controller. If the feature is enabled, it will operate again even after rebooting the device.

The following sections describe the possible settings for the Firewall, divided according to the tables of the *Firewall* section.

5.14.3.4.1. General Configuration

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

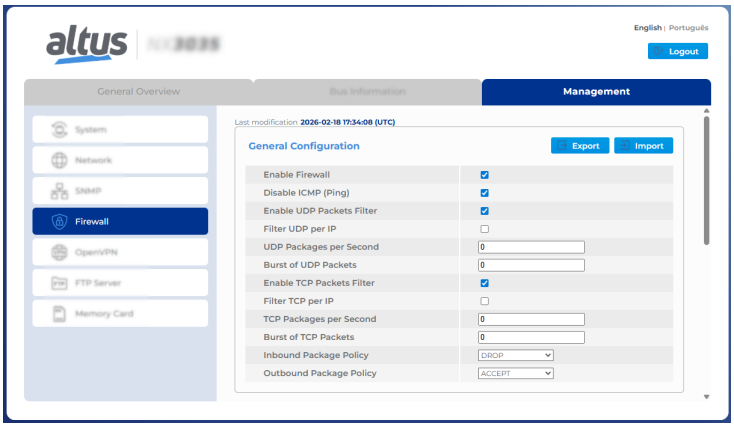


Figure 169: Firewall General Settings Table

This table expands dynamically by selecting the options to enable UDP and TCP packet filters, revealing all the items that can be configured. The first item in this table, *Enable Firewall*, is used to enable and disable this functionality. When the Firewall is enabled, the web page settings, when submitted to the device, will be applied to the configuration files, and then the Firewall will filter what has been configured. If the Firewall is disabled, the configuration that was made is stored, but the rules are not applied in the controller.

The field *Disable ICMP (Ping)* enables or disables protection against the ICMP protocol. When protection is enabled, the controller will not respond to *Ping* requests, since it will drop packets that use the ICMP protocol. When disabled, the operation of the device for *Ping* responses maintains its normal behavior.

When enabled, the fields that enable UDP and TCP packet filtering, filter these protocols according to the limits configured in their respective fields. The packet filtering rule works like this: for a packet to be accepted, there must be "credits" available, and one credit is used to accept a data packet.

The setting of the field *Burst of XXX Packets* sets the initial value of packages (credits), which will be accepted. In this way, it is possible to set an overflow limit for these packets, where if there is a large flow of packets, only the configured amount will be accepted. The *XXX Packages per Second* field sets how many credits that rule will earn per second. For example, if the value is 5, each second, the rule will receive five new credits, so it will be able to accept five more packages. The limitation for this increment in the number of credits is the configuration of *Burst of XXX Packets* itself, and the limit set here is not exceeded, even with the increment of packets every second. These settings are applied as a *stock*, where upon receiving a data packet, it is first checked if there is any credit available in the stock, and then a decision is made whether or not to accept the package. If the packet is accepted in this quantity filter, it is forwarded to the filter of the other firewall rules.

The setting *Filter XXX per IP* causes the rule to differentiate the source addresses of each packet and apply the packet per second and packet overflow filters individually to each IP address. So, going back to the previous example, it can be considered that each source address has its *stock* of credits, and one address cannot use the credits that are in the *stock* reserved for others.

ATTENTION

Negative values are not allowed for the *XXX Packages per Second* and *Burst of XXX Packets* fields. If negative values are set, when applying the settings an error message will appear on the screen indicating the field that had a conflict. If the filter is enabled, but the values in these fields are left at 0, the filter is not applied.

The settings in this table are applied with the *Apply* button that appears in figure 171.

The fields for selecting both incoming and outgoing policies have options to accept and drop. If the Firewall is active, when data packets arrive, all the rules that have been configured are checked, and then the policy configured for these packets is applied, whether *Accept* or *Drop*. So if an accept policy is set, *Accept*, all packets that do not match any configured rule will be accepted by the firewall, and if a reject policy is set, *Drop*, they would all be dropped.

At the top of the table, the *Export* and *Import* buttons are available. Using the *Export* button allows you to download a file with the extension *.firewall* containing all applied Firewall configurations. The *Import* button allows importing a *.firewall* file previously exported from this or another CPU of the same model, containing the Firewall configurations. When importing a file, the rules contained within it will be validated and configured in the *General Settings* fields and in the *User Rules* table. After importing, the rules will be applied by clicking the *Apply* button.

ATTENTION

The exported file contains only the rules that have been effectively applied to the device. Any configuration or rule added in the web page list but not applied to the CPU will not be included in the generated file.

ATTENTION

Any modification to the *.firewall* file, such as manual editing or changing the extension, may prevent some rules from being applied correctly. Always use files generated by the *Export* button of the same CPU, or from another of the same model, when importing Firewall configurations.

5.14.3.4.2. User Rules

The *User Rules* table was created to allow greater control over the firewall's rule settings. With it, you can configure different rules dynamically and with more precise filters.

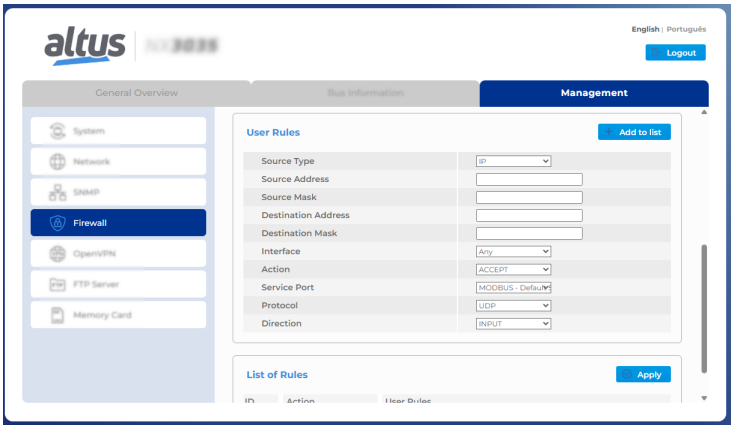


Figure 170: Firewall User Rules Configuration Table

This table changes its format according to the selected *Source Type*, which can be IP or MAC. When the type is *IP*, the table has the items shown in figure above, but when the type is selected as *MAC*, the source and destination mask fields disappear, as well as the *Destination Address* field. The item *Source Address* now accepts a MAC address as input in a format of six groups of two hexadecimal digits separated by colons, e.g. "1A:2B:3C:4D:5E:6F". Also, an address-based *MAC* rule can only be configured as an input rule. In other words, the *Direction* field will be forced with the value *INPUT*.

With the *Source Address* and *Destination Address* fields, it is possible to define the addresses that will be configured for that specific rule. By using the *Source Mask* and *Destination Mask* fields, it is possible to configure a network range for this rule. Defining an address together with a mask results in a group of IPs that will be assigned to the rule being configured. If only the address is configured, then only that specific address will be assigned to the rule. However, by using different network mask configurations, it is possible to obtain IP groups of different sizes that will be applied to the rule.

Interface configuration makes it possible to individually select each physical or virtual interface available to the controller. Based on which interface you select for a given rule, only data packets entering or leaving the interface will be filtered by the Firewall. If you use the option *Any*, this rule will have no interface filter. So the filtering rule will be valid for all available interfaces.

The *Action* field has three configuration options: *ACCEPT*, *DROP*, and *REJECT*. The action sets up what should be done with the package whose characteristics match the rule applied. If the chosen action is *ACCEPT*, the data packet having characteristics according to the rule will be accepted. If it is *DROP*, the packet will be dropped, and no reply will be sent to the sender of the package. Finally, if it is set to *REJECT*, the packet will be rejected, and a reply will be sent to the sender, stating that the requested *host* is inaccessible.

The *Service Port* field, is used to indicate which ports will be configured in this rule. All service ports that have a certain protocol or communication *standard* for the controller, such as the MODBUS protocol that has the standard port 502, are available with the service name and port used next to it. Thus, if you configure the rule for the MODBUS protocol, port 502 will be applied if you configure the rule for the WebVisu service, port 8080 will be applied, and so on for the other protocols listed in the checkbox.

This field also has two other settings, which are *Any* and *Other*. When you select the *Any* option, the rule is applied to all service ports except port 80 then two rules are created using the following port ranges: *1:79* and *81:65535*. If you select the *Other* option, a text box appears in which you can configure the port you want, except for port 80. To configure a port, you can type its number in the text box, but if you want to add more than a single port, you must use the *"&"* separator, and if you want to insert a range of ports, simply enter the start and end port using the separator *":"*.

Example of configuring ports 120, 144, and the range 1300 to 1450 in the same field: *120 & 144 & 1300:1450*.

This field doesn't accept values outside the range 1:65535, port 80, or port repeats.

The HTTP port 80 can only be set by selecting it from the list of known protocols and cannot be applied to the NET 1 interface. So if the HTTP protocol is chosen, the Interface field *NET 1* and *Any* won't be selectable.

In the *Protocol* field, you can select between UDP, TCP, and UDP|TCP protocols. If you select the UDP|TCP option, two rules will be created on the firewall, one for each transport protocol.

In the *Direction* field, you can select between *INPUT*, *OUTPUT*, and *INPUT|OUTPUT*. These options cause the rule to be applied to packets arriving at the device, option *INPUT*, or leaving it, option *OUTPUT*. If the joint option is configured, two rules will be created, one with each direction option.

The figure below demonstrates how a rule is applied:

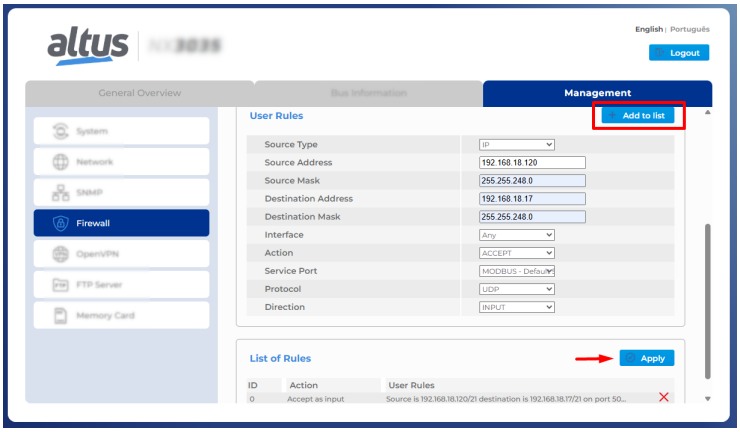


Figure 171: Firewall User Rules Enforcement Table

After filling in the fields as you wish to configure the firewall rule, you must click the *Add to list* button. By doing this, all the settings will be analyzed to check if there are invalid values or if there is any duplicate rule. It's impossible to add two rules with the same address, mask, interface, port, and direction parameters. If a conflict is found, a message will be displayed indicating the field that contains an invalid setting or the ID number of the rule in the table whose settings caused the conflict with the newly configured one.

After all, parameters are checked, the rule will be added to the list below the configuration table. This list expands automatically as rules are added or deleted. If you want to exclude a rule from the list, you can place the mouse over the one you want to exclude. When you do this, a red *X* button will appear on the right, as shown in the previous figure. By clicking it, the rule will be deleted from the table.

When adding new rules, or deleting an existing one, in the rules table, the *Apply* button below must be clicked for the configuration to be applied to the device.

ATTENTION

During the application of firewall rules, there may be a momentary instability in Ethernet communication.

5.14.3.4.3. Switch Mode or NIC Teaming Mode

Devices configured with Ethernet interfaces in Switch or NIC Teaming mode must always have user rules assigned to the first interface in the pair. This ensures that user rules are correctly applied to both interfaces that make up the Switch or NIC Teaming pair.

ATTENTION

User rules assigned to the second interface in the pair will be ignored.

5.14.3.4.4. Redundant architecture

In redundant architectures, the Firewall, as well as other settings configured through the system's web interface, is synchronized between the CPUs in the redundant pair, as described in Section General Characteristics of a Redundant CPU.

WARNING

When the Firewall is enabled, to ensure the exchange of keep-alive packets between the network interfaces, ACCEPT rules must be created for UDP ports 65404 and 65405 on the NET1 and NET2 interfaces, with the INPUT|OUTPUT direction. These rules must be configured individually for the MAC address of each Ethernet interface on each CPU in the redundant pair, ensuring the correct operation of this system.
The MAC addresses of the Ethernet ports can be obtained through the controller diagnostics, as described in Table 177.

5.14.3.5. OpenVPN

VPN (Virtual Private Network), used for surfing unsecured networks, transmitting data, or simply accessing the Internet with a high level of security and privacy. The VPN virtual network can be understood as a tunnel in which information travels securely, protected by security certificates and keys. OpenVPN is an *open-source* service, which means that it is free to use and distribute, and its source code is open for modifications if needed.

The main purpose of a VPN is to communicate securely over an unsecured network. To make this possible, data encryption is used based on certificates and keys generated using TLS, Transport Layer Security, a protocol that performs 256-bit encryption, one of the most secure.

To perform the configuration of the OpenVPN client or server, the OpenVPN page was created in the *Management* tab of the CPU's System Web Page. As shown in the figure below.

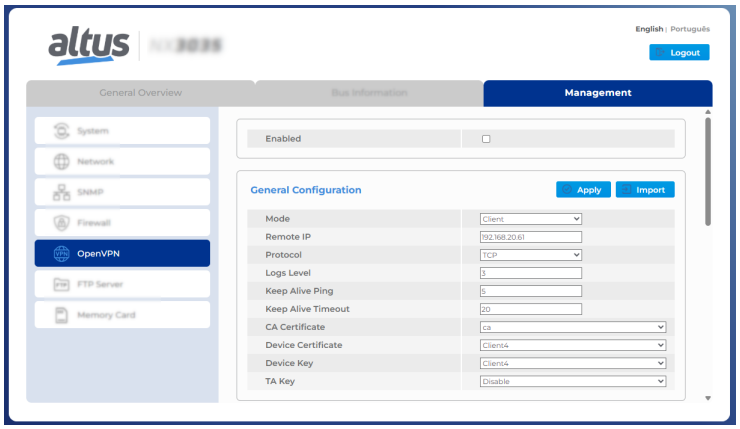


Figure 172: OpenVPN Configuration Screen

Because it is located within the *Management* tab, access to this page is password protected. The following sections describe the settings and functionality of this page.

- **Import Configuration**
  To quickly and easily configure the VPN on your device, you can use the *Import* button that appears in the picture 172 in the upper right corner of the page. Clicking on this button opens a file explorer window where you can select a configuration file. Files with extension *.conf* or *.ovpn* should be selected. When you select a file, its contents will be read and the configuration parameters present will fill their respective configuration fields on the web page.
  For the file's parameters to be interpreted correctly, they must follow standard OpenVPN configuration file syntax.
  If there are security files, certificates, or keys, written in the configuration file, along with the other parameters, they will be read and separated into separate files within the controller for use.

ATTENTION

Do not use spaces to separate the words in the name of the *".conf"* files. Instead, use *"_"* to separate them.

5.14.3.5.1. OpenVPN Configuration

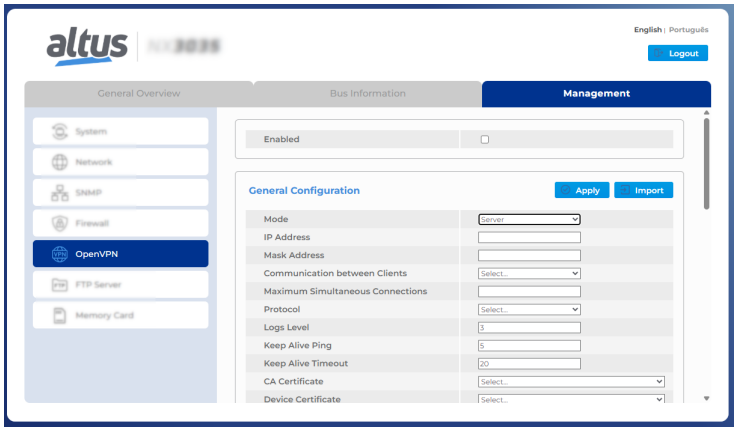


Figure 173: OpenVPN Server Configuration Table

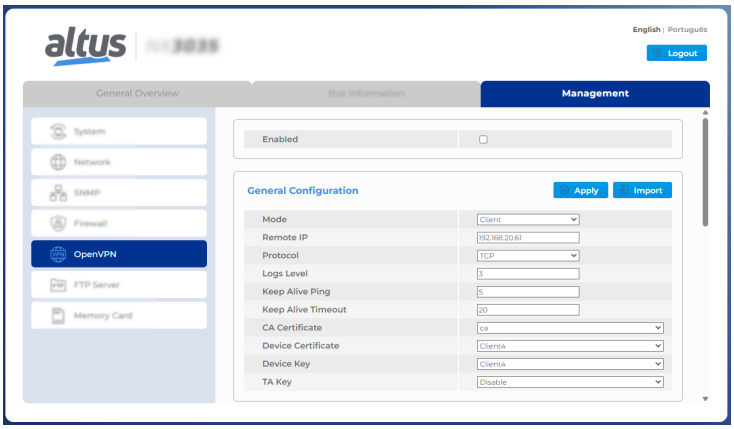


Figure 174: OpenVPN Client Configuration Table

This section shows how OpenVPN configuration is performed. The settings will be divided into three parts: settings common to both operating modes, settings unique to a server, and settings unique to a client.

- **Common Configurations**
  Looking at the figures with the client configurations, figure 174, and the server configuration, figure 173, you can identify that several parameters are the same for both configurations. These are:

  1. **Mode**
     With the configuration of the *Mode*, you can select between two options, client or server. When you select one of the two modes, the settings table changes automatically to allow the configuration of the necessary fields for each mode of operation.
  2. **Protocol**
     This field configures which transport protocol will be used for VPN communication. It can be set between UDP and TCP.

     ATTENTION

     The configuration of the server and all its clients must be the same. With a divergent configuration, OpenVPN is not able to perform communication.

3. **Logs Level**
   This field sets the level that the log file will receive. The setting ranges from 0 to 5, 0 being the most basic level and 5 being the most advanced.

   Level 0 only displays logs about some critical failure in OpenVPN and levels 4 and above are used for debugging as there is a lot of information being written to the log file. For normal operation, it is recommended to use value 3.

   This field only accepts numbers as input. You are not allowed to use letters or special characters.

4. **Keep Alive Ping**
   This field sets the time, *in seconds* when the *Ping* request will be forwarded. This request serves to verify the connection between the server and the clients.

   This parameter can be set on both the server and the OpenVPN clients, but if this parameter is set on the server, the clients will assume the server's value and not the value set on them. If the server doesn't have such a setting, each client assumes its setting normally. If you want to disable pinging between the server and the clients, set the value to 0.

   This field only accepts numbers as input. You are not allowed to use letters or special characters.

5. **Keep Alive Timeout**
   This field sets the time, *in seconds* when the timeout of the *Ping* request will occur. After the expiration of this time, without a response from the other VPN device, it will be considered disconnected.

   This parameter can be set on both the server and the OpenVPN clients, but if this parameter is set on the server, the clients will assume half of the server's value and not the value set on them. Clients receive half the amount to ensure that they are disconnected in case the server disconnects. If the Server does not have such a setting, each client assumes its setting normally. If you wish to disable this feature, set the value to 0.

   This field only accepts numbers as input. You are not allowed to use letters or special characters.

6. **Security Files**
   In the fields *CA Certificate*, *Device Certificate*, *Device Key* and *TA Key*, you must select which security file, certificate, or key, will be used to establish the OpenVPN communication. The options in each field, *combobox*, are filtered according to the type of key file or certificate, although there is no differentiation between keys and certificates.

   To be possible to select a file, it must first have been imported.

   All security files are required for correct communication to be established between clients and the VPN server, except for *TA Key*. This key is optional for communication, but if it is used on the server, it becomes mandatory for all clients on the server.

   See the TLS Key and Certificate Management section for further information about generating certificates and security keys based on TLS.

7. **TA Key**
   In the field *TA Key* it is set which type of encryption will be applied to the *TA Key*. This field stays hidden until you select a file for the TLS key because it is only used in conjunction with this key. The default value of this parameter is *SHA1*, but you can select from the following values: *SHA256*, *SHA512*, and *MD5*, in addition to the default SHA1.

   ATTENTION

   This configuration needs to be the same between the clients and the server in the same OpenVPN network. If the value of this field is different between the client and server, the connection will not be established.

- **Exclusive Server Configurations**
  The exclusive server configurations, seen in figure 173, are described below.

  1. **IP Address**
     The IP range that will be used to assign the server and client addresses for the VPN network is configured by the server by setting the *IP Address* and *Mask Address* fields. All IPs that will be assigned to the clients and the server will be taken from the specified range.

     The server's IP address is always the first available value in the configured range, and for IP assignments to clients, the values still available in the range are used, so the first available value is assigned as clients make their connection. For example, if a network is configured with the addresses 10.8.12.4 and mask 255.255.255.248, the server will assume IP 10.8.12.5 which is the first available address in the configured range. However, if mask 255.255.255.0 is set, the server will assume IP 10.8.12.1, which is the first available address in the range.

     The IP and Mask address fields only accept settings that have the syntax of an IP address and mask address, respectively. If anything out of the standard is configured, an alert message will be displayed, informing you that an error has occurred.

2. **Communication between Clients**
   In this field, you can enable or disable communication between clients in the VPN network. When the option is selected as *Disabled*, only client-server communication can be performed directly. If the option selected is *Enabled*, it will allow communication between the clients themselves in addition to the client-server communication.
3. **Maximum Simultaneous Connections**
   In this field, you can set the maximum number of clients that can connect to the server simultaneously. This field accepts only numeric characters, and the minimum value is 1.
4. **Private Networks**
   When you select OpenPVN's operating mode as a server, a table will be displayed, normally hidden, which allows the configuration of private networks that can be below the server and each client.



Figure 175: OpenVPN Private Network Configuration Table

To configure a private network that is below the server, simply select the network type as *Server* and configure the network addresses and mask. Configuring a private network for a client requires, in addition to setting the type as *Client*, to enter the *Common Name* of the client that owns the network being configured.

The Common Name of a client is set when generating the *Device Certificate*. This parameter is entered when creating the certificate and is unique for each client and server. The configuration of these private networks creates a routing table that will be checked when receiving or sending packets over the VPN.

Figure above, shows a configuration of a subnet *80* on the OpenVPN server, then a routing rule will be configured that will forward the data packets that will be received by the VPN to the device interface configured on this network. It also creates a rule, internal to the server, that if a data packet has the subnet *70*, this packet will be routed and forwarded through the VPN tunnel. The same behavior occurs with the *client2* client, but with the subnets switched, because below this client is the *70* subnet and it will forward packets with the *80* subnet to the VPN tunnel.

See the following figure for an example of architecture:

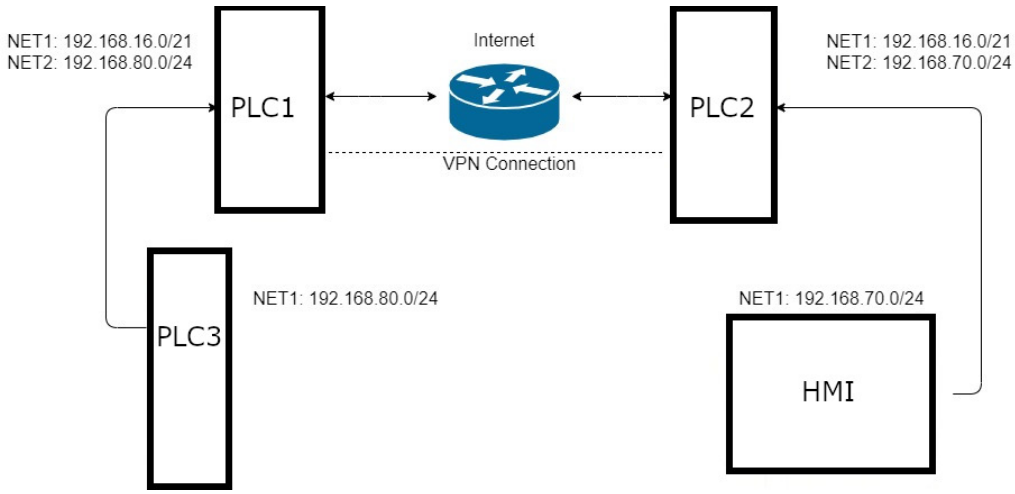


Figure 176: Architecture Example with Private Networks

In Figure 176, the PLC1 on the left has a private network 80 configured on its NET 2, and connected to it is the PLC3 on the same network. The PLC2 on the right has a private network 70 configured on its NET 2 and connected to it is an HMI, on the same network. The example architecture realizes the communication between the PLC3 and HMI devices over VPN by configuring their respective private networks.

For the settings present in the table to be applied to the device you must click the *Apply* button and confirm the operation in the confirmation window that will appear. When the rules are applied, a message will be displayed indicating whether the operation was successful or not.

- **Exclusive Client Configurations**
  There is only one configuration unique to OpenVPN clients on the page, which you can see in the picture 174. This configuration is the *IP Remote*.

  1. **Remote IP**
     The Remote IP field sets the address where the VPN server is expecting communication from the clients. If an OpenVPN server is established on a computer, the remote IP configuration must be done according to the IP address of this computer. This field also accepts *host names* as the remote address, so you can set an IP or a hostname in this parameter.

     ATTENTION

     Because of the need to allow for such different parameters, IPs, and host names, the only check that exists in this field is whether or not data exists. Be careful when performing the configuration.

  2. **Application Settings**
     To enable the functionality, the checkbox *Enabled*, shown in the figure 174, must be checked. If you just want to apply the settings you have made and not enable OpenVPN, uncheck this checkbox.

     After you have made all the desired settings, the settings must be applied to the device, to do this use the *Apply* button. This button is shown in the figure 174. When the settings are applied and the VPN is enabled, the web page will perform an automatic *scroll* to the OpenVPN *status* table, displayed in the Status section.

5.14.3.5.2. Security Files

Security files are used to establish OpenVPN's communication securely by performing the role of encrypting and decrypting the data packets that will travel through the VPN tunnel. In the TLS Key and Certificate Management section, it is described how to generate TLS keys and certificates. Here is a screenshot that shows the section responsible for managing the security files:

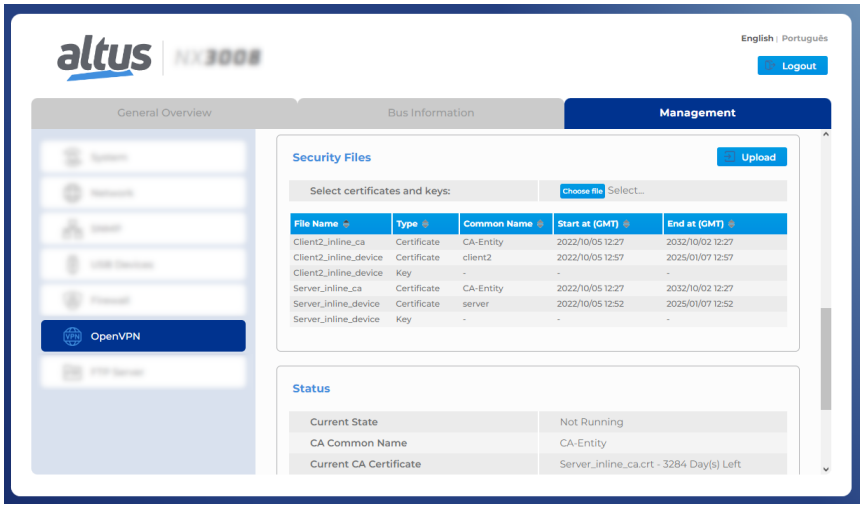


Figure 177: OpenVPN Security Files Table

In this section of the System Web Page, you can manage the security files. You can import files, monitor the validity of certificates, download files uploaded to the device, and delete files that have been uploaded.

By clicking the *Choose files* button, you can import certificates and keys, these files must have the respective extensions *.crt* and *.key*. This button opens a file explorer window and allows you to select one or more files.

ATTENTION

There is a limit of 12 files that can be imported into the controller.

The control of the files is done in the table, which is shown in the picture above. This table adds new items, or removes them, as the import or delete operations occur. You can identify whether the file is a key or a certificate by the second item in the list, the *Type*, which indicates what that file is. For the certificates, their *commons names* and their expiry dates, both start and expiry, are also displayed.

You can recover a file that has been imported into the part and also delete it. When you drag the mouse over a file in the table, two buttons appear, one for downloading and one for deleting.

5.14.3.5.3. Status

Designed to allow for data monitoring, OpenVPN's status table automatically expands as you change settings and displays various data about the connection such as the state of the VPN, the VPN IP assigned to that device, the data being transmitted, and the security files being used for communication.

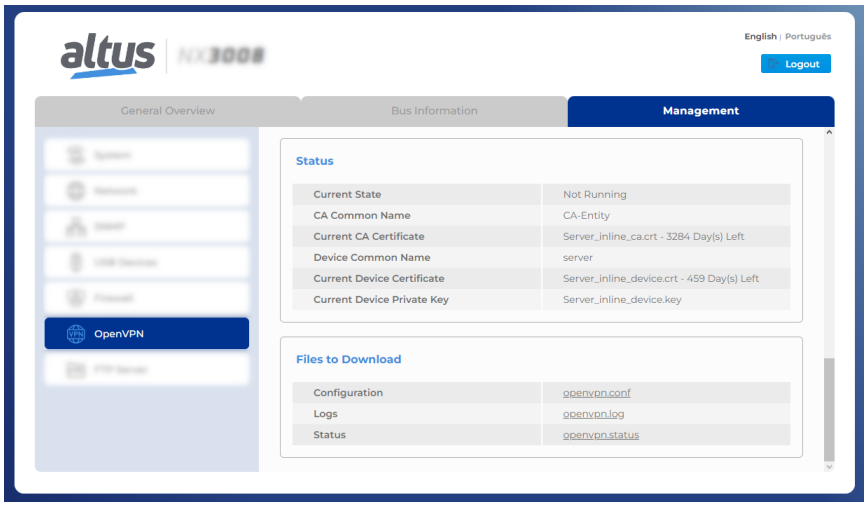


Figure 178: OpenVPN Status Table with Feature Disabled

When VPN is disabled, the table has few parameters. The field *Current State* indicates whether the VPN is enabled or not, and the other fields show which certificates and keys are configured for VPN communication. If one of the security files has not been selected, the character "-" will appear instead of its name, indicating that there is no file configured.

The common name fields for the CA and the device display the names given to the respective certificates, certificate authority, and device.

Next to the file name of each certificate is displayed the remaining time, *in days*, until its expiration date.

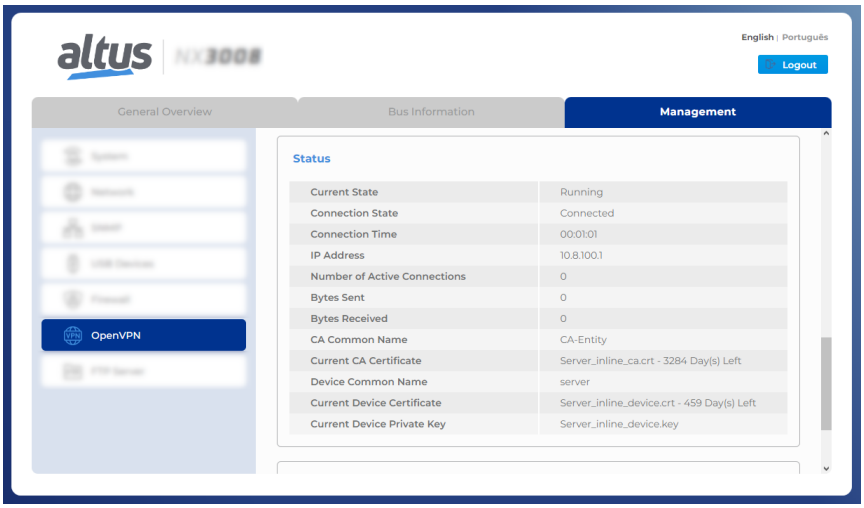


Figure 179: OpenVPN Status Table with Feature Enabled

When the functionality is enabled and the settings are applied on the device, the table has its cells dynamically modified so that the remaining information is displayed. Information about the OpenVPN connection status can be found in the first two topics of the list.

The item *Current State* has the states of *Not Running*, *Starting service...*, and *Running*, which indicate respectively that the VPN is disabled, is starting or is enabled.

The item *Connection State* has the states *Not connected*, *Connecting...*, and *Connected*.

The other information that can be obtained from this table is the total connection time, the device's IP address, and the amount of data sent and received, in bytes. The status of how many clients are currently connected is only displayed when OpenVPN is operating as a server.

5.14.3.5.4. Files to Download

You can check the information generated by OpenVPN through status and log files. The list of files to download is only displayed when there is a file to download. If there is none, the message "No file found in the controller!" is displayed. Clicking on any of the links will download the requested file through the browser.

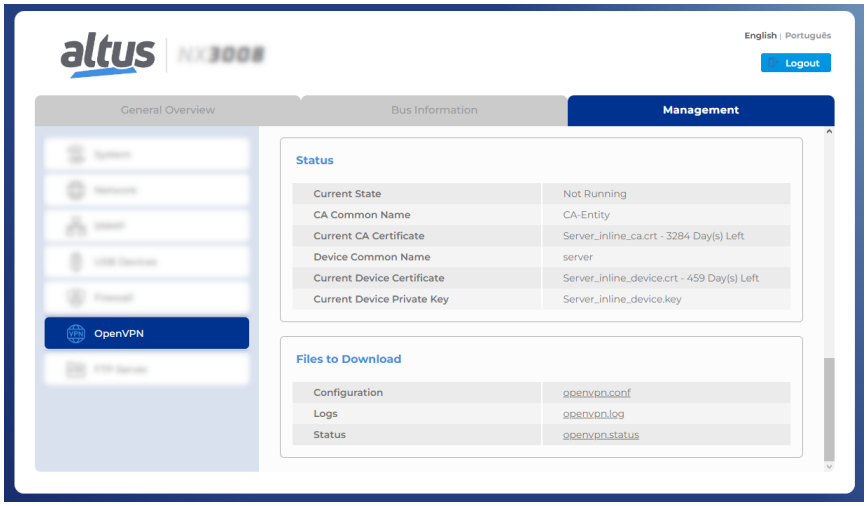


Figure 180: OpenVPN Download Section

5.14.3.5.5. *Architectures Configuration*

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

1. **Host-to-Host**



Figure 181: Example of Host-to-Host architecture

This topology allows the connection between two VPN hosts. Both hosts can be chosen to be configured as the server, then the other should be configured as the client, or both hosts can be configured as clients and have a third host that will be the server for the VPN network.
Setting up this type of architecture doesn't require any specific configuration. In other words, there is no restriction on the settings available on the OpenVPN web page.

2. **Host-to-Site**

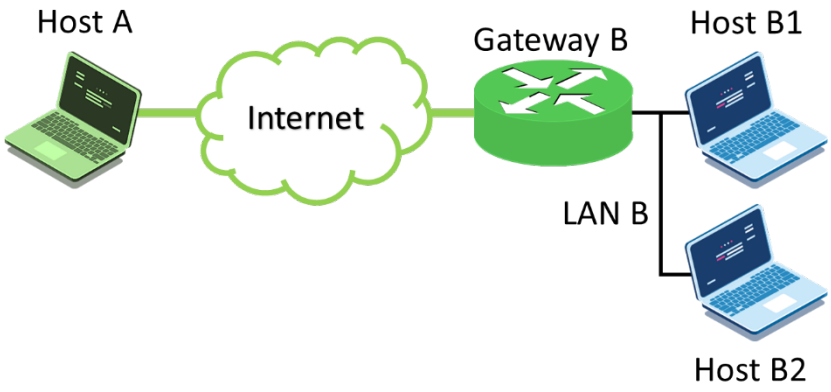


Figure 182: Example of Host-to-Site architecture

This topology allows the connection between two VPN hosts, but one of these hosts also acts as a gateway to the VPN network. Through this gateway, routing is performed to set up communication between hosts A, B1, B2, and Gateway B. In this scenario, either Host A or Gateway B can be the server. When one is the server on the network, the other will be the client.
The hosts, B1 and B2, that are on a private Lan B network below Gateway B, don't need to support OpenVPN to be able to communicate since all communication is handled by the VPN network gateway.
To enable communication between all devices on the network, you need to create routing rules for the VPN tunnel. Please refer to the section OpenVPN Configuration to see how to create private network rules.
This VPN connection architecture requires some specific configurations. The server must have its topology configuration as a Subnet, this being the default configuration of the controller, to configure the private networks under Gateway B, as seen in the image above.
You also need to enter the address of the private network, Lan B, that will be communicating through the VPN. This configuration is done using the command *push "route Lan_B_IP Lan_B's_Mask"* and is required regardless of whether the private network is located below the client or the OpenVPN server, but if the private network is below the VPN client, you must add, in addition to this command, the following configuration: *route Lan_B_IP Lan_B's_Mask*. The *route* command informs the server about the private network connected to the VPN network, and the command using *push* ensures that the clients of that VPN server receive the same information. These settings are written to the VPN server's configuration file.

3. **Site-to-Site**

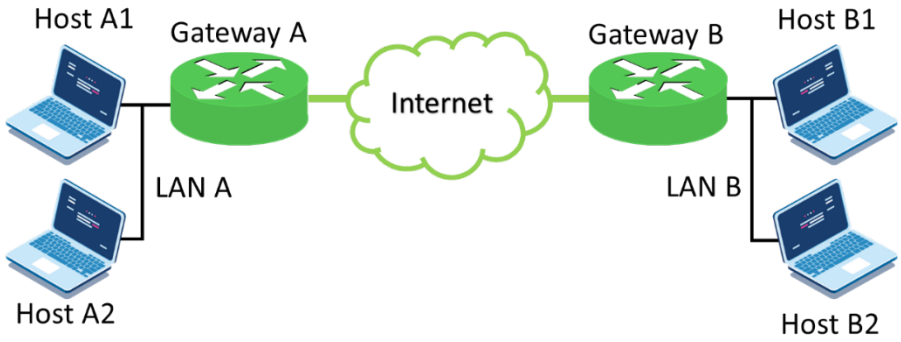


Figure 183: Example of Site-to-Site architecture

This topology allows the connection between two VPN hosts, both of which acts as a gateway to the VPN network. Through these gateways, access is provided to establish communication between hosts A1, A2, B1, B2, Gateway A, and Gateway B. In this scenario, any gateway can assume the role of a server, so the other will be the client.
None of the hosts that are in a private network below one of the two gateways need to support OpenVPN to be able to communicate, since all communication is handled by the VPN network gateways.
To enable communication between all devices on the network, you need to create routing rules for the VPN tunnel. Please refer to the section OpenVPN Configuration to see how to create private network rules.
The configurations for this architecture need the same specific settings described in section Architectures Configuration, with the difference that now, there are two private networks, and both must follow the configuration that has been demonstrated. Assuming that Gateway A is the server on this connection, you should add the following commands to the configuration file: *push "route Lan_A_IP Lan_A's_Mask"*, *route Lan_B_IP Lan_B's_Mask*, and *push "route Lan_B_IP Lan_B's_Mask"*. If the server is Gateway B, in the configuration file it would be added: *push "route Lan_B_IP Lan_B's_Mask"*, *route Lan_A_IP Lan_A's_Mask*, and *push "route Lan_A_IP Lan_A's_Mask"*.

5.14.3.6. FTP Server

FTP (File Transfer Protocol) is a protocol that allows files to be transferred between devices. It operates in the client-server mode, where the CPU becomes an FTP Server, storing files that FTP Clients can access for transfer, download, and upload.

The controlling FTP connection is a TCP connection established through port 21 of the FTP Server. Through port 21, the Client and Server exchange commands and responses to manage the file transfer session.

Through the FTP protocol, the FTP Client can read and write files that are stored either in the internal memory of the CPU or in external memory (memory card, for example) if present in the architecture. The maximum file size that can be transferred varies according to the amount of memory available between internal and external memory. When the memory limit is reached, the transfer will stop, and if the file has not been transferred completely, it will become a corrupted file.

ATTENTION

Downloading and uploading large files through FTP, both from internal and external memory, can affect the performance of the CPU considerably.

ATTENTION

The FTP server does not support communication through Windows File Explorer, so an FTP Client software is required to access it.

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

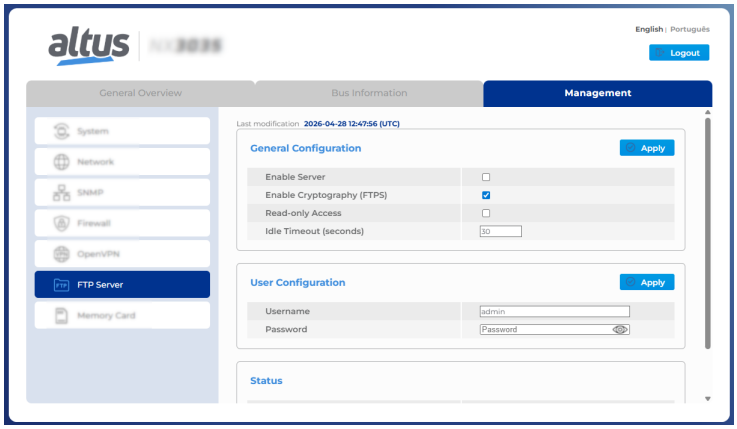


Figure 184: FTP Server Configuration Screen

FTP is a separate feature from Mastertool, meaning it does not require any interaction with Mastertool. The settings applied on the *FTP Server* section take effect when confirmed through the *Apply* button and are automatically saved in the controller. As long as the functionality is enabled, it will resume operation even after the device is restarted.

ATTENTION

For proper FTP operation, FTP clients must operate in Active mode. Passive mode is not supported.

5.14.3.6.1. General Configuration

1. **Enable Server**
   Allows to activate or deactivate the functionality. When the FTP server is enabled, the settings made through the System Web Page are applied to the configuration files. That means the server is available according to the configuration made. If FTP is disabled, the configuration is still stored, but the service cannot be used.
2. **Enable Cryptography (FTPS)**
   Enables and disables encrypted communication. This communication is done through Explicit FTPS, also known as FTPES. This is a secure extension of the FTP protocol, which adds a layer of encryption to the transfer.
   In this type of communication, when a client-server connection is established, instead of immediately starting the data transfer, the FTP Client sends an AUTH SSL command to request a secure connection. Then, when enabling encrypted communication, the CPU generates a self-signed certificate to guarantee communication using the SSL (Secure Sockets Layer) protocol.
   When the FTP Client performs the authenticated connection, the certificate and its respective information will be displayed, allowing Explicit FTPS communication (FTP over SSL) to be established through port 21.

   ATTENTION

   The FTPES certificate is valid for one year from the date it was created. To generate a new certificate, simply reapply the configuration.

3. **Read-only Access**
   Enables and disables limiting write and read access to the Server. By default, the parameter is enabled.
   When the parameter is enabled, the FTP Client has read-only access without the possibility to add or remove any files from the FTP Server. In this case, the only operation allowed is uploading files, that is, transferring them from the Server to the Client. When the parameter is disabled, all operations can be performed.
4. **Idle Timeout (seconds)**
   Sets the maximum time the connection will be maintained before the FTP Server closes it due to inactivity. In other words, once the connection to an FTP Client is opened, if there is no activity after the Idle Timeout, the connection to the Client is closed by the Server.
   The parameter can be configured with values between 10 to 60 seconds, the default being 60 seconds.

ATTENTION

The configured server idle timeout must be equal or shorter than the client's timeout.

5.14.3.6.2. User Configuration

1. **Username**
   User or *Username* required for the Client to connect to the Server.
   If a new configuration is made, the previous one is removed. In other words, there is only one FTP user, but this one can be used for multiple connections.
2. **Password**
   Manages the authentication key so the FTP Client can connect to the FTP Server.
   There is no password recovery so, if the password has been lost, it is necessary to add a new user configuration.

Configurable Item	Minimum Size	Maximum Size	Allowed Characters	Default
Username	4	30	[a-z][A-Z][0-9]@$*_	admin
Password	4	30	[a-z][A-Z][0-9]@$*_	admin

Table 155: FTP User Settings

5.14.3.6.3. Status

1. **Current State**
   Displays the current status of the FTP Server. The possible states are *"Running"*, *"Not Running"*, and *"Restarting Service"*. Each configuration applied will restart the service.
2. **Active Connections**
   Shows the user the current number of active connections.
   The FTP Server accepts a maximum of two active connections simultaneously. Some FTP Clients, such as *Filezilla*, use a feature called Multithread File Transfer to improve the efficiency and speed of the transfer. This feature allows the FTP Client to open more than one connection for the transfer of a file. Consequently, when using an FTP Client of this type, just one Client already counts as two active connections on the Server.
   Other FTP Clients, such as the command terminal, only use one active connection, allowing two FTP Clients to connect to the Server simultaneously.

5.14.3.7. Memory Card

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

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. The card presence is not displayed. If the user decides to format the files, the CPU takes a few minutes to execute the operation, depending on the cycle time (execution) of the application which is running in the CPU. Once the memory card is mounted, the CPU will read its general information, leaving access to the memory card slower in the first few minutes. This procedure is done only when the card is inserted, in case of the CPU reset or through the device's web page.

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.

A web page was developed for managing the memory card. Among the features offered are formatting, the option to unmount and remove the card, and the ability to enable and disable the card interface. These settings were developed in the "Memory Card" section of the CPU webpage, under the "Management" tab. In addition to the settings, information about the device's current status, total and free storage space, both measured in *kB*, are also displayed. The image below shows the home page, with the interface enabled and no devices connected.

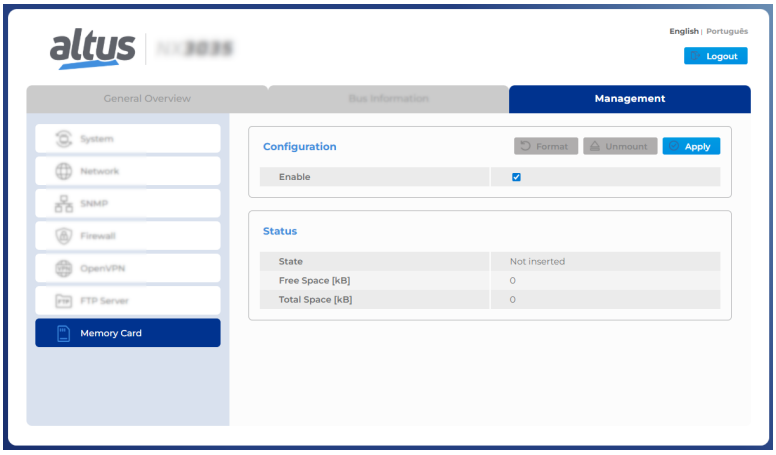


Figure 185: Memory Card Home Page

As long as there is no memory card inserted in the CPU, the *Format* and *Unmount* buttons remain locked for use. The *Status* table indicates the *"Not Inserted"* state. When a memory card is inserted into the CPU, the *Format* button is enabled for use. After the card is mounted, the *Unmount* button also becomes available for use. When inserting a memory card into the CPU, it may take a few moments for the card to be mounted and the information updated on the web page. The image below shows the web page when a card is connected and mounted in the CPU.

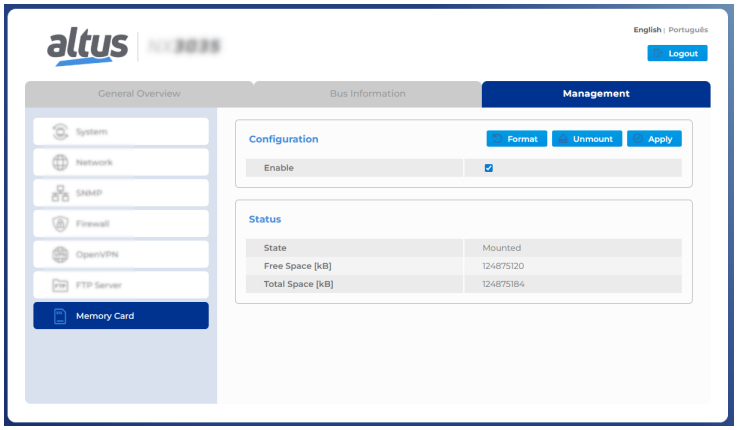


Figure 186: Memory Card Device Mounted

ATTENTION

For the memory card to properly mount, there must be at least *1Kb* of free space available on the CPU. If the card is inserted and there is not enough free space, the card will not mount and the web page will display the message *"Unmounted"*

5.14.3.7.1. Configuration

1. **Format Not Supported**
   If the device has a file system that is not supported by the CPU, the state *"Format not supported"* will appear when the device is inserted, as shown in the figure below. In this case, the device cannot be used until it is formatted with a supported file system.

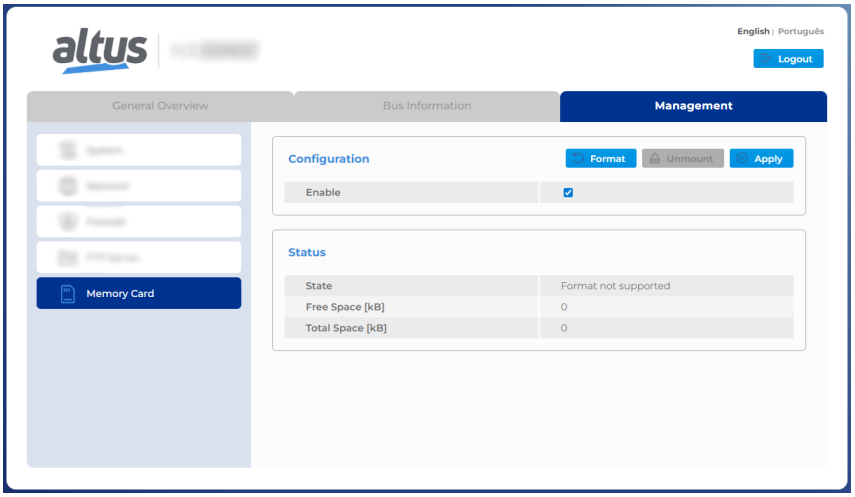


Figure 187: Format Not Supported Message

   To format the device, the *Format* button may be used, as shown as shown below.
2. **Formatar**
   To format the device, use the *Format* button. When you click, a *pop-up* style message will appear, asking you to confirm the operation. The image below presents this message.

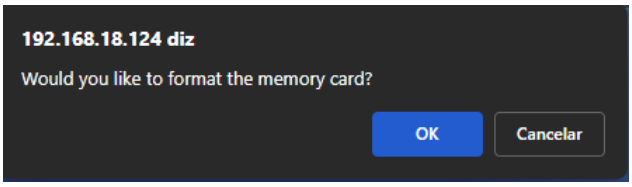


Figure 188: Format Confirmation Message

   Upon confirming with the *OK* button, the operation begins, after which all configurations are blocked. The *Format*, *Unmount* and *Apply* buttons, as well as the checkbox, become unavailable during formatting. The formatting process is indicated in the *Status* table, with the *State* value changed to *"Formatting..."*, as shown in the figure below.

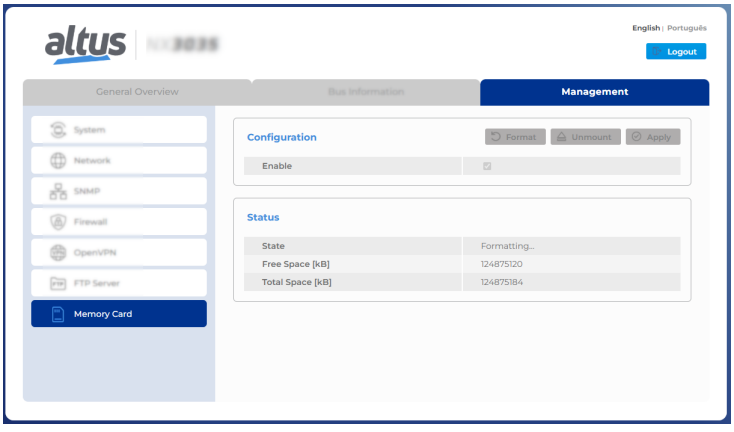


Figure 189: Formatting Memory Card

At the end of the formatting process, a message is displayed indicating that the operation has been completed on the device. The following figure shows this message.

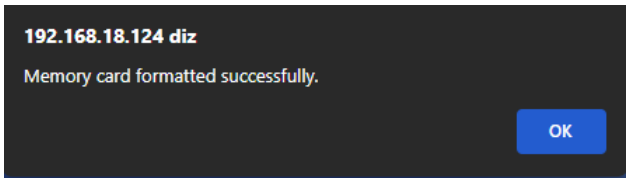


Figure 190: Formatting Complete Message

Upon completion of the operation, the web page returns to its initial state, unlocking all buttons, as well as the checkbox, as shown in figure Memory Card Device Mounted.

3. **Unmount**
To unmount and remove the device, click the *Unmount* button. A confirmation *pop-up* will appear asking you to confirm the operation. The image below shows this message.

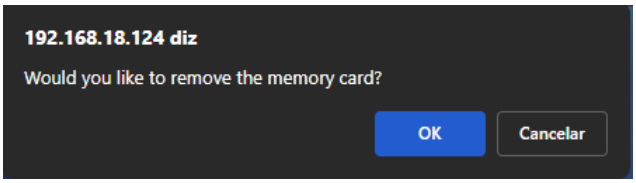


Figure 191: Confirmation Message to Unmount

Upon confirming with the *OK* button, the operation begins, after which all configurations are blocked. The *Format*, *Unmount* and *Apply* buttons, as well as the checkbox, become unavailable during unmounting operation. The unmounting process is indicated in the *Status* table, with the *State* value changed to *"Unmounting..."*, as shown in the figure below.

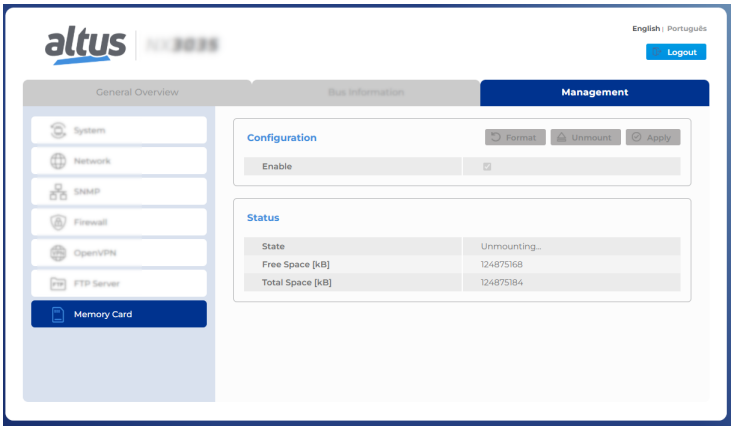


Figure 192: Unmounting Memory Card

At the end of the unmounting process, a message is displayed indicating that the operation on the device has been completed. The following figure shows this message.

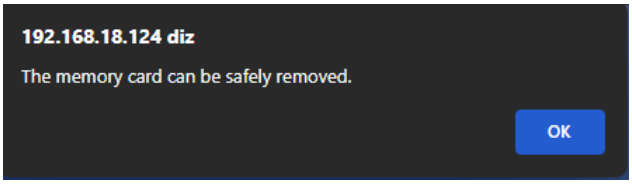


Figure 193: Unmount Complete Message

Upon completion of the operation, the *Format* and *Apply* buttons, as well as the checkbox, become available for use. The *Unmount* button remains locked because the card has already been unmounted. The *Status* table displays the data for an unmounted card, as can be seen in the image below:

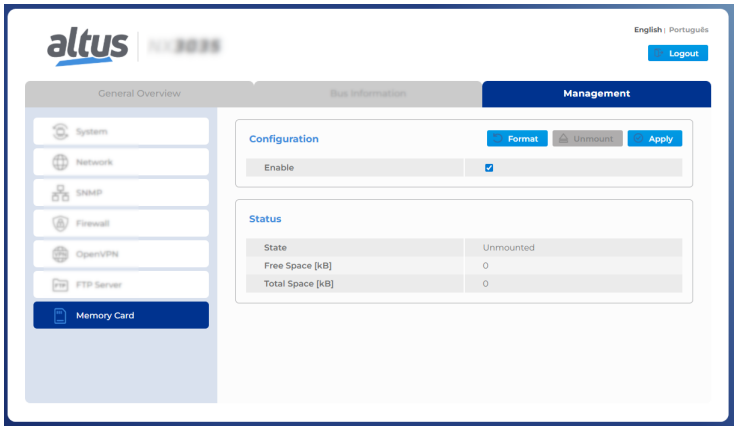


Figure 194: Memory Card Unmounted

4. **Enable**
A setting was developed on the card's web page to enable and disable the memory card interface, this functionality is part of the level one cybersecurity requirements according to IEC 62443. To enable, check the *Enable* checkbox in the *Configuration* table. Then use the *Apply* button to submit the new configuration. To disable, uncheck the *Enable* checkbox and use the same button to apply the setting. Clicking the *Apply* button will display a *pop-up* message asking you to confirm the operation. The image below shows the message.

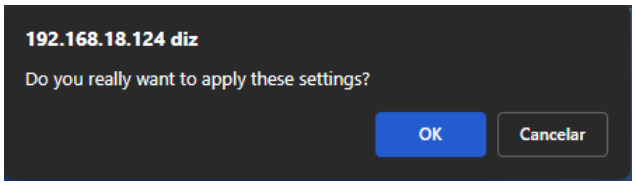


Figure 195: Confirmation Message to Apply Configuration

After confirming by clicking the *OK* button, the new configuration is sent to the CPU. If the interface has been enabled, the information displayed on the web page depends on whether or not a memory card is inserted for mounting. If there is no device, the page will display the information shown in figure Memory Card Home Page. When a device is connected, the information displayed will be as shown in figure Memory Card Device Mounted, after the card has been properly mounted.
If the new configuration disables the interface, the information displayed will be as shown in the image below.

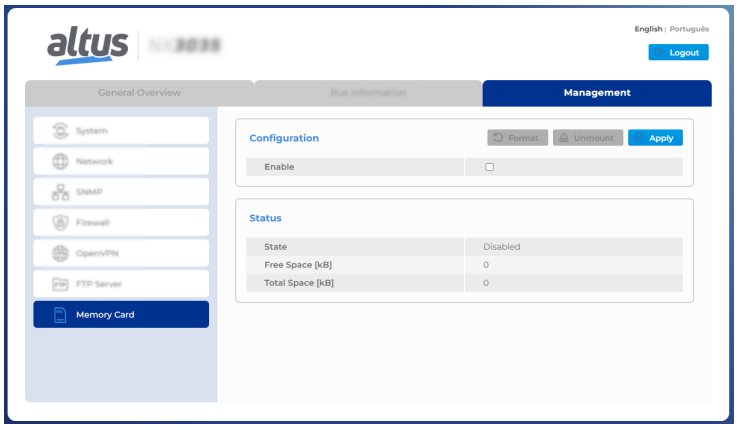


Figure 196: Memory Card Disabled Interface

ATTENTION

If the memory card deactivation is in effect, the MemoryCard folder will not be mounted.

5. **Memory Card Interface Management via Application**
To facilitate the management of the memory card interface, a function was developed that can be called directly by the user's application code. The *SetMemCardState* function was implemented within the *NextoStandard* library. The image below shows the library information, as presented in the *Library Manager*.

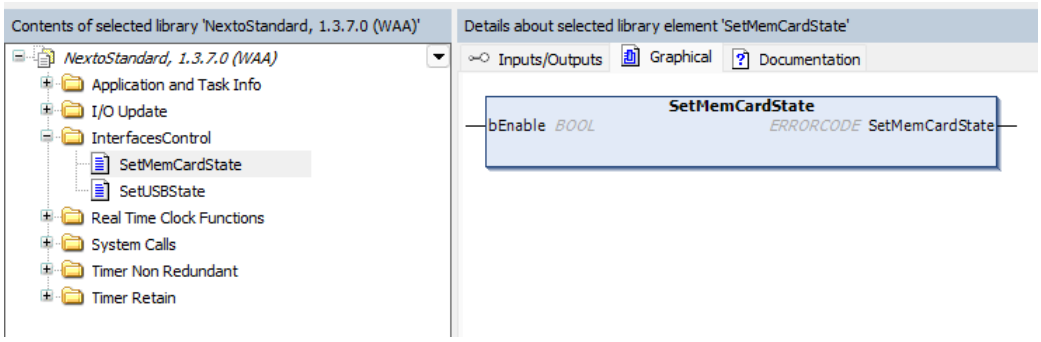


Figure 197: SetMemCardState information in Library Manager

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

```
PROGRAM UserPrg
VAR
  bSetMemoryCardInterfaceState : BOOL;
  stErrorCode : NextoStandard.ERRORCODE;
END_VAR
-------------------------------------------------------------------------
// Function call example to configurate memory card's interface
stErrorCode:= NextoStandard.SetMemCardState(bEnable :=
   bSetMemoryCardInterfaceState);
```

ATTENTION

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

6. Redundancy with NX3035 CPU

6.1. Introduction

This is a hot-standby redundancy, where controllers are duplicated. One of the controllers is usually in Active state and controlling the process, while the other controller is in Standby state, staying synchronized with the Active controller. If a failure occurs in the Active controller that prevents it from continuing to control the process, the Standby controller automatically switches to Active within a sufficiently low time to avoid process disruption, without causing discontinuities in the outputs controlling the process.

Hot-standby redundancy is a method used to increase fault tolerance and, consequently, enhance the availability of the automation system. The basic idea is that no single failure in duplicated components should cause a process control interruption.

Hot-standby redundancy is widely applied in:

- Oil exploration platforms;
- Power generation and distribution systems;
- Continuous processes, such as chemical plants, oil refineries, pulp production, etc.

In addition to the controllers, field networks (PROFIBUS DP and Ethernet), supervisory Ethernet networks, and control Ethernet HSDN (High-Speed Deterministic Network) can optionally be duplicated. By choosing to duplicate these networks, even higher availability can be achieved.

Hot-standby redundancy of Nexto Series CPUs does not provide for I/O module duplication. If I/O module redundancy is desirable, it can be handled at the application level by the end user. For example, the user can duplicate or even triplicate an analog input module and create a voting algorithm to determine which of the inputs will be considered at a given time in their application.

The figure below shows a typical example of a redundant architecture with the NX3035 CPU.

The central part of a redundant CPU is composed of two identical backplane racks, called PLCA and PLCB. In the context of redundancy, each backplane rack (PLCA and PLCB) is referred to as a half-cluster, while the set formed by these two backplane racks is called a cluster. The NX3035 CPU of the PLCA is referred to as CPU A, while the NX3035 CPU of the PLCB is referred to as CPU B.

Two synchronization channels connect the PLCA and PLCB via fiber optics through the NETA and NETB ports of the NX3035 (CPU A and CPU B) and are used for redundancy synchronization between the two half-clusters.

ATTENTION

The PLC configuration performed through the product's web page is not synchronized between the two controllers. Therefore, if any changes are made to the PLC configuration via the web page, these changes must be manually replicated on the other controller.

In this example, there are several additional networks:

- A redundant PROFIBUS field network (PROFIBUS 1A and PROFIBUS 1B).
- A redundant Ethernet supervisory network connected to the NET1 and NET2 ports of the NX3035.
- A redundant Ethernet network for the HSDN connected to the NET3 and NET4 ports of the NX3035.
- A "Redundant Ethernet 1" network for other uses connected to the NET5 and NET6 ports of the NX3035.
- A "Redundant Ethernet 2" network for other uses connected to the first two NX5000s of each half-cluster.
- A "Non-redundant Ethernet 3" network for other uses connected to the third NX5000 of each half-cluster.

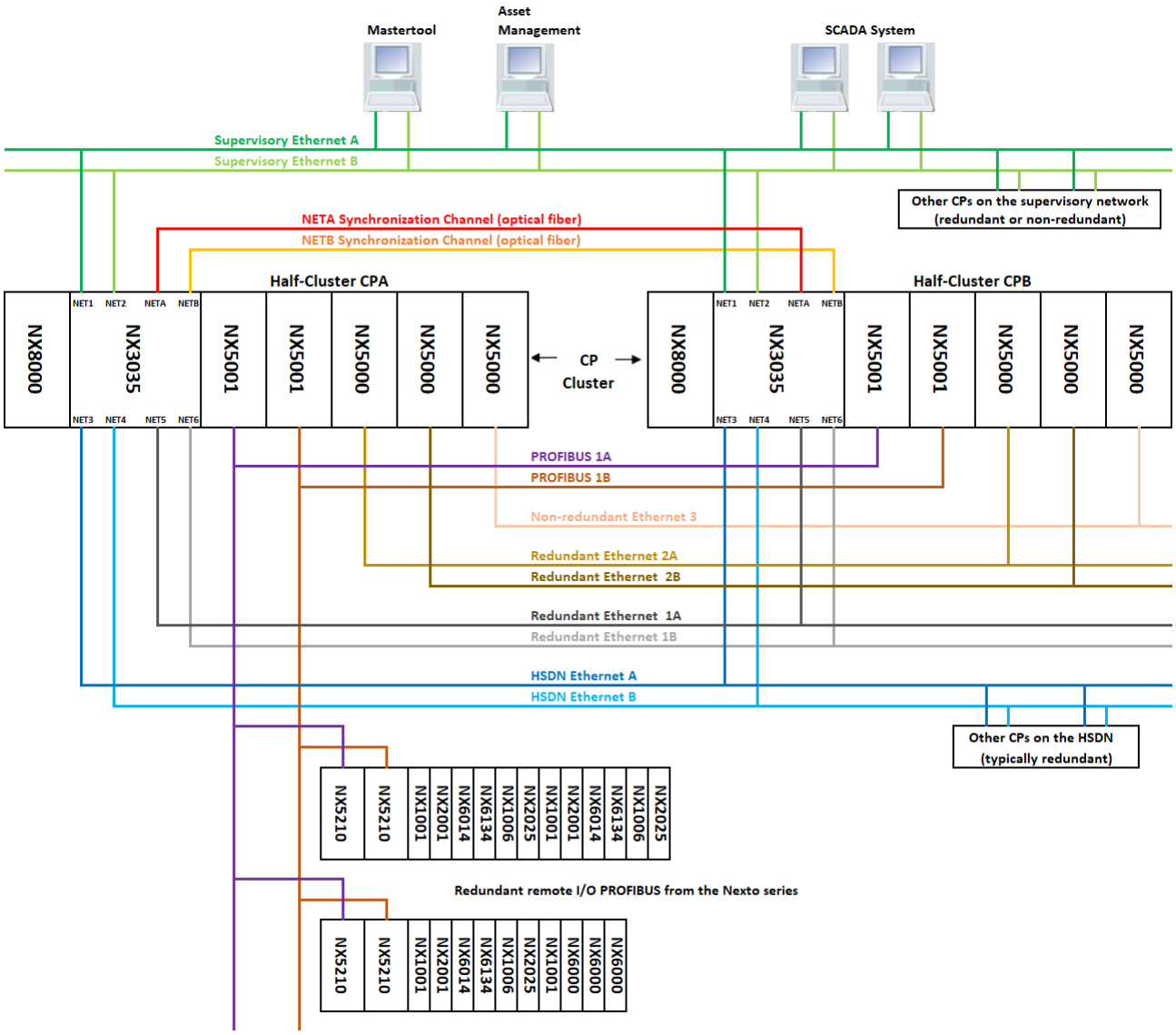


Figure 198: Example of Redundant Architecture with NX3035 CPU

6.2. Technical Description and Configuration

6.2.1. Minimum Configuration of a Redundant CPU

A redundant PLC consists of at least two identical half-clusters.

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

- The backplane 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 NX8000 power supply, in positions 0 and 1 of the backplane.
- The NX3035 CPU, in positions 2 to 5 of the backplane.

The figure below shows an example of a minimum configuration for a redundant PLC, using the NX9000 backplane (8 positions). Positions 6 and 7 of this backplane remain free.

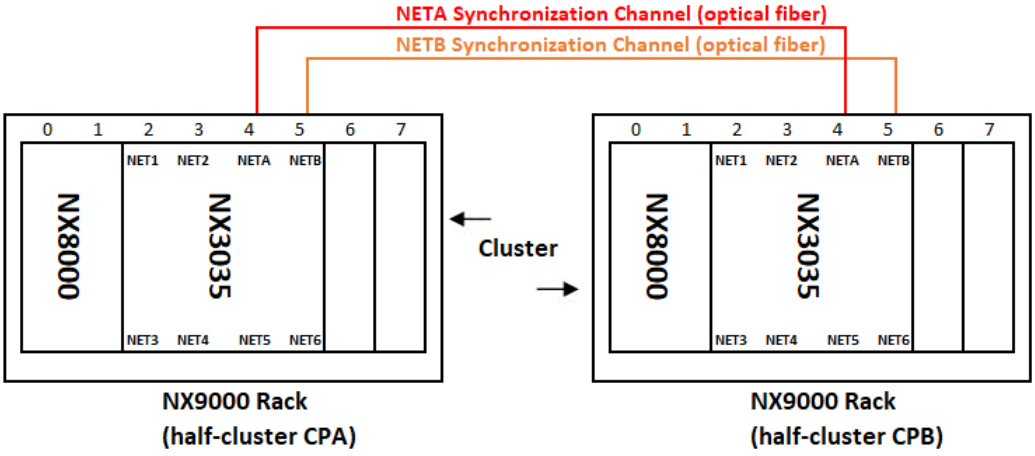


Figure 199: Minimum Configuration for a Redundant PLC in an NX9000 Backplane Rack

6.2.2. Synchronization Channels between PLCA and PLCB

Figure 199 shows that a connection must be made between the PLCA and PLCB half-clusters for synchronization purposes (exchange of redundant data, diagnostics, force lists, and projects).

This connection is made through two 1 Gbps communication ports, called NETA and NETB. Two optical cables must be installed between the two half-clusters:

- One between the NETA ports of CPU A and CPU B.
- Another between the NETB ports of CPU A and CPU B.

The NETA and NETB communication ports on the CPU NX3035 are located at the bottom of the unit and are SFP connectors where two types of transceivers can be installed:

- NX9500: Gigabit SFP multimode fiber transceiver (550m)
- NX9501: Gigabit SFP single-mode fiber transceiver (10Km)

These transceivers can be hot-swapped in the NETA and NETB connectors.

For more details on these transceivers, such as compatible optical connector and fiber types, see their CE.

ATTENTION

The connection between the NETA ports of CPU A and NETA ports of CPU B, as well as the connection between the NETB ports of CPU A and NETB ports of CPU B, must be made directly between the NX9500 or NX9501. No intermediate equipment (e.g., switch) should be installed between these two connections.

No synchronization service between PLCA and PLCB will work if the synchronization channel cables are reversed. For example, connecting the NETA channel of CPU A to the NETB of CPU B and the NETB channel of CPU A to the NETA of CPU B.

In some cases, half-clusters may be very close to each other, for example, within the same electrical panel. In this case, the NX9500 and two short-distance optical cables can be used to connect them.

In other cases, it is desirable for the two half-clusters to be far apart to prevent both from being damaged by an event such as a fire, resulting in a loss of redundancy. For example, in tunnel applications, the PLCA can be installed in a control room at one end of the tunnel, and the PLCB in a control room at the other end of the tunnel. In this case, depending on the distance, the NX9501 and two long-distance optical cables should be used to connect them.

6.2.3. Typical Configurations of a Redundant CPU

A minimal configuration, such as the one shown in Figure 199, can already be useful as a “redundant processing unit.” It would be possible to use the serial and Ethernet communication channel of the NX3035 CPU, for example, for MODBUS TCP communication with a SCADA system, and MODBUS RTU and/or MODBUS TCP communication with intelligent field devices or MODBUS remote I/O systems.

In larger configurations, additional modules are inserted into the PLCA and PLCB half-clusters, for example, to provide a PROFIBUS remote I/O system and additional Ethernet channels.

Among the additional modules that can optionally be inserted into the half-clusters are:

- PROFIBUS NX5001 masters
- NX5000 Ethernet interfaces

It is also possible to install non-redundant I/O modules in each of the half-clusters. This can be useful, for example, to create a panel of buttons and lamps to show the redundancy status and switch the redundancy status.

If necessary, larger backplanes can be used, such as the NX9001 (12 positions), NX9002 (16 positions), and NX9003 (24 positions).

The modules that can be inserted into the backplane occupy the following number of positions:

- NX8000: 2 positions (always positions 0 and 1)
- NX3035: 4 positions (always positions 2 to 5)
- NX5001: 2 positions (starting from position 6)
- NX5000: 2 positions (from position 6)
- I/O modules (from position 6): most I/O module models occupy 1 position in the backplane, and few models use 2 positions.

When inserting modules into a backplane, there are some limits, which are automatically checked by the Mastertool programmer:

- The number of positions in the selected backplane
- The current capacity supplied by the NX8000 power supply and consumed by the other modules.
- The quantity of certain modules:
  - NX8000: exactly 1 must be inserted
  - NX3035: exactly 1 must be inserted
  - NX5001: a maximum of 6 can be inserted
  - NX5000: a maximum of 6 can be inserted

6.2.3.1. Adding NX5001 Modules for PROFIBUS Networks

Up to six NX5001 modules can be inserted into each half-cluster for use with PROFIBUS networks in a redundant PLC.

To create a redundant PROFIBUS network, two NX5001 modules must be inserted into each half-cluster. To create a simple PROFIBUS network, simply insert one NX5001 module into each half-cluster.

By installing six NX5001 modules in each half-cluster, it is possible to configure up to three redundant PROFIBUS networks, or up to six simple PROFIBUS networks. Other possibilities also exist as long as the limit of six NX5001 modules is not exceeded (for example, two redundant PROFIBUS networks and two simple PROFIBUS networks).

If the PROFIBUS network “n” (where “n” is a number from 1 to 6) is redundant, the two networks that comprise it are named PROFIBUS “n”A and PROFIBUS “n”B. If the PROFIBUS network ‘n’ is simple, the network that comprises it is named PROFIBUS “n”A.

In the example in Figure 198, there is only one PROFIBUS network (PROFIBUS 1), and it is redundant (PROFIBUS 1A and PROFIBUS 1B). In this example, therefore, two NX5001 modules were inserted in each half-cluster.

More information on configuring PROFIBUS networks is provided in section PROFIBUS Network Configuration with NX5001 Modules.

6.2.3.2. Adding NX5000 Modules for Ethernet Networks

Up to six NX5000 modules can be inserted into each half-cluster to add remote Ethernet ports to a redundant PLC if the six Ethernet ports already present on the NX3035 are not sufficient.

Each NX5000 inserted into the two half-clusters adds one more 100/10 Mbps Ethernet port that allows connection to a simple Ethernet network. A redundant pair of NX5000s inserted into the two half-clusters adds two Ethernet ports that allow connection to a redundant Ethernet network, using a strategy called NIC Teaming.

By installing six NX5000s in each half-cluster, it is possible to configure up to three redundant Ethernet networks, or up to six simple Ethernet networks. Other possibilities also exist as long as the limit of six NX5000 modules is not exceeded (for example, two redundant Ethernet networks and two simple Ethernet networks). Additionally, the interfaces can be configured with the vital failure option. In this case, a complete network failure will trigger a switch-over.

In the example in Figure 198, there are 3 NX5000 modules installed in each half-cluster. The first two form a redundant pair (NIC teaming) connected to a redundant Ethernet network, and the third connects to a single Ethernet network.

More information on configuring NX5000 modules is provided in section Configuration of Ethernet Interfaces with NX5000 Modules.

6.2.3.3. Adding I/O Modules

It is possible to install I/O modules in the local backplanes of a half-cluster, although this is not common. One possible application would be to install an NX1005 module to implement a redundancy control panel with buttons and lamps. The buttons can be used to command changes in redundancy status, and the lamps can be used to show the current redundancy status.

6.2.4. General Characteristics of a Redundant CPU

	General Characteristics of a Redundant CPU
Permitted CPUs	NX3035
Redundancy type	Hot-standby
Fault tolerance	It tolerates, at a minimum, a single failure in duplicated equipment within the half-clusters. In specific cases, it may tolerate multiple failures.
5 redundancy states of a Half-Cluster	**Not-Configured:** initial state, also considered when the PLC is turned off or not executing the main task. **Initializing:** temporary state assumed after Not-Configured, where some tests will determine the next state (Inactive, Active, Standby - or back to Not-Configured). **Inactive:** state reached after certain types of failures or for scheduled maintenance. **Active:** controlling the user process. **Standby:** ready to switch to Active and control the user process, should there be a demand (e.g., failure of the Active PLC).
Main failures that cause switchover between the Active CPU and the Standby CPU. The Standby CPU switches to Active, and the Active CPU may transition to Inactive or Not-Configured.	- Lack of power supply on the Active PLC. - Failure in the Active PLC power source. - Failure in the Active PLC CPU (stop of the MainTask execution). - Failure of both synchronization channels (NETA and NETB) with evidence showing that the Active PLC is no longer controlling the process (example: standby PLC realizes that the PROFIBUS network is masterless). In this case, the Standby PLC takes the initiative to switch to Active. - Failure in a PROFIBUS network configured as vital failure. - Failure in an Ethernet network configured as vital failure.
Commands that cause switchover between the Active CPU and the Standby CPU	- Commands received from the Mastertool or from a SCADA system, through this PLC (local) or the other PLC (remote). - Commands generated by the user's application, for example, due to other diagnostics (e.g., Ethernet communication failure), through this PLC (local) or the other PLC (remote). - Command generated via the CPU's OTD button.
Main failures that prevent a CPU from remaining in, or assuming, the Standby state. Such failures cause the CPU to transition to the Not-Configured or Inactive states.	- Lack of power supply on this PLC. - Failure in this PLC's power source. - Failure in this PLC's CPU (stop of the MainTask execution). - Failure of both synchronization channels (NETA and NETB) where it is perceived that the other PLC is controlling the process (example: this PLC perceives that there is a master on the PROFIBUS network). - Failure in the redundant data synchronization service, but the NETA or NETB channels still report that the other PLC is in the Active state. - Failure in the redundant forcing list synchronization service, but the NETA or NETB channels still report that the other PLC is in the Active state. - Total failure in a PROFIBUS network configured as vital failure. - Total failure in an Ethernet network configured as vital failure. - Project different from the Active PLC's project, with automatic project synchronization enabled. - Firmware version incompatible with the Active PLC.
Commands that remove the CPU from the Standby state	- Commands received from the Mastertool or from a SCADA system, through this PLC (local) or the other PLC (remote). - Commands generated by the user's application, for example, due to other diagnostics (e.g., Ethernet communication failure), through this PLC (local) or the other PLC (remote). - Command generated via the CPU's OTD button.

	General Characteristics of a Redundant CPU
Switchover time	- Typically one to three MainTask cycles, depending on the stimulus for state change (command or failure). - In case of a PROFIBUS network failure, two MainTask cycles + 500 ms.
Bumpless switchover	- A switchover does not cause discontinuities in the controller outputs, nor in internal variables.
Redundancy overhead (CPU usage per MainTask cycle added by redundancy)	- Maximum value automatically calculated by Mastertool and informed to the user. - Typical average value of 31 ms for 1 Mbytes of redundant data.
CPU display	Among other diagnostics, it shows the state of redundancy (Active, Standby, Inactive, Not-Configured, Initializing). For the identification of the PLC, it is necessary to check the Redundancy menu, where the PLCA or PLCB identification is present.
Redundancy diagnostics	- Indicate failures in both the PLCA and PLCB, regardless of their states (Active or Non-Active). - Prevent "hidden failures." - Allow for fast maintenance, which is essential for achieving high availability.
Redundancy commands	- Allow the execution of the same control actions, in addition to other commands (e.g., commanding switchovers). - Can be executed on the local PLC, or forwarded to the other PLC (remote) via NETA / NETB synchronization channels. - Can be received from the Mastertool or from a SCADA system. - Can be executed from the user's application. - Can be executed using the OTD button of the NX3035.
Redundancy events	The log accessible via Mastertool registers modifications in redundancy diagnostics and commands, with a timestamp, thus allowing an investigation into the causes of a switchover.
SNTP (Simple Network Time Protocol)	Allows events to have a precise timestamp adjusted to world time. It also synchronizes the PLC's real-time clock for other applications.
Command and diagnostics synchronization	In every MainTask cycle, PLCA and PLCB exchange diagnostics and commands through the NETA or NETB synchronization channels. In this way, one PLC knows the diagnostics and commands of the other.
Redundant data synchronization	In every MainTask cycle, the Active PLC copies redundant data to the Non-Active PLC through the NETA and NETB synchronization channels. Non-redundant data is not synchronized.
Synchronization of the redundant forcing list	In every MainTask cycle, the Active PLC copies the redundant forcing list to the Non-Active PLC through the NETA and NETB synchronization channels. This list only includes forced redundant variables. In this way, PLCA and PLCB may have different sets of forced non-redundant data, as these forcings are not synchronized.
Single project for PLCA and PLCB	There is a single common project for the PLCA and PLCB, generated by the Mastertool. The project is composed of the application project (executable code) and the *project archive* (source code).
Identification of a PLC	Through the Mastertool, a CPU is identified as PLCA, PLCB, or Non-redundant PLC. This identification is not part of the application project generated by the Mastertool, although it is written into a PLC using the Mastertool. The identification of each PLC enables the characteristic of having a single project for PLCA and PLCB.
Automatic project synchronization	If the Active PLC's project becomes different from the Non-Active PLC's project, it is copied from the Active PLC to the Non-Active PLC. This synchronization occurs at low priority and may take several seconds, and during its execution, the Non-Active PLC cannot reach the Standby state. It should be remembered that the project is composed of the application project (executable code) and the source code (project archive), both of which are synchronized. This synchronization can be disabled in special circumstances to enable project modifications that can only be loaded offline on non-redundant PLCs. Examples of such special circumstances are described in the chapter Offline Program Download Without Interrupting Process Control.

	General Characteristics of a Redundant CPU
Hot expansion of modules and PROFIBUS remote stations	There are project modifications that cannot be performed hot (online) on a non-redundant PLC, such as the inclusion of new PROFIBUS modules or remote I/O. However, by leveraging the redundancy of the PLC and the PROFIBUS networks, a procedure has been defined to achieve this objective, which is very important for systems requiring high availability (see chapter Offline Program Download Without Interrupting Process Control).
Automatic synchronization of Firewall, SNMP, FTP, SysLog, and Memory Card settings	If the settings for any functionality in the Active PLC become different from the settings in the Non-Active PLC, the same settings will be copied from the Active PLC to the Non-Active PLC, provided that the Project Synchronization is enabled in both PLCs. This synchronization can be disabled in special circumstances to allow configuration changes without immediate synchronization. Changing the settings of these functionalities is not permitted while the PLC is in the Standby (SBY) state. In the Inactive (INA) and Not-Configured (NCF) states, changes are permitted, but are subject to being overwritten by the Active PLC if the Project Synchronization is enabled and this PLC assumes the Standby (SBY) state.
Private IP addresses for PLCA and PLCB	It is possible to connect to a specific PLC (PLCA or PLCB) using a private IP address for it. This serves, for example, to obtain specific diagnostics for that half-cluster. There is a PLCA IP address for each local interface of the CPU A (NET1 ... NET6), and there is also a PLCB IP address for each local interface of the CPU B (NET1 ... NET6), as described in section IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup. For NX5000 modules, there are also addressing modes that have specific addresses for the PLCA and PLCB, as described in section IP Address Configuration on NX5000 Modules in a Redundant CPU Setup.
Active IP	A strategy that allows Ethernet clients to connect to a server on the Active PLC, always using the same IP address. This avoids the need for complex scripts to change the IP address when switchovers occur due to redundancy. There is one Active IP address for each local interface of the Active PLC (NET1 ... NET6), as described in section IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup. For NX5000 modules, there are also addressing modes that have one Active IP address, as described in section IP Address Configuration on NX5000 Modules in a Redundant CPU Setup.
NIC Teaming	A strategy that allows two Ethernet interfaces of a half-cluster to form a redundant pair sharing the same IP address(es). This strategy can also be used in non-redundant PLCs and has already been described in sections Integrated Ethernet Interfaces Configuration and Configuration of Ethernet Interfaces with NX5000 Modules. This strategy can be defined between the following pairs of Ethernet ports: - NET1 and NET2 - NET3 and NET4 - NET5 and NET6 - Redundant pair of NX5000 modules
PROFIBUS Networks and Vital Failure Configuration	The PLC supports up to six single PROFIBUS networks or up to three redundant PROFIBUS networks. For each of them, it is possible to configure whether a total failure is considered vital (causes a switchover) or not. The total failure of a single PROFIBUS network is determined when the NX5001 master cannot communicate with any of the PROFIBUS slaves in this network. The total failure of a redundant PROFIBUS network is determined when both NX5001 masters cannot communicate with any of the PROFIBUS slaves in the two redundant networks.
Single, cyclic user task	Only one user task is allowed, called the MainTask. This task is cyclical.
Main POUs: NonSkippedPrg and ActivePrg	In the creation of a redundant project, the Mastertool automatically generates two empty program-type POUs that must be filled by the user: - NonSkippedPrg: This POU is executed on both PLCs (PLCA and PLCB), regardless of the redundancy state (Active or Non-Active). It is intended for the management of special diagnostics and commands. - ActivePrg: This POU is executed only on the Active PLC, and is intended for the control of the end-user's process.

General Characteristics of a Redundant CPU

Table 156: General Characteristics of a Redundant CPU

6.2.5. Purchase Data for Redundancy

The minimum configuration for a redundant PLC involves purchasing the following modules:

- Two backplanes, which must be chosen from four available models according to the modules to be installed:
  - NX9000: eight positions
  - NX9001: twelve positions
  - NX9002: sixteen positions
  - NX9003: twenty-four positions
- Two NX8000
- Two NX3035
- Four NX9500 or NX9501

In addition, you may need to purchase the following optional modules:

- Two NX5001 modules for each additional single PROFIBUS network
- Four NX5001 modules for each additional redundant PROFIBUS network
- Two NX5000 modules for each additional single Ethernet network
- Four NX5000 modules for each additional redundant Ethernet network (NIC Teaming)

6.3. Principles of Operation

This section describes the functions of a redundant PLC with an NX3035 CPU, its behavior, and states. It also presents programming and configuration concepts and restrictions that will be used in the following sections.

6.3.1. Identification of an NX3035 CPU

An NX3035 CPU has a non-volatile identification register where it can be identified as:

- **Non-Redundant:** cannot be used in a redundant PLC (factory default setting)
- **PLCA:** used in the PLCA of a redundant PLC
- **PLCB:** used in the PLCB of a redundant PLC

The identification register can be adjusted with the Mastertool programmer. The first thing to do on an NX3035 CPU, before loading the redundant PLC project onto it, is to identify it as PLCA or PLCB. If the identification is not performed, it is not possible to load a redundant project into the PLC.

ATTENTION

The PLC Identification register is not part of the redundant PLC project, and therefore is not saved as part of this project on the computer where Mastertool is running. The register is only saved in the CPU's non-volatile memory.

6.3.2. Single Redundant Project

There is a single project for the redundant PLC, identical for both PLCA and PLCB.

Configuration parameters that must be different between PLCA and PLCB (e.g., IP addresses of Ethernet interfaces) appear duplicated in the redundant PLC project (one for PLCA, one for PLCB). Each PLC will consider the one that corresponds to it, after analyzing its identification record.

6.3.3. Features Configured on the WEB Page

Some features configured on the web page of the redundant PLC will be identical for PLCA and PLCB.

These features are listed in Table General Characteristics of a Redundant CPU.

6.3.4. Redundant Project Structure

6.3.4.1. Redundancy Template

The design of a redundant PLC is automatically created from a template called *Redundancy Template*.

The template is based on the minimum configuration of a redundant PLC, as defined in section Minimum Configuration of a Redundant CPU. In addition, some dialogs with the user are performed to select the backplane and insert 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.

Additionally, basic program-type tasks and POUs are created, as described in the following sections.

6.3.4.2. Single and Cyclic Task MainTask

The redundant PLC project has a single task, called MainTask, which is cyclical. The user must adjust the task interval according to their needs.

6.3.4.3. MainPrg Program

The MainTask is associated with a single program-type POU, called MainPrg. The MainPrg program is created automatically.

The code for the MainPrg program is as follows, in ST language:

Configuration (Bus) | MainPrg

```
(*Main POU associated with MainTask that calls StartPrg, UserPrg/ActivePrg and NonSkippedPrg. This POU is blocked to edit.*)
PROGRAM MainPrg
VAR
   isFirstCycle : BOOL := TRUE;
END_VAR
```

```
SpecialVariablesPrg();
IF isFirstCycle THEN
   StartPrg();
   isFirstCycle := FALSE;
ELSE
   fbRedundancyManagement();
   IF fbRedundancyManagement.m_fbDiagnosticsLocal.eRedState = REDUNDANCY_STATE.ACTIVE THEN
      SpecialVariablesRedundantPrg();
      NonSkippedPrg();
      ActivePrg();
   ELSE
      NonSkippedPrg();
   END_IF;
END_IF;
```

Figure 200: MainPrg Program Code

MainPrg calls three other program-type POUs, named *NonSkippedPrg*, *ActivePrg*, and *SpecialVariablesRedundantPrg*.

The POU *NonSkippedPrg* is always executed, both in Active and Non-Active PLC.

The POUs *ActivePrg* and *SpecialVariablesRedundantPrg* are only executed when the PLC is in the Active state.

The POUs *MainPrg* and *SpecialVariablesRedundantPrg* are locked for editing, so they cannot be modified by the user.

The user can modify the *NonSkippedPrg* and *ActivePrg* programs. Initially, when the redundant project is created from the *Redundancy Template*, these two programs are created "empty," but the user can insert code into them.

ATTENTION

When the OPC DA communication option is enabled during project creation, the *NonSkippedPrg* program will not be created empty. For more information, see the section Use of OPC DA Communication with Redundant Projects.

6.3.4.4. ActivePrg Program

The main objective of this program, which runs only on PLC Active, is to control the end user process.

This program normally acts on redundant variables, including the direct representation variables %I and %Q associated with the remote I/O system. For more information, see the section Redundant CPU Programming - MainTask Configurations - ActivePrg Program.

6.3.4.5. NonSkippedPrg Program

This program, executed on both PLCs (PLCA and PLCB) regardless of redundancy status, is typically used for functions such as:

- Organizing non-redundant diagnostics to report to a SCADA system.
- Receiving and executing non-redundant commands from a SCADA system.
- Managing switch-over conditions not normally covered automatically by the redundant PLC, which may vary from user to user. For example, a given user may execute a switch-over to the Standby PLC if the Active PLC is not communicating with the SCADA system, while another user may not want a switch-over in this situation.
- Enable or disable I/O drivers depending on the redundancy status, for example, disable a MODBUS RS-485 master on the Non-Active PLC.
- Detect failures in I/O drivers on a Non-Active PLC to avoid hidden failures. Some I/O drivers do not provide automatic detection of such failures. Other I/O drivers, such as PROFIBUS, do so automatically.
- Other activities that, for some reason, need to be performed on both the Active PLC and the Standby PLC.

For more information, see the section Redundant CPU Programming - MainTask Configurations - NonSkippedPrg Program.

6.3.4.6. Redundant and Non-redundant Variables

The variables of a redundant PLC can be classified as redundant or non-redundant. Redundant variables are copied from the Active PLC to the Non-Active PLC at the beginning of each MainTask cycle through the NETA and NETB synchronization channels. On the other hand, non-redundant variables are not copied between half-clusters and, therefore, may have different values in PLCs A and B.

Non-redundant variables are used to store information specific to each half-cluster (PLCA or PLCB), such as diagnostics of a module within this half-cluster, including redundancy diagnostics (redundancy status of this half-cluster, etc.). For example, the failure of an NX5001 module installed in position 6 of the PLCA backplane may have a different value than the failure of the corresponding NX5001 module installed in position 6 of the PLCB backplane. In addition, the redundancy status is usually different (e.g., PLCA in Active state, PLCB in Standby state)

Redundant variables relate to shared information and process control. Variables associated with remote I/O modules are typical examples of redundant variables. In general, a variable needs to be redundant when its value in the current application cycle can influence the behavior of the application in the next cycle. Thus, if it were not copied from the Active PLC to the Standby PLC, there would be a discontinuity in the event of a switchover (if the Standby PLC became Active). For example, in the following code snippet, the COUNTER variable needs to be redundant:

```
COUNTER := COUNTER + 1;
```

If the COUNTER variable were not redundant, in the event of a switchover, the value of the COUNTER variable in the Standby PLC could be different from the value of the COUNTER variable in the Active PLC. Thus, the value of the COUNTER variable could revert to a value lower than the current value, or advance to a value higher than the current value, depending on the value stored in the Standby PLC. In both cases, there would be a discontinuity in the application's behavior.

There are also variables that can be redundant or non-redundant without affecting the correctness of the user's application. It is not very easy to identify this type of variable. As an example, we could mention auxiliary variables used for intermediate calculations within a POU.

It is easy to identify when a variable should be non-redundant. As explained earlier, this is a variable that represents information specific to each half-cluster, for example, a specific diagnosis of the PLCA or PLCB.

On the other hand, it is not very easy to identify variables that could be redundant or non-redundant without affecting the correctness of the application. For this reason, the following strategy is suggested:

1. Identify whether the variable should be non-redundant. In this case, declare it in a non-redundant POU or GVL.
2. Declare all other variables as redundant, even if they could be non-redundant.

This strategy may fail if there is not enough redundant memory. In this case, it will be necessary to identify some variables that could be redundant or non-redundant without affecting the correctness of the application, and declare them as non-redundant.

The following sections describe the redundant memory limits for direct representation variables (%I, %Q, and %M) and for symbolic variables, and also describe how to declare them as redundant or non-redundant.

6.3.4.7. Redundant and Non-redundant %I Variables

The NX3035 CPU allocates 384 Kbytes of %I variables (%IB0 ... %IB393215):

- The first 368 Kbytes may be redundant or non-redundant (%IB0 ... %IB376831)
- The last 16 kbytes will never be redundant (%IB376832 ... %IB393215)

The 368 Kbytes area that may be redundant is allocated for remote I/O system entries, such as:

- Input variables from I/O modules installed on remote PROFIBUS devices.
- Input variables from read mappings of a MODBUS client.

The 16 Kbytes non-redundant area is allocated for fast private diagnostics of a half-cluster and for I/O modules installed in the same local backplane rack where the NX3035 is installed. Fast diagnostics are those that must be updated at each MainTask cycle. We have the following examples of variables allocated in this non-redundant area:

- Quick diagnostics for each NX5001
- Quick diagnostics for each NX5000
- Input variables for I/O modules installed on the local bus of the NX3035.

The user can configure the number of redundant %I variables, between 0 bytes and 376832 bytes, in multiples of 1 byte (the default value is 16384 bytes - %IB0 ... %IB16383).

The following figure illustrates the allocation of redundant and non-redundant direct representation variables %I, where the following parameters can be configured in Mastertool:

- **RI:** amount in bytes of %I configured as redundant.

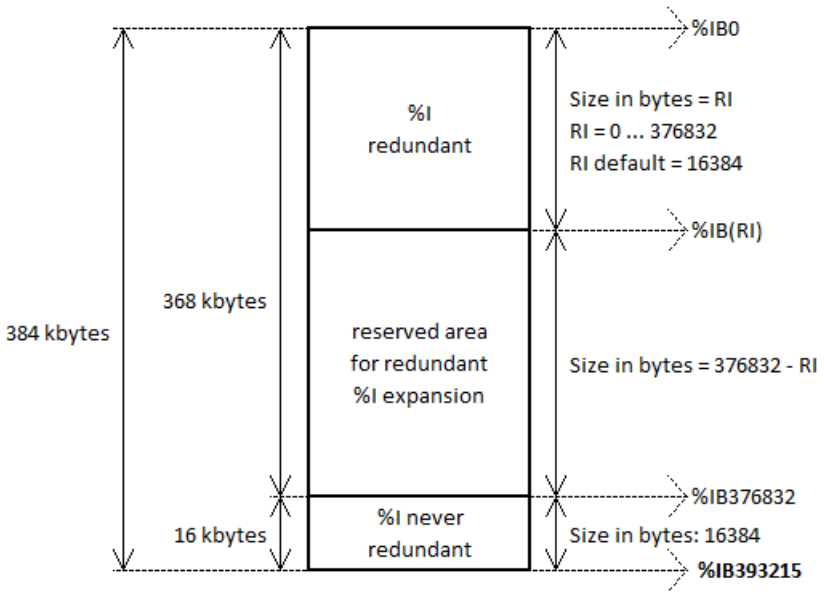


Figure 201: Allocation of Redundant and Non-Redundant %I

6.3.4.8. Redundant and Non-redundant %Q Variables

The NX3035 CPU allocates 384 Kbytes of %Q variables (%QB0 ... %QB393215):

- The first 368 Kbytes may be redundant or non-redundant (%QB0 ... %QB376831)
- The last 16 kbytes will never be redundant (%QB376832 ... %QB393215)

The 368 Kbytes area that may be redundant is divided into two sections.

The first section that may be redundant always starts at %QB0. Its size can vary between 0 and 368 kbytes, but the default size is 16 kbytes. This section is allocated for outputs from remote I/O systems, such as:

- Output variables from I/O modules installed on remote PROFIBUS devices.

- Output variables from write mappings of a MODBUS client.
- Variables available for reading/writing in mappings of a MODBUS server.

The second section, which may be redundant, can start from %QB0, but the default start address is %QB262144. Its size can vary between 0 and 368 kbytes, but the default size is "8 kbytes + NRP * 16 kbytes," where NRP is the number of PROFIBUS networks generated by the Mastertool Wizard when creating the project (maximum NRP value = 6). This section is allocated for diagnostics of redundant I/O systems. Unlike fast diagnostics (allocated in %I), such diagnostics allocated in %Q may take more than one MainTask cycle to update. Examples of such diagnostics include:

- MODBUS client mapping diagnostics.
- PROFIBUS slave diagnostics, which are divided into:
  - PROFIBUS remote head diagnostics.
  - Diagnostics of I/O modules installed in PROFIBUS remotes.

The last 16 kbytes, which are never redundant (%QB376832 ... %QB393215), are used to allocate, for example:

- NX3035 diagnostics
- Diagnostics for each NX5001
- Special commands for each NX5001
- Diagnostics for each NX5000
- Special commands for each NX5000
- Diagnostics for each MODBUS client
- Disabling mappings for each device for each MODBUS client
- Diagnostics for each MODBUS server
- Disabling mappings for each MODBUS server
- Diagnostics for I/O modules installed in the local rack of the NX3035
- Output variables of I/O modules installed in the local rack of the NX3035

The following figure illustrates the allocation of redundant and non-redundant direct representation variables %Q, where the following parameters can be configured in Mastertool:

- **RQS:** amount in bytes of the first section of redundant %Q, used for remote I/O system outputs.
- **INI_RQD:** starting address of the second section of redundant %Q, used for remote I/O system diagnostics.
- **RQD:** amount in bytes of this second section of redundant %Q.

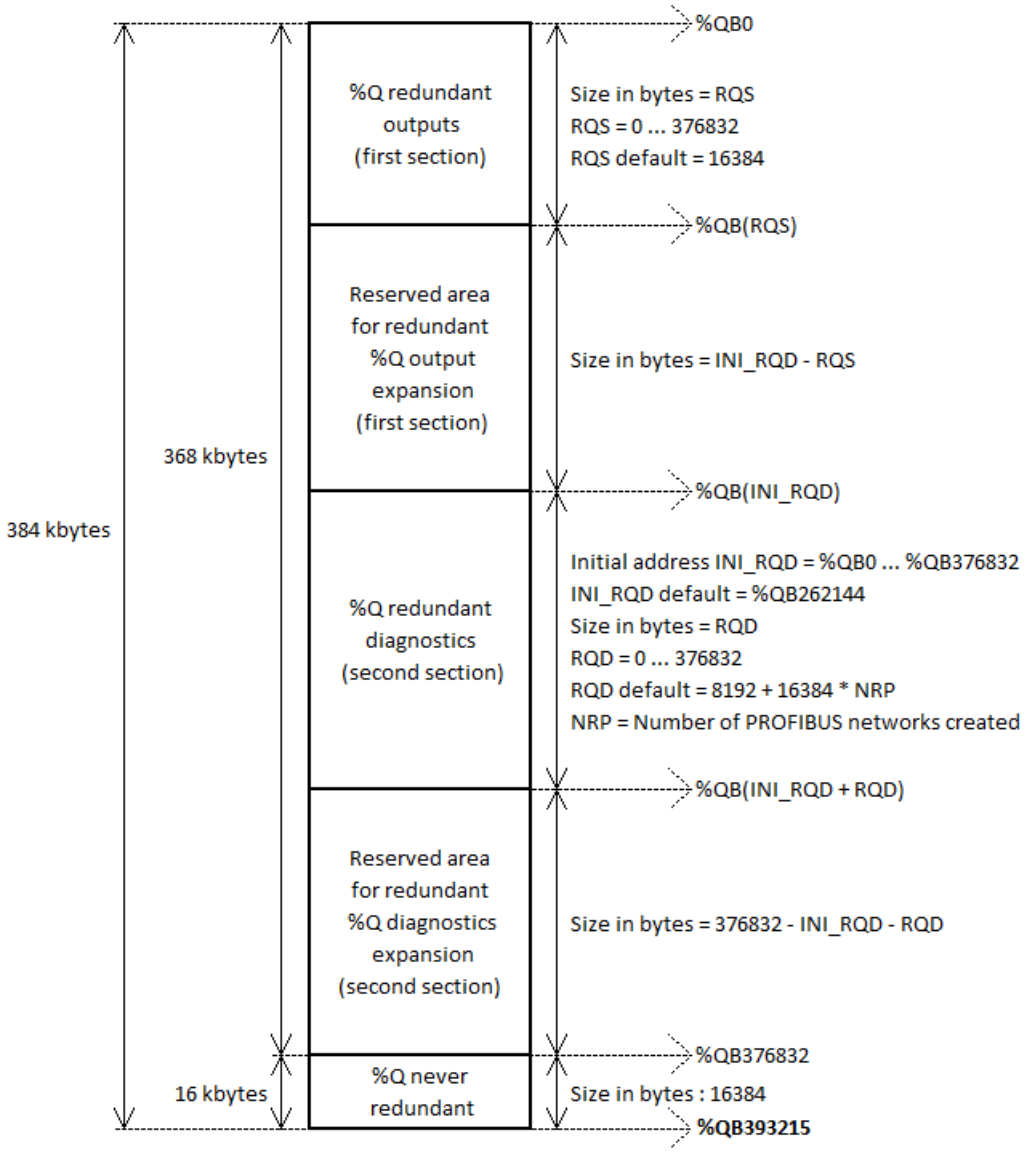


Figure 202: Allocation of Redundant and Non-Redundant %Q

The second section of redundant %Q variables, with initial address INI_RQD and size RQD, is divided into "1 + NRP" subsections, where NRP is the number of configured PROFIBUS networks.

- The first subsection is intended for redundant diagnostics not associated with PROFIBUS networks. It can be used, for example, for diagnostics of MODBUS client mappings. The starting address of this subsection is INI_RQD. It is recommended to manually configure the size of this subsection according to the application's requirements. A size of 8192 bytes is recommended.
- Each of the next NRP subsections is intended for diagnostics of slaves on a PROFIBUS network. The starting address of each of these subsections is defined by a parameter of the corresponding PROFIBUS network master NX5001. This is the parameter "Starting Address of Slave Diagnostics in %Q". It is recommended that the user manually configure the size of each subsection according to the application's requirements. A size of 16384 bytes is recommended for each subsection.

The following figure shows the organization of the subsections of the second section of redundant %Q variables. Using Mastertool, the user can configure the following parameters shown in this figure:

- **INI_RQD:** already described above.
- **INI_PB1, INI_PB2, ..., INI_PB(NRP):** parameters "Initial Address of Slave Diagnostics in %Q" of the NX5001 masters of the configured PROFIBUS NRP networks.

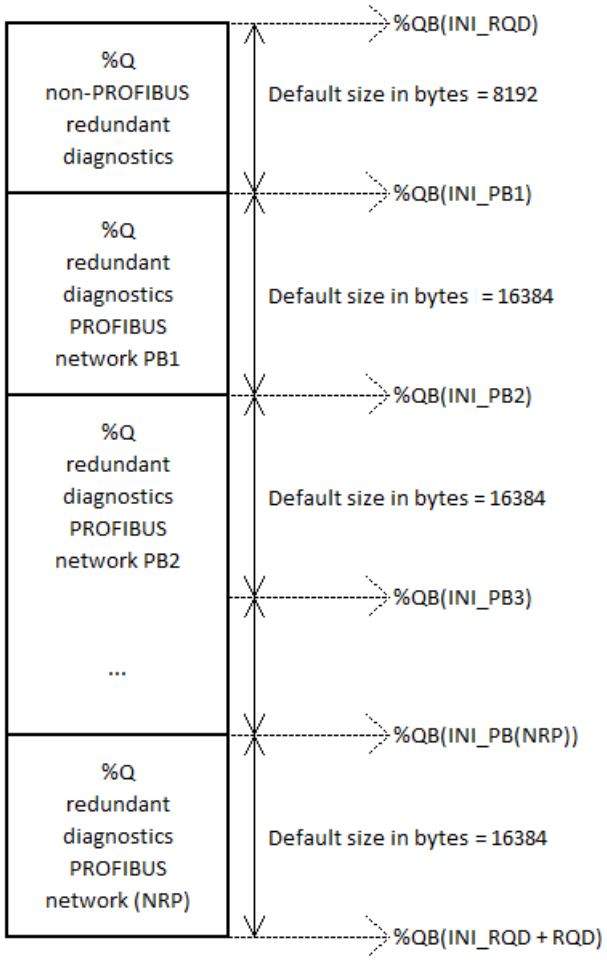


Figure 203: Redundant %Q Subsections for Remote I/O Diagnostics

6.3.4.9. Redundant and Non-redundant %M Variables

The NX3035 CPU allocates 128 Kbytes of %M variables (%MB0 ... %MB131071).

All 128 kbytes can be redundant. By default, the number of redundant %M variables is 0.

The use of redundant and even non-redundant %M variables should be avoided, preferring the use of symbolic variables (see section Redundant and Non-Redundant Symbolic Variables).

The following figure illustrates the allocation of redundant and non-redundant %M direct representation variables, where the following parameters can be configured in Mastertool:

- **RM:** amount in bytes of redundant %M variables.

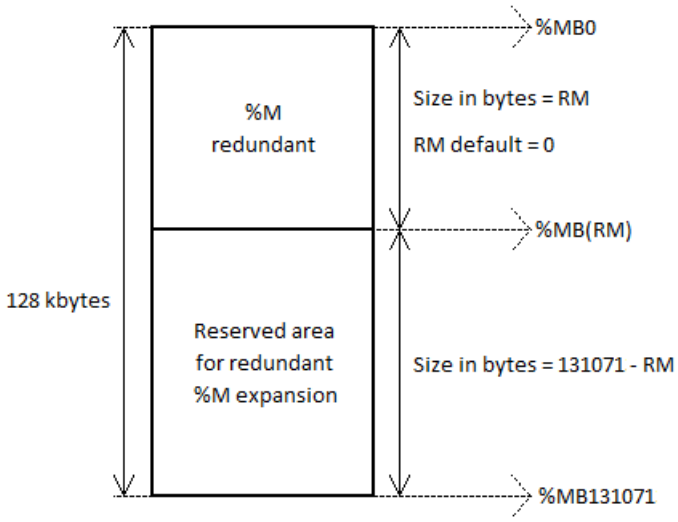


Figure 204: Allocation of Redundant and Non-Redundant %M

6.3.4.10. Redundant and Non-Redundant Symbolic Variables

In addition to the direct representation variables (%I, %Q, and %M) that are allocated automatically, the user can explicitly declare symbolic variables within POUs or GVLs.

The maximum size allowed for allocating redundant and non-redundant symbolic variables can be found in Table 7.

ATTENTION

Symbolic variables should not be confused with symbolic variables addressed using the AT directive. Symbolic variables that use the AT directive are symbolic names assigned to direct representation variables (%I, %Q, and %M). Therefore, variables that use the AT directive do not allocate any symbolic variable memory. For more details, see section Symbolic Variables Addressed with the AT Directive.

Some POUs automatically generated by the Wizard when creating the project are non-redundant, that is, the symbolic variables declared within them are non-redundant:

- MainPrg
- NonSkippedPrg
- SpecialVariablesPrg

Some GVLs automatically generated by the Wizard when creating the project are non-redundant, i.e., the symbolic variables declared within them are non-redundant:

- System_Diagnostics
- Module_Diagnostics
- ReqDiagnostics
- IOQualities
- Special_Variables

All other POUs and GVLs automatically generated by the Wizard when creating the project are redundant.

When the user creates a POU or GVL, it is defined as redundant. Normally, this is the situation desired by the user, but in some cases, the user may wish to change it to non-redundant. For example, a POU called within the NonSkippedPrg POU for diagnostic handling should typically be non-redundant.

Using the *Redundancy Configuration* object in the Mastertool device tree, the user can define whether a POU or GVL will be redundant or not. However, some POUs and GVLs automatically generated by the Wizard have a fixed redundancy attribute (locked for user editing).

The criteria for defining whether a variable should be redundant were discussed in the section Redundant and Non-redundant Variables.

The following additional observations on symbolic variable redundancy should be considered:

- Variables defined in FUNCTION-type POUs are non-redundant (even if they have the VAR STATIC attribute, which is not recommended as it is considered poor programming practice).
- The mere declaration of a FUNCTION BLOCK type POU does not allocate any memory. But when a FUNCTION BLOCK is instantiated within a PROGRAM type POU or within a GVL, this instance will allocate memory. This memory will be redundant or non-redundant, according to the redundancy configuration of the POU or GVL where the FUNCTION BLOCK was instantiated (object *Redundancy Configuration* in the Mastertool device tree).

6.3.4.11. Symbolic Variables Addressed with the AT Directive

The AT directive can be used within GVLs to assign symbolic names to direct representation variables (%I, %Q, and %M).

For example, if the variable %IW2120 corresponds to an analog input that measures the level of a tank, we can give it a meaningful name within a GVL or POU using a statement such as the following:

```
TANK_LEVEL AT %IW2110 : WORD;
```

Therefore, variables declared using the AT directive do not allocate symbolic memory. They merely function as an "alias" for a direct representation variable allocated as %I, %Q, or %M.

Variables named using the AT directive can only be declared in POUs or GVLs marked as "non-redundant" in the *Redundancy Configuration* object of the Mastertool device tree. This is true even if the direct representation variable is redundant.

If the AT directive is used within a POU or GVL marked as redundant, Mastertool will report a compilation error.

6.3.5. Data Structures for Diagnostics, Commands, and User Information

Each PLC has several data structures related to redundancy, under the data structure *DG_NX3035.tRedundancy*.

The following structures are symbolic variables that use the AT directive to map variables of type %Q:

- **RedDgnLoc:** contains diagnostics from this PLC (local) related to redundancy, such as the redundancy status of this PLC.
- **RedDgnRem:** is a copy of the *RedDgnLoc* structure from the other PLC (remote), received via NETA/NETB synchronization channels. This way, this PLC (local) has access to the diagnostics of the other PLC (remote).
- **RedCmdLoc:** contains commands that must be applied to the local PLC (when they have the suffix *Local*) or to the remote PLC (when they have the suffix *Remote*). For example, the *bStandbyLocal* field in this data structure requests that the local PLC go to the *Standby* state, while the *bStandbyRemote* field requests that the remote PLC go to the *Standby* state.
- **RedCmdRem:** is a copy of the *RedCmdLoc* structure of the remote PLC, received via NETA/NETB synchronization channels. This way, this PLC (local) can execute commands that have the *Remote* suffix received from the other PLC (remote).
- **RedUsrLoc:** contains 128 bytes of data freely filled in by the user on the local PLC and transmitted to the remote PLC via NETA/NETB synchronization channels. A typical use of this area is to transmit private data in an unusual direction, from the non-active PLC to the active PLC.
- **RedUsrRem:** is a copy of the *RedUsrLoc* structure from the other PLC, received via NETA/NETB synchronization channels. A typical use of this area is to transmit private data in an unusual direction, from the non-active PLC to the active PLC.

In section Redundancy Maintenance, subsection Structure of Diagnostics, Commands, and User Information for Redundancy will provide further details on these data structures.

6.3.6. Cyclic Synchronization Services through NETA and NETB

This section describes the three synchronization services that occur cyclically in a redundant PLC, between PLCA and PLCB, through the NETA and NETB synchronization channels.

These services are executed at the beginning of each MainTask cycle, in the order listed below, that is:

- First, the Diagnostics and Commands Exchange service is executed.
- Second, the Redundant Data Synchronization service is executed.
- Third, the Redundant Forcing List Synchronization service is executed.

6.3.6.1. Diagnostics and Commands Exchange

This service is responsible for exchanging the following data structures in each MainTask cycle:

- Copy *RedDgnLoc* from PLCA to *RedDgnRem* from PLCB
- Copy *RedCmdLoc* from PLCA to *RedCmdRem* from PLCB
- Copy *RedUsrLoc* from PLCA to *RedUsrRem* from PLCB
- Copy *RedDgnLoc* from PLCB to *RedDgnRem* from PLCA
- Copy *RedCmdLoc* from PLCB to *RedCmdRem* from PLCA
- Copy *RedUsrLoc* from PLCB to *RedUsrRem* from PLCA

The service will be performed using only one of the synchronization channels (NETA or NETB). This way, the service can be completed even if one of the channels is experiencing problems.

6.3.6.2. Redundant Data Synchronization

This service is responsible for transferring redundant variables from the Active PLC to the Non-Active PLC. As seen previously, there are symbolic redundant variables and also redundant variables of direct representation (%I, %M, and %Q).

For this service to be executed, several conditions must be fulfilled:

- The previous synchronization service of this MainTask cycle (Diagnostics and Command Exchange) must have been successfully completed.
- If this PLC is in Active state, the other must be in Non-Active state. On the other hand, if this PLC is in Non-Active state, the other must be in Active state.
- The designs of the two PLCs must be identical, except when automatic design synchronization is disabled (see section Project Synchronization Disabling).
- At least one synchronization channel (NETA or NETB) must be in operational status.

6.3.6.3. Redundant Forcing List Synchronization

This service is responsible for transferring the list of redundant forces from the Active PLC to the Non-Active PLC.

For this service to be executed, several conditions must be met:

- The two previous synchronization services in this cycle (Diagnostics Exchange and Command and Redundant Data Synchronization) must have been successfully completed.
- If this PLC is in Active state, the other must be in Non-Active state. On the other hand, if this PLC is in Non-Active state, the other must be in Active state.
- The designs of the two PLCs must be identical, except when automatic design synchronization is disabled (see section Project Synchronization Disabling).
- At least one synchronization channel (NETA or NETB) must be in operational status.

ATTENTION

The list of redundant constraints contains only constraints on redundant variables. In addition, in each of the PLCs (PLCA and PLCB), there may be a different list of constraints on non-redundant variables.

6.3.7. Sporadic Synchronization Services through NETA and NETB

The following synchronization services are performed sporadically. That is, they are not performed at each cycle of the MainTask, and another system task performs these sporadic services in the background, with lower priority.

6.3.7.1. Project Synchronization

This service is responsible for keeping Active and Inactive PLC projects synchronized. This only occurs when the projects are different and automatic project synchronization is enabled on both PLCs.

Synchronization always occurs from the Active PLC to the Inactive PLC, and only one of the synchronization channels (NETA or NETB) needs to be operational for this service to run.

When synchronization is enabled, the following files and services will be synchronized:

- Application project (executable code)
- Project archive (source code)
- Users and groups
- Access rights
- PLC Settings
- Trace

The synchronization service will start within thirty seconds after one of the PLCs enters Active mode. After starting, the service will compare the CRC of the Active PLC and Non-Active PLC projects every five seconds.

When synchronization is initiated, the Non-Active PLC will go into Stop mode and assume the Unconfigured state. After transferring all necessary files, the Non-Active PLC will go into Run mode and assume the Initializing state. If the transfer fails, the PLC will return to the Unconfigured state. If the transfer is successful, the PLC will normally go to the Standby state (if there are no failures that prevent this).

The time it takes to synchronize projects will depend on the size of the project and the occupancy of the NX3035 CPU.

If synchronization is interrupted (loss of communication between the synchronization channels) during the transfer of files from the Active PLC to the Non-Active PLC, the procedure will be aborted and restarted when communication is restored. Only after the entire procedure is complete will the Non-Active PLC go into Run mode.

In addition to keeping projects synchronized, Project Synchronization will also prevent the Non-Active PLC from assuming states higher than Initializing if the CRC is different or if any Online Changes are pending in the Active PLC.

During project synchronization in a redundant system, temporary failures in the data exchange between the Active and Non-Active PLC may be detected and indicated through synchronization diagnostic counters (for example, diagnostic/command exchange synchronization failures).

This behavior is expected while the Non-Active PLC is receiving and applying the updated application, since the synchronization mechanisms operate asynchronously relative to the project synchronization process. These temporary failures do not necessarily indicate a malfunction in the redundancy system.

ATTENTION

A project synchronization has the same effects as a download to the Non-Active PLC. This service is not executed if automatic project synchronization is disabled, as described in the section Disabling Project Synchronization. No synchronization service between PLCA and PLCB will operate if the synchronization channel cables are reversed. For example, connecting the NETA channel of CPU A to NETB of CPU B, and the NETB channel of CPU A to NETA of CPU B.

6.3.8. Project Synchronization Disabling

The section Sporadic Synchronization Services through NETA and NETB describes services for synchronizing the application project and the project archive. These services should normally be enabled and are useful when project modifications can be uploaded online to the Active PLC and then automatically retransmitted to the Standby PLC via NETA / NETB synchronization channels.

However, there are project modifications that cannot be uploaded online to any PLC, such as the inclusion of modules in a remote PROFIBUS, or the inclusion of a new remote PROFIBUS. In these cases, taking advantage of the redundancy of the PLC and the PROFIBUS network, such modifications can be made without interrupting process control, using the procedures described in the chapter Offline Program Download Without Interrupting Process Control. In these procedures, it is necessary to temporarily disable project synchronization, allowing one of the PLCs to operate with the new version of the project for a while, while the other PLC still operates with the old version of the project.

An NX3035 CPU has a non-volatile *Project Synchronization Disable* register that allows you to disable services for synchronizing the application project and the project archive. This register can be adjusted with the Mastertool programmer. Simply disable project synchronization on one of the two PLCs so that it no longer occurs.

To disable project synchronization, you must first connect to the desired PLC with the Mastertool software (see section Mastertool Connection with an NX3035 CPU in a Redundant CPU Setup).

Next, in the *Communication / Redundancy Configuration* menu, open the *Project Synchronization* combo box, which allows you to select one of the following two options:

- Enable
- Disable

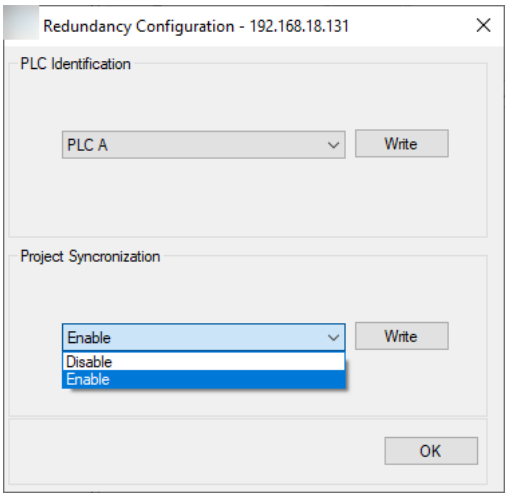


Figure 205: Disable or Enable Project Synchronization

Select the *Disable* option and then press the *Write* button next to this combo box. A message will inform you whether the operation was successful or not.

ATTENTION

The configuration for disabling project synchronization is not part of the redundant project developed with Mastertool. This configuration resides only in a non-volatile register of the NX3035 CPU, which can be read or written using Mastertool. Mastertool does not save this configuration in any file.

This setting is copied every MainTask cycle from the non-volatile register to the *DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag* diagnostic of the local PLC. This diagnostic will be TRUE when project synchronization is disabled on the local PLC. The user can check this diagnostic in the local PLC to verify that the command was successful, provided that this PLC is in RUN mode. If the PLC is not in RUN mode, this setting can be checked directly on the NX3035 CPU display (see section Redundancy Diagnostics on the NX3035 CPU Graphical Display).

It is also possible to observe the project synchronization enablement of the other half-cluster (remote PLC) through the *DG_NX3035.tRedundancy.RedDgnRem.sGeneral_Diag.bProjectSyncDisable* diagnostic (provided that the remote PLC is in RUN mode and one of the NETA or NETB synchronization channels is functioning correctly).

For project synchronization not to occur, it only needs to be disabled on one of the two redundant PLCs. For project synchronization to occur, it must be enabled on both redundant PLCs.

6.3.9. PROFIBUS Network Configuration

Up to six NX5001 PROFIBUS master modules can be installed in each half-cluster. Up to three redundant PROFIBUS networks or up to six single PROFIBUS networks can be configured. Other combinations with six NX5001 modules are also possible, such as two redundant networks and two single networks.

Figure 198 shows an example with a single PROFIBUS network (PROFIBUS 1), which is redundant (PROFIBUS 1A and PROFIBUS 1B).

Each PROFIBUS network can be configured in two different modes:

- **Vital failure:** if this network fails completely, this failure may cause a transition to redundancy mode in the redundant PLC (switch-over). In the case of a redundant PROFIBUS network, a complete failure consists of the failure of both networks that comprise it.
- **Non-critical failure:** even if this network fails completely, this failure will not cause a transition to redundancy mode in the redundant PLC (switch-over).

Only certain types of remote stations can be connected directly to a redundant PROFIBUS network:

- PO5065: PROFIBUS DP-V1 slave with HART for Ponto Series remote stations.
- NX5210: PROFIBUS DP-V1 slave with HART for Nexto Series remotes.

These topics have already been covered in greater detail in the section PROFIBUS Network Configuration with NX5001 Modules.

6.3.10. Redundant Ethernet Networks with NIC Teaming

Figure 198 shows several examples of redundant Ethernet networks with NIC Teaming:

- The connection of the NX3035 CPU via NET1 and NET2 to a redundant supervision network (Supervision Ethernet A and B).
- The connection of the NX3035 CPU via NET3 and NET4 to a redundant HSDN network (HSDN A and B).
- The connection of the NX3035 CPU via NET5 and NET6 to a redundant general-purpose Ethernet network (redundant Ethernet 1A and 1B).
- The connection of a pair of NX5000 interfaces to a redundant Ethernet network for general use (redundant Ethernet 2A and 2B).

Further details on configuring NIC Teaming ports are provided in the section Integrated Ethernet Interfaces Configuration and Configuration of Ethernet Interfaces with NX5000 Modules.

6.3.11. IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup

6.3.11.1. IP Address Configuration

In a redundant PLC, there are some differences in the IP address configurations of the integrated ports (NET1 ... NET6) of the NX3035 CPU, compared to a non-redundant PLC.

In a redundant Nexto Series cluster, it is necessary to configure three IP addresses for each of the Ethernet ports of the NX3035 CPU (NET1 to NET6), as shown in the following figure:

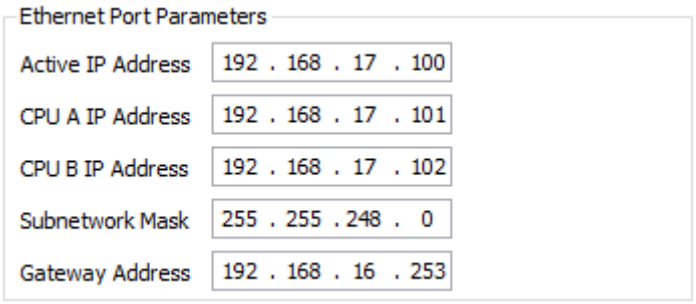


Figure 206: IP Addresses of the Ethernet Interfaces NET1 to NET6 of the NX3035 CPU

- **Active IP Address:** this address applies to the half-cluster that is currently in an active state. It can be used, for example, by a SCADA system to read and write data from the controlled process. Thanks to the Active IP Address, the SCADA system does not need to worry about changing the IP address in case of switchover between the PLCA and PLCB half-clusters.
- **CPU A IP Address:** this IP address always applies to the PLCA half-cluster. It should be used, for example, by a SCADA system to read diagnostics or write private commands to the PLCA.
- **CPU B IP address:** this IP address always applies to the PLCB half-cluster. It should be used, for example, by a SCADA system to read diagnostics or write private commands to the PLCB.

While the PLCA is in active state:

- The Ethernet port of CPU A will have two simultaneous IP addresses: *IP address of CPU A* and *Active IP address*.
- The Ethernet port of CPU B will have a single IP address: *IP address of CPU B*.

While the PLCB is active:

- The Ethernet port of CPU A will have a single IP address: *IP address of CPU A*.
- The Ethernet port of CPU B will have two simultaneous IP addresses: *IP Address of CPU B* and *Active IP Address*.

6.3.11.2. Ethernet Port Mode Configuration

For the integrated Ethernet ports of the NX3035 CPU, this subject was covered in Section Ethernet Port Mode Configuration.

For the remote Ethernet ports of the NX5000 modules, this subject was covered in Section Ethernet Port Mode Configuration for the NX5000.

6.3.11.3. Ethernet Port Failure Mode Configuration

For the integrated Ethernet ports of the NX3035 CPU, this subject was covered in Section Ethernet Port Failure Mode Configuration.

For the remote Ethernet ports of the NX5000 modules, this subject was covered in Section Ethernet Port Failure Mode Configuration.

6.3.12. IP Address Configuration on NX5000 Modules in a Redundant CPU Setup

In a redundant PLC, there are some differences in the IP address configurations of the NX5000s compared to a non-redundant PLC.

Unlike the Ethernet ports of the NX3035 CPU (NET1 ... NET6) that support two simultaneous IP addresses (see previous section), the Ethernet port of an NX5000 supports a single IP address. For this reason, four different IP switching methods have been developed for the NX5000. These methods, described below, are: Fixed IP, Exchange IP, Active IP and Multiple IP.

6.3.12.1. Fixed IP

This method can be configured on the Ethernet interfaces of the NX5000 Ethernet modules, as shown in the following figure:

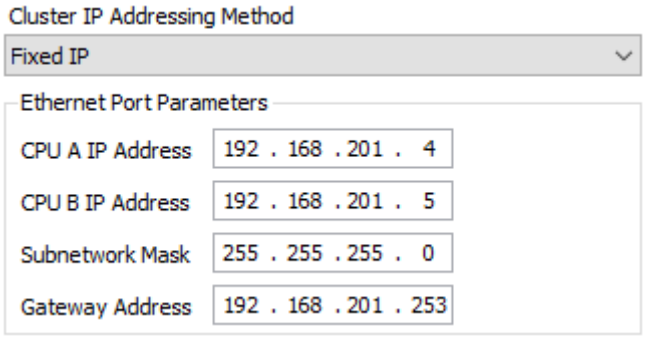


Figure 207: Fixed IP Method

In this simple method, the IP address of each half-cluster does not depend on its redundancy status (Active or Inactive):

- The PLCA's NX5000 will always have the *IP address of CPU A*.
- The PLCB's NX5000 will always have the *IP address of CPU B*.

6.3.12.2. Exchange IP

This method can be configured on the Ethernet interfaces of the NX5000 Ethernet modules, as shown in the following figure:

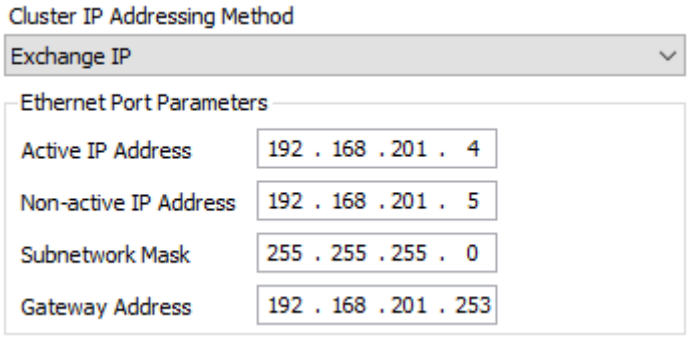


Figure 208: Automatic IP Switching Method

In this method, the IP address of each half-cluster depends on its redundancy status (Active or Inactive):

- The PLC's NX5000 in Active status will have the *Active IP Address*.
- The NX5000 of the PLC in the Inactive state will have the *Inactive IP Address*.

While the PLCA is in Active state:

- The PLCA's NX5000 will have the *Active IP Address*.
- The PLCB's NX5000 will have the *Inactive IP Address*.

While the PLCB is in active state:

- The PLCA's NX5000 will have the *Inactive IP Address*.
- The PLCB's NX5000 will have the *Active IP Address*.

ATTENTION

In this addressing method, the Ethernet ports of both PLCs (PLCA and PLCB) will assume the same *Inactive IP Address* while both PLCs are in the Inactive state, causing an addressing conflict on the network. Considering that this is an unusual and temporary situation (e.g., simultaneous power-on of both half-clusters), where no PLC is controlling the system, this does not constitute a serious problem.

6.3.12.3. Active IP

This method can be configured on the Ethernet interfaces of the NX5000 Ethernet modules, as shown in the following figure:

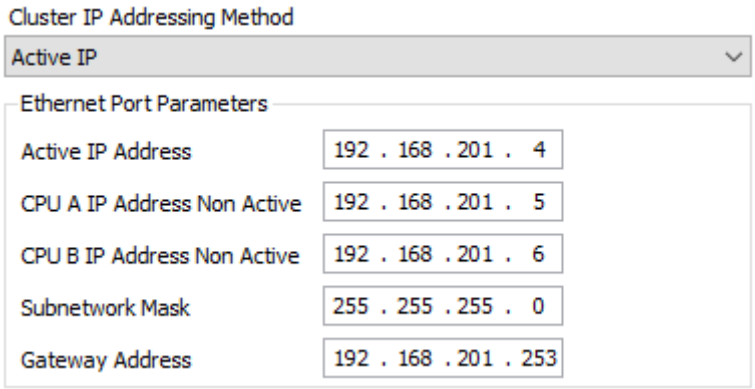


Figure 209: Active IP Method

In this method, the IP address of each half-cluster depends on its redundancy state (Active or Inactive):

- The NX5000 of the PLC in Active state will have the *Active IP Address*.
- The NX5000 of the PLCA, in the Inactive state, will have the *Inactive CPU A IP Address*.
- The NX5000 of the PLCB, in the Inactive state, will have the *Inactive CPU B IP Address*.

While the PLCA is in the active state:

- The PLCA's NX5000 will have the *Active IP Address*.
- The PLCB's NX5000 will have the *Non-Active CPU B IP Address*.

While PLCB is active:

- The PLCA's NX5000 will have the *CPU A IP Address Not Active*.
- The PLCB's NX5000 will have the *IP Address Active*.

6.3.12.4. Multiple IP

This method can be configured on the Ethernet interfaces of the NX5000 Ethernet modules, as shown in the following figure:

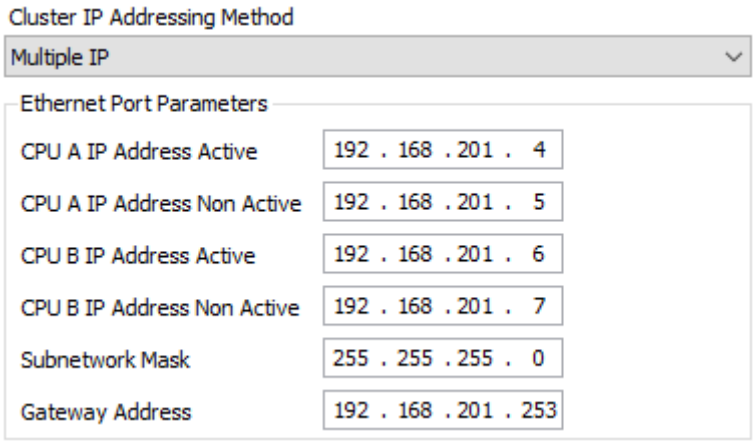


Figure 210: Multiple IPs Method

In this method, the IP address of each half-cluster depends on its redundancy state (Active or Inactive):

- The PLCA's NX5000, in Active state, will have the *IP Address of Active CPU A*.
- The PLCA's NX5000, in the Inactive state, will have the *IP Address of CPU A Inactive*.
- The PLCB's NX5000, in Active state, will have the *Active CPU B IP Address*.
- The PLCB's NX5000, in Inactive state, will have the *Inactive CPU B IP Address*.

While the PLCA is in the active state:

- The PLCA's NX5000 will have the *IP Address of CPU A Active*.
- The PLCB's NX5000 will have the *IP Address of CPU B Not Active*.

While PLCB is in an active state:

- The PLCA's NX5000 will have the *IP Address of CPU A Not Active*.
- The PLCB's NX5000 will have the *IP Address of CPU B Active*.

6.3.13. Combined Use of NIC Teaming and Switch Configurations with Active IP

When a given pair of ports is configured as a NIC Teaming or Switch pair on a redundant CP, the high availability provided by the NIC Teaming or Switch/RSTP configurations is combined with the convenience of the Active IP strategy, which eliminates the need for scripts in SCADA systems or other clients connected to servers running on the Active CP. The Switch/No Protection configuration can also be used, although it provides lower reliability.

For example, if NET 1 and NET 2 on the NX3035 CPU form a pair configured for NIC Teaming or Switch mode, then:

- **CPA IP Address:** IP address assigned to the NET 1 + NET 2 ports of the NX3035 CPU in CPA
- **CPB IP Address:** IP address assigned to the NET 1 + NET 2 ports of the NX3035 CPU in CPB
- **Active IP Address:** IP address assigned to the NET 1 + NET 2 ports of the NX3035 CPU currently operating as the Active CP

6.3.14. Ethernet Interfaces Use with Vital Fault Configuration

The Ethernet ports on the CPU (NET1 through NET6) and on NX5000 modules can be configured to generate Vital Failures. This option is important for applications in which I/O modules are distributed over an Ethernet network. In such cases, if a Vital Failure occurs on an Ethernet port, it will trigger a switchover. This behavior applies only to Ethernet ports that have at least one communication driver capable of generating a Vital Failure.

The communication drivers that can generate a Vital Failure are the MODBUS Client and the MODBUS Symbol Client (all references to the MODBUS Client in the following sections apply to both drivers). Therefore, if an Ethernet port has a MODBUS Client driver configured and an Ethernet port failure occurs, a switchover will be triggered if the Vital Failure option is enabled.

For a failure to be considered a Vital Failure on an Ethernet port used by a MODBUS Client, all servers configured in the driver must be in a fault condition. If more than one MODBUS Client driver is configured on the same Ethernet port, a Vital Failure is considered to have occurred only when all servers associated with all configured MODBUS Clients are in a fault condition.

MODBUS Client drivers can also be created under paired Ethernet ports using configurations such as NIC Teaming and Switch. With NIC Teaming and Switch/RSTP configurations, system reliability is increased because the likelihood of a Vital Failure (loss of communication with all servers) is reduced. The Switch/No Protection configuration can also be used, although it provides lower reliability in this scenario.

6.3.14.1. Rapid Vital Failure Detection Through Link Loss on Ethernet Ports

On the Active CP, a link loss on a single Ethernet port, or on both Ethernet ports of a pair configured with NIC Teaming or Switch, can be detected quickly and trigger a switch-over to the Inactive state. In this case, it is known in advance that the Active CP will lose communication with all servers connected through the affected Ethernet port(s), making it unnecessary to wait for communication timeouts with the servers (see the next section).

On the Standby CP, or on a CP in the Initializing state, such link losses also trigger a switch-over to the Inactive state. The CP must not assume or remain in the Standby state because, if it were switched to the Active state, it would be unable to communicate with the servers connected through the affected Ethernet port(s).

6.3.14.2. Vital Failure Detection Through Communication Failure with All MODBUS Servers

In some cases, an Ethernet port with one or more MODBUS client drivers may lose communication with all connected MODBUS servers, even if the port itself has not experienced the link loss conditions described in the previous section.

In this situation, each MODBUS client will activate its *bAllDevicesCommFailure* diagnostic (see Section MODBUS Diagnostics Used in Redundancy) if it loses communication with all of its MODBUS servers. The time required to detect this communication loss depends on the timeout values configured in the MODBUS client.

After all MODBUS clients associated with the Ethernet port have activated their *bAllDevicesCommFailure* diagnostic, the Active CP will wait an additional 3 seconds before switching to the Inactive state.

6.3.14.3. Keep-Alive Messages via NET1 and NET2 Between the Redundant CPs

This section describes the keep-alive messages that can be exchanged between the two redundant CPs through the NET1 and/or NET2 ports.

For keep-alive messages to be exchanged through NET1 and/or NET2, all of the following conditions must be met:

- No PROFIBUS network must be configured (that is, no NX5001 master may be present).
- The NET1/NET2 ports of both redundant CPs must be connected to the same Ethernet network so that a communication path is available for transmitting these messages.

Under these conditions, keep-alive messages are exchanged between the two redundant CPs during each MainTask cycle. Among other information, these messages contain the redundancy status of the transmitting CP.

If NET1 and NET2 are configured as a NIC Teaming or Switch pair, keep-alive messages are transmitted and received only through the port that is currently active (NIC Teaming) or unblocked (Switch). If the ports are configured as Single (non-redundant), independent keep-alive messages are exchanged over both NET1 and NET2.

The primary purpose of keep-alive messages is to detect that the other CP is present and executing its application, as well as to determine its redundancy status, which is particularly useful in the event of a complete failure of the synchronization channels (NETA / NETB). This mechanism prevents both CPs from simultaneously entering the Active state under certain conditions. When a PROFIBUS network is configured, this mechanism is unnecessary and therefore disabled, since a similar mechanism for detecting the other CP is already provided by the PROFIBUS masters (NX5001).

Keep-alive messages will be exchanged through NET1/NET2 even if NET1/NET2 are not-configured for vital failure detection. Note that other Ethernet ports (NET3 ... NET6, NX5000) may still be configured for vital failure detection.

Each CP provides diagnostics to indicate whether it is receiving keep-alive messages from the other CP (see Section Redundancy Diagnostics):

- **RedDgnLoc.sGeneral_DiagExt.bRemCpuKeepAliveNet1:** Indicates that keep-alive messages are being received through NET1 (or through NET2 if NET1/NET2 are configured as a NIC Teaming or Switch pair). It is TRUE when messages are being received, and FALSE otherwise.
- **RedDgnLoc.sGeneral_DiagExt.bRemCpuKeepAliveNet2:** Indicates that keep-alive messages are being received through NET2. It is TRUE when messages are being received, and FALSE otherwise. It is always FALSE if NET1/NET2 are configured as a NIC Teaming or Switch pair.

It is important to configure the diagnostics described above as alarms to indicate when maintenance is required. For example, if the non-active CP is not receiving keep-alive messages through NET1/NET2, immediate maintenance can help prevent the following issues:

- Prevent this CP from incorrectly entering the Active state after power-up or in other situations.
- If MODBUS client drivers connected through NET1/NET2 are configured for vital failure detection, this condition may indicate that the Standby CP will be unable to communicate with the MODBUS servers if it is switched to the Active state.

6.3.14.4. Using Additional Diagnostics for Vital Failure Maintenance

In addition to the keep-alive message reception diagnostics described in the previous section, it is recommended that the following diagnostics also generate alarms to trigger immediate maintenance:

- Link loss diagnostics for individual Ethernet ports (NET1 ... NET6, NX5000) on each PLC. These diagnostics are not redundant and should therefore be monitored on both CPs (Active and Non-Active).
- Communication failure diagnostics for each MODBUS client mapping. These diagnostics should only be considered on the CP that is currently in the Active state.
- General communication failure diagnostics indicating that a MODBUS client has lost communication with all of its servers. These diagnostics should only be considered on the CP that is currently in the Active state.

6.3.14.5. Specific Behavior of the Active and Non-Active CPs

On Ethernet ports configured for vital failure detection, only the Active CP communicates with the MODBUS servers through the MODBUS client drivers. The MODBUS client drivers remain inactive on the Non-Active CP.

The Non-Active CP only transmits keep-alive messages, and only through NET1/NET2 when no PROFIBUS masters are configured in the project, as described previously.

The Standby CP may switch to the Inactive state if it detects a link loss on Ethernet ports configured for vital failure detection.

6.3.15. Use of OPC DA Communication with Redundant Projects

The OPC DA protocol can be configured for redundant cluster communication with SCADA systems. When this option is selected when creating a redundant project, the *Symbol Configuration* object is added to the project. In this object, the system variables that will be sent to the SCADA system are configured. This communication option is enabled on the Ethernet ports of the NX3035 CPU.

For more information about configuring OPC DA communication with redundant projects, see the section Configuration with the PLC on the OPC DA Server with Connection Redundancy in this Manual.

6.3.16. Redundant CPU States

In a redundant system, a PLC or half-cluster (PLCA or PLCB) can assume the following states:

- Active
- Standby
- Inactive
- Not-Configured
- Initializing

ATTENTION

This manual will often use the designation “Non-Active” for any state other than Active, that is, to designate any of the other four states (Standby, Inactive, Not-Configured, and Initializing).

These five states are briefly described below. Further details on the states of a redundant PLC, and on the transitions between them, will be covered in section Redundancy State Machine.

6.3.16.1. Active

In this state, the PLC controls the automated process using the *ActivePrg* program, which runs only in this state. The Active PLC also updates the PROFIBUS remote I/O system, putting its PROFIBUS masters into active mode to establish communication with the PROFIBUS remotes (slaves).

6.3.16.2. Standby

In this state, the PLC is ready to switch to the Active state if there is a demand for it, such as a failure in the Active PLC. In addition, it checks its own integrity and can switch to the Inactive or Unconfigured state in case of a failure.

6.3.16.3. Inactive

This state is reached after the detection of a fault that requires maintenance. It can also be reached due to a manual request from the user to perform scheduled maintenance.

6.3.16.4. Not-Configured

This is the initial redundancy state. By convention, it is also assumed that the PLC is in this state while it is powered off.

6.3.16.5. Initializing

Unlike all other four states, which can have an indefinite duration, the Initializing state lasts only a few seconds. This state is always reached from the Unconfigured state.

Upon entering the Initializing state, several actions, tests, and checks are performed to decide what the next state will be.

6.3.17. Redundancy State Machine

The following figure shows the redundancy state machine, illustrating all redundancy states and all possible transitions between them.

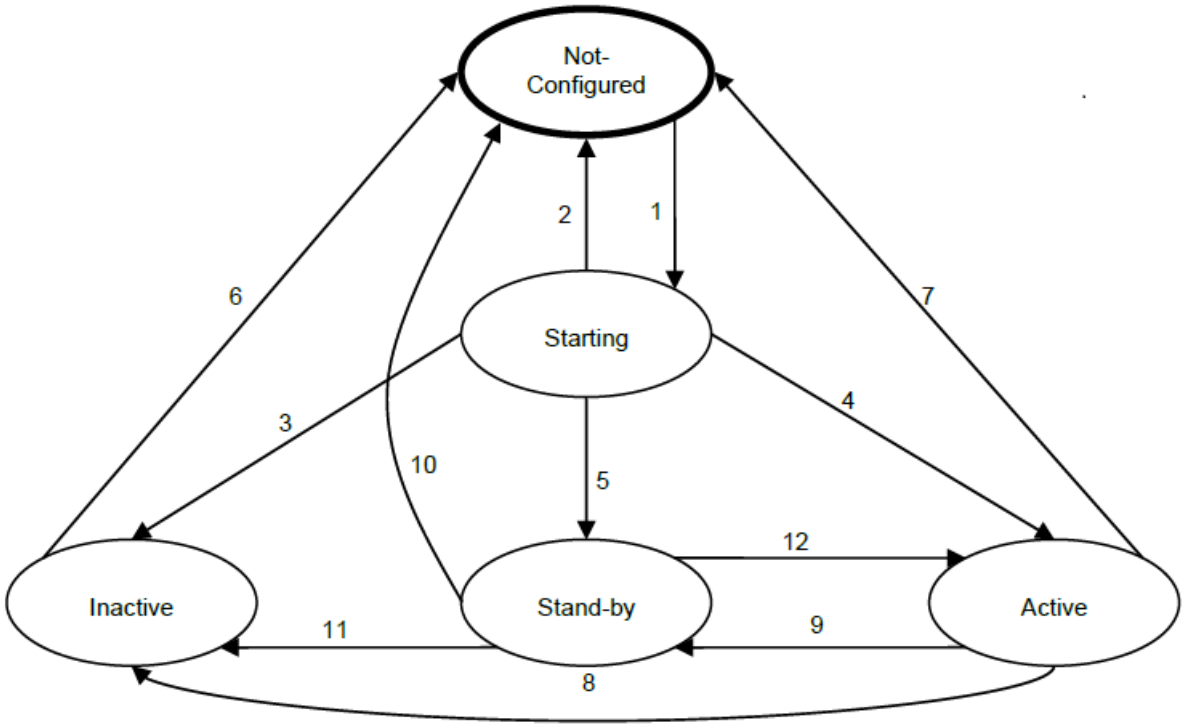


Figure 211: Redundancy State Machine

The following subsections describe the behavior of each state and the causes of transitions from this state to other states.

To correctly interpret the operation of this state machine, it is necessary to establish some rules and sequences:

- Transitions originating from the same state must be evaluated in the sequence given by the transition number. For example, transitions 2, 3, 4, and 5 originate from the Initializing state. In this example, transition 2 is evaluated first, then 3, then 4, and finally 5. If transition 2 is triggered, transitions 3, 4, and 5 will not be evaluated.
- Transitions can only be triggered if the PLC is energized and the MainTask is running. Otherwise, it is assumed that the PLC is in the Unconfigured state.

6.3.17.1. Not-Configured State

This is the initial state of the state machine.

The PLC assumes this state whenever the MainTask task is not running, due to reasons such as:

- The PLC is turned off (unpowered).
- The PLC was recently turned on, but has not yet started executing the MainTask.
- A STOP command was assigned by the user via the Mastertool programmer.
- A reset command was assigned by the user via the Mastertool programmer (Hot Reset, Cold Reset, or Origin Reset).
- A watchdog or other exception condition caused MainTask to stop.

When the MainTask is running, the Not-Configured state can be reached from the other 4 states, as shown in the state machine in Figure 211.

During Not-Configured state:

- The PROFIBUS NX5001 masters are disabled.
- Cyclic synchronization services are performed (see section Cyclic Synchronization Services through NETA and NETB), provided that the conditions for their execution are present.
- Sporadic synchronization services may also be executed (see section Sporadic Synchronization Services through NETA and NETB) provided that there is a need to execute them and that the conditions for their execution are present. This occurs particularly when the other CP is in the Active state and project synchronization is required.

WARNING

If a CP in the not-configured state receives a project synchronization from the other CP in the Active state, and the received project is not compatible with this CP's firmware version, the CP will remain in the not-configured state, continuously requesting a new project synchronization. To resolve this issue, update the firmware of this CP. It is recommended that both CPs always be kept at the same firmware version.

6.3.17.1.1. Transition 1 – Not-Configured to Initializing

This transition occurs as soon as all of the following conditions are met:

- The MainTask must be running (CP in RUN mode). For example, if the CP has been placed in STOP mode, it must be switched back to RUN mode.
- If the other CP is in the Active state and project synchronization is enabled (see Section Project Synchronization Disabling), and there is a difference between the projects on this CP and the Active CP, a project synchronization will be performed from the Active CP to this CP. Wait until this synchronization completes successfully. After that, the project received from the Active CP must also be compatible with this CP's firmware version.

6.3.17.2. Initializing State

Unlike the other four states, which can have an indefinite duration, the Initializing state lasts only a few seconds. This state is always reached from the Unconfigured state through Transition 1.

Upon entering the Initializing state, several actions, tests, and checks are performed to decide what the next state will be.

During the Initializing state, the PLC:

- Checks if the PLC identification is correct (it must be PLCA or PLCB).
- Checks if there are problems in the configuration parameters extracted from the Mastertool project.
- Cyclic synchronization services are performed (see section Cyclic Synchronization Services through NETA and NETB), provided that the conditions for their execution are present.
- Checks the compatibility of firmware versions between the two PLCs. Considering that a version has the format X.Y.Z.W, two versions will be compatible if the digit X of both is the same.
- Checks the equality of projects between the two PLCs, if automatic project synchronization is enabled (see section Project Synchronization Disabling).
- Enables its PROFIBUS masters (NX5001) in passive state and attempts connections with the respective NX5001 masters of the other PLC. Each connection will be successful if the other PLC is in Active state and there are no problems in the NX5001 PROFIBUS transmission and reception circuits or in the physical medium.

6.3.17.2.1. Transition 2 – Initializing to Not-Configured

- This PLC has been shut down or reset (Hot Reset, Cold Reset, or Source Reset) or its CPU has entered STOP Mode.
- The identification record for this PLC is invalid (different from PLCA or PLCB).
- There are logical configuration errors in the project received from Mastertool.
- The other PLC is in Active state, and the firmware version of this PLC is incompatible with the firmware version of the Active PLC. Considering that a version has the format X.Y.Z.W, two versions will be compatible if the digit X of both is the same.
- The other PLC is in Active status, and the project for this PLC is different from the project for the Active PLC, and automatic project synchronization is enabled (see section Project Synchronization Disabling).

6.3.17.2.2. Transition 3 – Initializing to Inactive

- The other PLC is in Active state, but it is not possible to synchronize the redundant data or the redundant force list.
- The status of the other PLC cannot be discovered via the synchronization network (NETA/NETB), but some NX5001 of this PLC, in passive mode, can detect traffic on some of the PROFIBUS networks. Thus, this PLC assumes that the other PLC is in Active status and controlling the process, even though NETA / NETB are not working to verify this.
- The other PLC is in Active state. In addition, for some PROFIBUS network configured in vital failure mode, all NX5001s of this PLC, in passive mode, cannot detect traffic on this PROFIBUS network. This indicates a failure in all NX5001s of this PLC on this PROFIBUS network configured in vital failure mode.
- A complete link loss is detected on an Ethernet interface configured for vital failure detection. A complete link loss refers to the single port operating in Single mode, or both ports of a pair operating in NIC Teaming or Switch mode.
- The state of the other CP cannot be determined through the synchronization network (NETA / NETB) in a project that does not include NX5001 PROFIBUS masters. In addition, an Ethernet port (NET1 or NET2) on this CP, operating in passive mode, is receiving keep-alive messages from the other CP indicating that this CP is in the Active state.

6.3.17.2.3. Transition 4 – Initializing to Active

- The other PLC is in a Non-Active state. Before making this transition, this condition must remain true for a few seconds. When PLCA and PLCB are powered up simultaneously, the PLC that finishes system initialization first will assume the Active state.
- The state of the other CP cannot be determined through the synchronization network (NETA / NETB). In addition, one of the following conditions must remain true for a certain period before transitioning to the Active state:
  - NX5001 modules are present in the project, but these NX5001 modules, operating in passive mode, are not detecting traffic on their respective PROFIBUS networks. Therefore, the other PLC is assumed to be powered off or not executing.
  - No NX5001 modules are present in the project, and this CP is not receiving keep-alive messages through NET1 and/or NET2. Therefore, the other PLC is assumed to be powered off or not executing.
  - There is no NX5001 in the project, and this CP is receiving keep-alive messages via NET1 and/or NET2 indicating that the other CP is in a state other than Active.

6.3.17.2.4. Transition 5 – Initializing to Standby

- The other PLC is in Active state and the redundant data synchronization and redundant force list synchronization services are functioning correctly.

6.3.17.3. Inactive State

This state is reached after detecting a fault that requires maintenance.

It can also be reached due to a manual request from the user to perform scheduled maintenance. The manual command before starting scheduled maintenance can be, for example:

- RedCmdLoc.bInactiveLocal, executed on this PLC
- RedCmdLoc.bInactiveRemote, executed on the other PLC
- Command via OTD button, executed on this PLC

To exit the Inactive state, after correcting the faults that led to this state, or after performing scheduled maintenance, the user must execute a manual command, which can be, for example:

- RedCmdLoc.bStandbyLocal, executed on this PLC
- RedCmdLoc.bStandbyRemote, executed on the other PLC
- Command via OTD button, executed on this PLC

Instead of executing the manual command mentioned above, it is also possible to turn the PLC off and on again, or use a reset command (Hot Reset, Cold Reset, or Origin Reset), or put the PLC in STOP mode and then in RUN mode.

6.3.17.3.1. Transition 6 – Inactive to Not-Configured

- This PLC was shut down or reset (Hot Reset, Cold Reset, or Source Reset) or its CPU entered Stop Mode.
- A manual command was executed to exit the Inactive state, for example:
  - RedCmdLoc.bStandbyLocal, executed on this PLC
  - RedCmdLoc.bStandbyRemote, executed on the other PLC
  - Command via OTD button, executed on this PLC, to go to Standby state

6.3.17.4. Active State

In this state, the PLC controls the automated process using the ActivePrg program, which runs only in this state. The Active PLC also updates the PROFIBUS remote I/O system by putting its PROFIBUS NX5001 masters into active mode, which serves to establish communication with the PROFIBUS remotes (slaves).

The Active PLC also checks its internal diagnostics and user switch-over requests to determine if a switch-over is necessary. The PLC will normally only exit the Active state if it knows that the other PLC is in the Standby state and therefore able to assume the Active state.

However, there are some situations in which the Active PLC may exit the Active state even without being certain that the other PLC is in the Standby state, for example:

- If this PLC is shut down
- If this PLC is put into STOP mode
- If a reboot command occurs on this PLC (Hot Reset, Cold Reset, or Source Reset)

6.3.17.4.1. Transition 7 – Active to Not-Configured

- This PLC has been shut down or reset (Hot Reset, Cold Reset, or Source Reset) or its CPU has entered Stop Mode.

6.3.17.4.2. Transition 8 – Active to Inactive

- This PLC, with its NX5001 in active mode, cannot communicate with any of the PROFIBUS slaves on a PROFIBUS network configured in vital mode. In addition, this PLC knows that the other PLC is in Standby state. This condition is not evaluated in the first 2 seconds in Active state.
- This PLC detected a total failure in the Ethernet networks configured in vital failure mode, and knows that the other PLC is in the Standby state.

6.3.17.4.3. Transition 9 – Active to Standby

- Both PLCs, for some reason, are in the Active state, and this conflict must be resolved. The PLCA will switch to the Standby state if this conflict lasts for a certain amount of time, and the PLCB will do the same after a different amount of time, which is shorter than the PLCA's time. Thus, in case of conflict, the PLCA has priority to remain in the Active state. If the status of the other PLC is unknown due to a failure in the synchronization network (NETA/NETB), the Active PLC will not go into Standby state.
- The other PLC is in Standby state, and this PLC has received a manual command to go into Standby state. This condition is not evaluated in the first 2 seconds in Active state. The manual command to go to the Standby state can be, for example:
  - RedCmdLoc.bStandbyLocal, executed on this PLC
  - RedCmdLoc.bStandbyRemote, executed on the other PLC
  - Command via OTD button, executed on this PLC

6.3.17.5. Standby State

In this state, the PLC is ready to switch to the Active state if there is a demand for it, such as a failure in the Active PLC.

The Standby PLC also checks its own diagnostics and may switch to the Unconfigured or Inactive state depending on certain failures.

NX5001 PROFIBUS masters are enabled in passive mode. Passive mode is used to test the PROFIBUS transmission and reception circuits, as well as the physical medium, in order to prevent hidden failures. A complete failure of a PROFIBUS network configured for vital failure detection causes a transition to the Inactive state. A complete PROFIBUS network failure corresponds either to both networks that make up a redundant PROFIBUS network or to the single network in a non-redundant PROFIBUS configuration.

If Ethernet ports configured for vital failure detection contain MODBUS client drivers, a complete failure on these ports also causes a transition to the Inactive state. A complete failure corresponds either to both ports of a pair configured in NIC Teaming or Switch mode, or to the single port in a Single configuration. In the Standby state, the MODBUS drivers are disabled. Therefore, the only type of failure that can be detected is link loss. In this case, a complete failure corresponds to a link loss on the single port (Single configuration) or a link loss on both ports of a pair (NIC Teaming or Switch configuration).

6.3.17.5.1. Transition 10 – Standby to Not-Configured

- This PLC has been shut down or reset (Hot Reset, Cold Reset, or Source Reset), or its CPU has entered Stop Mode.
- The other PLC is in Active state and the project of this PLC is different from the project of the Active PLC, and automatic project synchronization is enabled (see section Project Synchronization Disabling).
- The other PLC is in Active state, and the firmware version of this PLC is incompatible with the firmware version of the Active PLC.

6.3.17.5.2. Transition 11 – Standby to Inactive

- A manual command to switch to the Inactive state was executed. For example:
  - RedCmdLoc.bInactiveLocal, executed on this PLC
  - RedCmdLoc.bInactiveRemote, executed on the other PLC
  - Command via OTD button, executed on this PLC

  This is typically done to perform scheduled maintenance on the Non-Active PLC. Scheduled maintenance should be avoided on the Standby PLC, so it is advisable to switch it to the Inactive state first.

- The other PLC is in Active state. However, the redundant data synchronization service has not worked correctly in the last four MainTask cycles, or the redundant diagnostics synchronization service has not worked correctly in the last two MainTask cycles.
- The other PLC is in Active status. However, this PLC cannot monitor PROFIBUS traffic on all NX5001s in a PROFIBUS network configured in vital failure mode.
- The other CP is in the Active state. However, this CP has detected link loss failures on Ethernet ports configured for vital failure detection.
- The state of the other CP is unknown due to failures in the synchronization network (NETA / NETB), and the project contains NX5001 PROFIBUS masters. In addition, this CP is able to monitor PROFIBUS traffic on at least one of its NX5001 modules operating in passive mode, on either PROFIBUS network. This indicates that the other CP is still in the Active state.
- The state of the other CP is unknown due to failures in the synchronization network (NETA / NETB), and the project does not contain NX5001 PROFIBUS masters. In addition, this CP is receiving keep-alive messages from the other CP through NET1 and/or NET2 indicating that the other CP is still in the Active state.

6.3.17.5.3. Transition 12 – Standby to Active

- The status of the other PLC is known and different from Active.
- The state of the other CP is unknown due to failures in the synchronization network (NETA / NETB), and the project contains NX5001 PROFIBUS masters. In addition, this CP is unable to monitor PROFIBUS traffic on any of its NX5001 modules operating in passive mode, on either PROFIBUS network. This indicates that the other CP is not in the Active state.
- The state of the other CP is unknown due to failures in the synchronization network (NETA / NETB), and the project does not contain NX5001 PROFIBUS masters. In addition, this CP is not receiving keep-alive messages from the other CP through NET1 and/or NET2. Then it assumes that the other CP is not in the Active state.

6.3.18. First Instants in Active State

During the first 2 seconds in the Active state, several transitions that would normally take the PLC out of the Active state are not evaluated (see previous subsections that define transitions from the Active state).

Only two conditions allow the PLC to leave the Active state during these two seconds:

- This PLC has been turned off or reset (Hot Reset, Cold Reset, or Source Reset) or its CPU has entered Stop Mode, causing a transition to Unconfigured.
- Both PLCs, for some reason, are in the Active state, and this conflict must be resolved. The PLCA will switch to the Standby state if this conflict lasts for a certain amount of time, and the PLCB will do the same after a different amount of time, which is less than the PLCA time. Thus, in case of conflict, the PLCA has priority to remain in the Active state.

In addition, in the first moments that a PLC assumes the Active state, some diagnostics that are not redundant may not be valid, for example, the diagnostics of the NX5000 and NX5001 modules. The method for not considering these possibly "invalid" diagnostics is described in section Reading Non-Redundant Diagnostics.

6.3.19. Common Failures which Cause Automatic Switchovers between Half-Clusters

This section lists the most common failures that automatically cause a switchover from Active PLC to Non-Active PLC, and from Standby PLC to Active PLC. These failures trigger a subset of the transitions examined in the Redundancy State Machine section.

- Power failure in the Active PLC. It is important that both PLCs have redundant power systems so that a common power failure does not affect both.
- Failure of the Active PLC's NX8000 power supply.
- Failure in the Active PLC's backplane rack (NX9000, NX9001, NX9002, or NX9003).
- Failures in the Active PLC's NX3035 CPU, such as:
  - Watchdog.
  - Reboot (Hot Reset, Cold Reset, or Source Reset).
  - Stop.
- Total failure of a PROFIBUS network in the Active PLC, if this network is configured in vital mode. If the PROFIBUS network is redundant, both networks that comprise it must be in failure (double failure).

6.3.20. Switch-overs between Half-Clusters Managed by the User

Among the transitions examined in section Redundancy State Machine, some allow the user to manage switchovers between half-clusters due to failures that do not normally generate switchovers automatically.

There are very specific cases that depend on each customer's philosophy. Consider an example in which the SCADA system loses communication with the Active PLC, but can still communicate with the Standby PLC. The operator can send a command from SCADA to the Standby PLC, and the Standby PLC can then use the RedCmdLoc.bStandbyRemote command to request that the Active PLC go to the Standby state, causing a switchover between the Active and Standby PLCs.

Another example is illustrated in section Example of Managing a MODBUS TCP Client in a Redundant CPU. This example describes an automatically managed switch-over without any operator intervention. In this case, it is a MODBUS TCP client, whose communications with the server must be enabled only on the Active PLC. If the Active PLC loses all communications with the server, or has link-down problems, an automatic switchover should occur between the Active and Standby PLCs. This example also shows how to prevent hidden failures in the standby PLC, where communications with the server must be disabled.

Through data structures such as those mentioned in the section Data Structures for Diagnostics, Commands, and User Information, it is possible to exchange diagnostics and commands between half-clusters via NETA and NETB, allowing the user to perform special redundancy management for failures that would not normally cause switchovers. Further details on these data structures will be provided in section Structure of Diagnostics, Commands, and User Information for Redundancy.

6.3.21. Fault Tolerance

The main purpose of a redundant PLC is to increase system availability. Availability is the ratio between the time the system is functioning correctly and the total time since the system was implemented. For example, if a system was implemented 10 years ago, and during this period it was down for a total of one year due to failures, then its availability was only 90%. Availability levels of this order are generally unacceptable for critical systems, and values of 99.5% or even higher may be required for these systems.

To achieve availability of this order, several strategies are necessary:

- Use of more reliable components (with high MTBF, or mean time between failures), which will contribute to increasing the MTBF of the system as a whole.
- Use of redundancy at least for the most critical components or components with lower MTBF, so that the failure of one component can be tolerated without stopping the system. If redundancy is implemented through component duplication, both components must fail for the system as a whole to become unavailable.
- High diagnostic coverage, especially for redundant components. Component redundancy is of little use in increasing availability when it is not possible to detect that a redundant component has failed. In this case, the first failure in one of the components does not yet compromise the system, but because it remains hidden, a second failure will eventually occur that will compromise the system, since the first failure has not yet been repaired. Failures can be classified as diagnosable or hidden, and it is highly desirable that most failures of redundant components be diagnosable.
- It is also important that non-redundant components have comprehensive diagnostic coverage, as the system can often continue to function even with the failure of a non-redundant component. The component may not be in demand. For example, a relay with an open contact, whose coil is rarely activated, will not have its failure detected until the system requests its closure.
- Low repair time for non-redundant components. The failure of a non-redundant component can compromise the system, and during repair, the system will be unavailable.
- Possibility of repairing or replacing a redundant component without stopping the system. If this possibility exists, there is a significant increase in availability. Otherwise, a system shutdown must be scheduled to replace the component, and the repair time counts as downtime.
- Low repair time for redundant components. The failure of a redundant component does not compromise the system, but during its repair, a failure may eventually occur in its redundant pair. For this reason, it is important that the failure be repaired quickly after diagnosis. The longer the repair time, the greater the probability of a second failure in the redundant component during the repair of the first failure, which would compromise the system. Therefore, the longer the repair time, the lower the system availability.
- Schedule periodic offline tests on components to detect faults that cannot be automatically diagnosed by the system. The goal is to detect hidden faults, especially in redundant components or simple components that are not being requested (e.g., a safety relay). Offline tests sometimes involve system downtime, which reduces availability. Special occasions, such as scheduled plant maintenance shutdowns, are usually taken advantage of. The longer the period between offline tests, the longer a fault can remain hidden and, therefore, the greater the likelihood that this fault will compromise the system, i.e., the lower the system availability.

These principles were considered in the design of redundant PLCs with NX3035 CPUs.

The following subsections analyze various types of failures and how they are tolerated or not tolerated, and whether there are switchovers associated with tolerated failures.

6.3.21.1. Simple Failure with Unavailability

Some components, because they are not duplicated, cannot tolerate even a single failure without causing some type of unavailability. In a redundant PLC with an NX3035 CPU, these components are as follows:

- PROFIBUS remotes (slaves) in a non-redundant PROFIBUS network.
- Ethernet remotes (slaves) in a non-redundant Ethernet network.
- I/O modules.

The fault tolerance of a PROFIBUS network can be increased by opting for a redundant PROFIBUS network, which is recommended in systems that require high fault tolerance. Figure 198 shows an example of an architecture with a redundant PROFIBUS network.

In the same way, the fault tolerance of an Ethernet network can be increased by using a redundant Ethernet network with NIC Teaming configuration.

As for the unavailability of an I/O module, it should be noted that this does not constitute total system unavailability. It constitutes partial unavailability, only of the control loops that use this I/O module.

Although there is no provision for redundancy of I/O modules, the user application can manage it in special cases. For example, the user can insert three analog input modules into three different PROFIBUS remotes and implement a voting scheme between triple analog inputs for some critical system. Such solutions must be managed by the user. There is no automatic support for them. Such solutions generally also imply redundancy of sensors and actuators in the field.

6.3.21.2. Simple Failure without Unavailability Causing a Switchover

Some redundant components tolerate single failures without causing downtime, but cause switch-over:

- PLCA or PLCB half-cluster backplanes (NX9000, NX9001, NX9002, or NX9003)
- PLCA or PLCB power supplies (NX8000)
- PLCA or PLCB CPUs (NX3035)
- NX5001 modules (PROFIBUS masters) in non-redundant PROFIBUS network configuration
- NX5000 modules (Ethernet) in non-redundant configurations (without NIC Teaming)
- PROFIBUS slave interface of a redundant remote (PO5063V5, PO5065, NX5210, or AL-3416). In this case, unlike the previous ones, the switch-over takes place within the remote, between PROFIBUS networks A and B.

6.3.21.3. Double Failure without Unavailability Causing a Switchover

Some components are duplicated in each half-cluster, so 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 (user-managed switchover, or configured for vital failure detection).

6.3.22. Redundancy Overhead

A redundant application causes an increase in the processing time of an application when compared to the time required for an equivalent non-redundant application.

This additional time is mainly due to the execution of cyclic synchronization services, described in the section Cyclic Synchronization Services through NETA and NETB, in addition to a shorter time for redundancy management itself (state machine, etc.). The total additional time due to redundancy (redundancy overhead) is estimated by Mastertool and displayed in the Messages window after compiling the redundant PLC project.

In addition, it is up to the user to define an interval for MainTask sufficient to accommodate:

- The time required to execute the main POUs (NonSkippedPrg and ActivePrg). This time is normally measured after project development (discounting redundancy overhead).
- Some slack for executing other CPU tasks (operating system, PROFIBUS I/O drivers, MODBUS, etc.). The percentage of this slack may vary depending on the performance required for these tasks. For example, if MODBUS communication with the SCADA system needs to allocate a lot of processing power to achieve satisfactory performance, this slack should be increased.

6.4. Redundant CPU Programming

6.4.1. Wizard for a New Redundant Project Creation

To create a new redundant project, use the *File/New Project* command and select the *Standard Project*. The user must also provide the desired project name and the folder where it will be stored, as shown in the figure below.

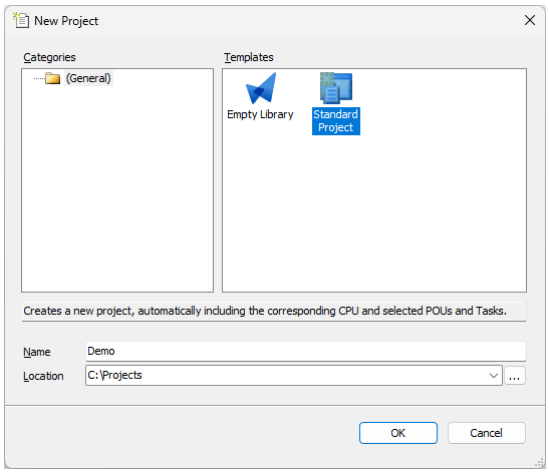


Figure 212: New Project

Next, the *Wizard* that generates the redundancy project asks the user a series of questions regarding the desired configuration, which must be answered throughout a sequence of screens.

The first screen of the *Wizard* is shown in the following figure, where the *NX3035 Redundant Mode* controller must be selected.

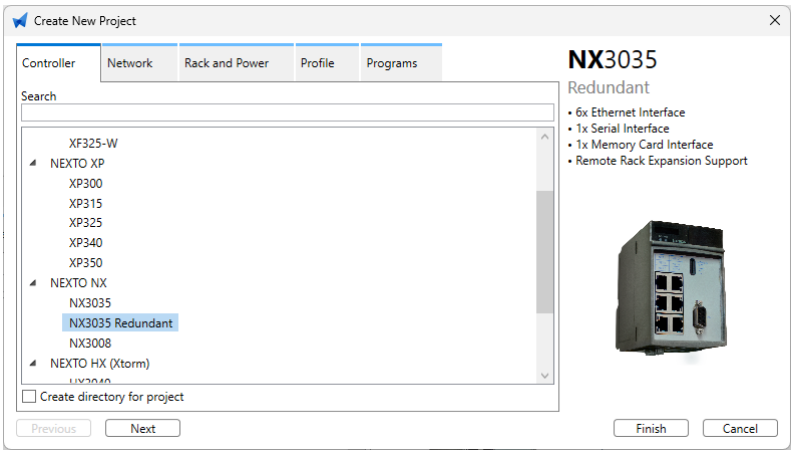


Figure 213: Tab - Controller Selection

Click the *Next* button to select the second tab of the Wizard. The second tab of the Wizard is shown in the following figure, already filled with some options compatible with NX3035 redundancy. This screen has two functions:

- Define the number of additional Ethernet networks (between 0 and 6) using NX5000 modules, and whether each of these networks is single (allocates one NX5000) or NIC Teaming (allocates two NX5000). In the following example, there is a single NIC Teaming network with failure mode disabled (allocates two NX5000). Note that there is a limit of a maximum of 6 NX5000 modules per half-cluster (that is, although it is possible to define up to 6 additional Ethernet networks, not all of them can be redundant, as this would require 12 NX5000 modules).
- Define the number of PROFIBUS networks (between 0 and 6), and whether each of them is single (allocates one NX5001) or redundant (allocates two NX5001). In the following example, the first two networks (PROFIBUS 1 and PROFIBUS 2) are redundant, and the third network (PROFIBUS 3) is single. All PROFIBUS networks created by the

Wizard have the vital fail mode enabled, meaning that a switchover will occur in case of failure in one of these networks. It is possible to disable this fail mode later on each network, if desired. Note that there is a limit of a maximum of 6 NX5001 modules per half-cluster (that is, although it is possible to define up to 6 PROFIBUS networks, not all of them can be redundant, as this would require 12 NX5001 modules).

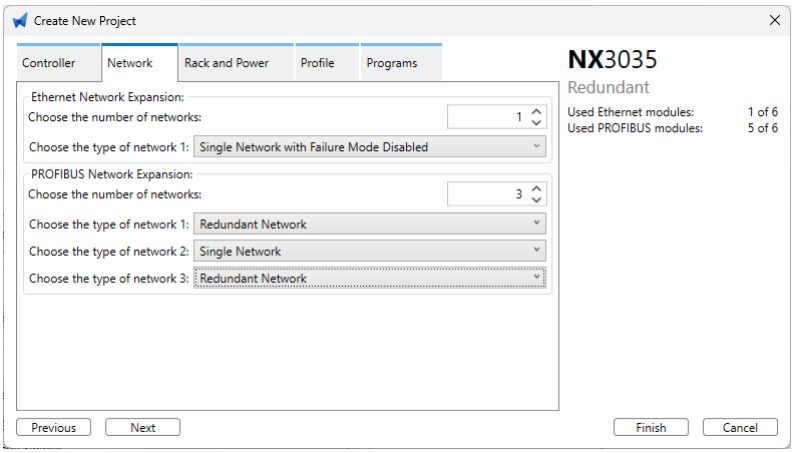


Figure 214: Tab - Network

Click the *Next* button to select the *Rack and Power* tab of the Wizard. This Wizard screen is shown in the following figure, already filled with some options compatible with NX3035 redundancy.

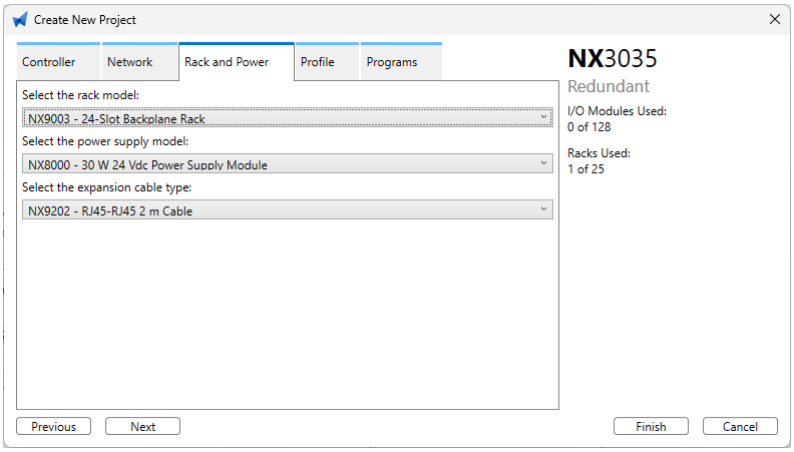


Figure 215: Tab - Rack and Power

Click the *Next* button to select the *Profile* tab of the Wizard. This Wizard screen is shown in the following figure, already filled with the only option compatible with NX3035 redundancy.

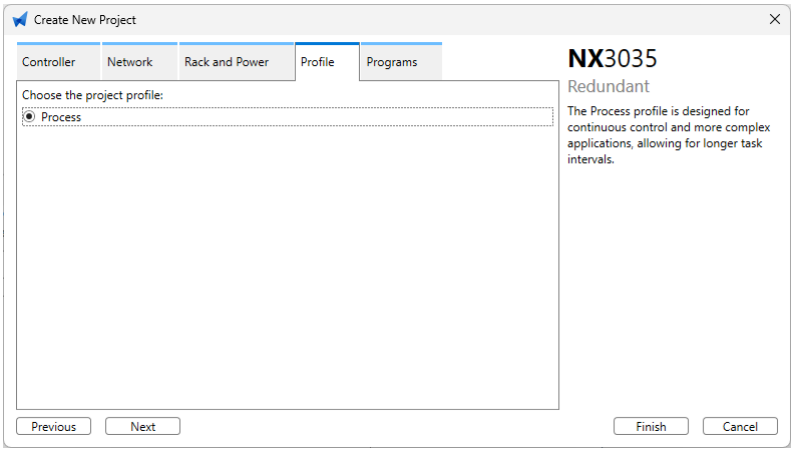


Figure 216: Tab - Profile

Click the *Next* button to select the last screen of the Wizard. The *Programs* tab of the Wizard is shown in the following figure, already filled with some options compatible with NX3035 redundancy. This screen only allows selecting the languages used to edit the POUs StartPrg and ActivePrg. In this example, the ST language was selected for both POUs, but other options are available (CFC, CFC page oriented, SFC, LD, FBD).

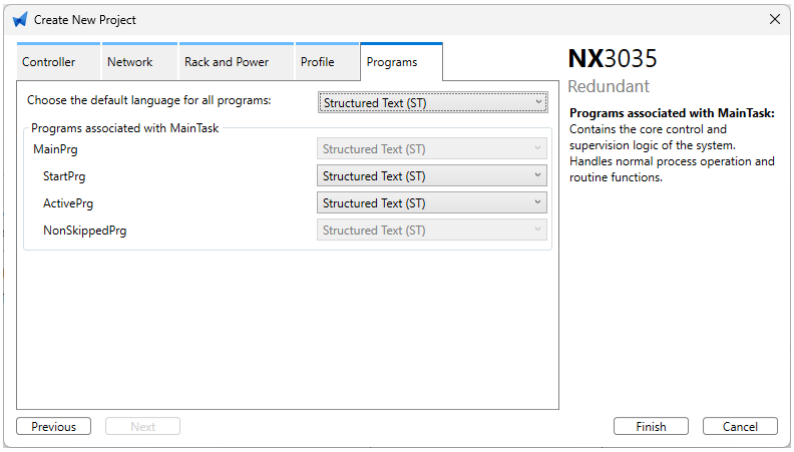


Figure 217: Tab - Programs

Finally, the user must press the *Finish* button. After that, the Wizard will generate configurations consistent with the user's answers across the five Wizard screens. For example, considering the examples from the previous five screens, the following hardware configuration will be generated in the *Configuration* tab.

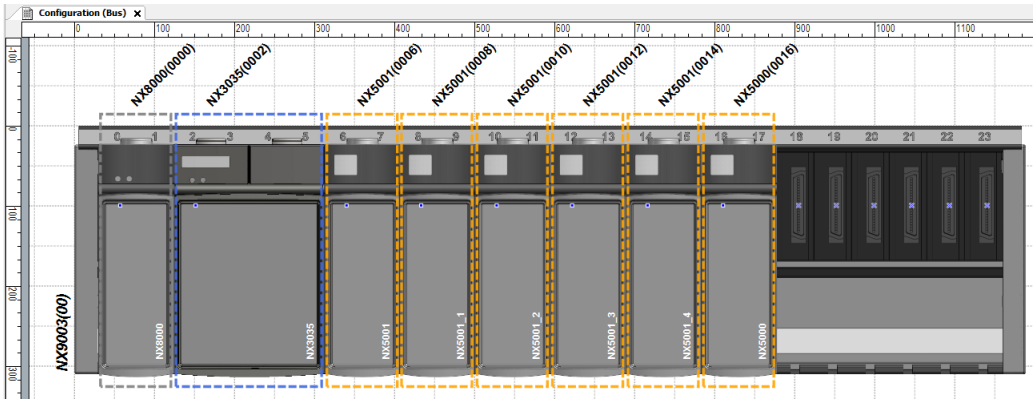


Figure 218: Half-cluster hardware configuration generated by the Wizard

6.4.2. Half-Clusters Configuration

The Wizard is always used to generate the first version of a redundant project. This ensures that the initial version of the project will be generated quickly and correctly.

However, some modifications may be necessary in a half-cluster, such as the insertion of new NX5001, NX5000, or local I/O modules, which can be done by changing the half-cluster configuration screen. The following chapters show how to add and/or configure the NX5000 and NX5001 modules.

Some rules and precautions must be followed for a redundant design, as described in the following section.

6.4.2.1. Fixed Configuration in the 0 to 5 Rack Positions

The following modules must always be installed in positions 0 to 5 of the selected rack:

- NX8000 power supply (position 0)
- NX3035 CPU (position 2)

These modules must not be removed from the original design generated by the Wizard. Any different configuration in these positions will result in an error reported by Mastertool when compiling the design.

6.4.3. Configuration of the Integrated Ethernet Ports of a Redundant NX3035 CPU (NET 1 to NET 6)

6.4.3.1. IP Address Configuration

For the integrated Ethernet ports of the NX3035 CPU, this topic was covered in the section IP Address Configuration on Integrated Ports NET1 ... NET6 of an NX3035 CPU in a Redundant CPU Setup.

For the remote Ethernet ports of the NX5000, this topic was covered in the section IP Address Configuration on NX5000 Modules in a Redundant CPU Setup.

6.4.3.2. NIC Teaming Configuration

For the integrated Ethernet ports of the NX3035 CPU, this topic was covered in the section Integrated Ethernet Interfaces Configuration, more specifically in the section Ethernet Port Mode Configuration.

For the remote Ethernet ports of the NX5000, this topic was covered in section IP Addressing of Remote Ethernet Ports on the NX5000 (NET1), more specifically in section Ethernet Port Mode Configuration for the NX5000.

6.4.3.3. NX5001 Modules Configuration

This topic was covered in section PROFIBUS Network Configuration with NX5001 Modules, and applies to both redundant and non-redundant CPs.

There are few differences in the configuration of a PROFIBUS network that apply only to redundant CPs:

- There is a single configuration parameter for the NX5001 module that applies exclusively to redundant PLCs. This is the *Fault Mode* parameter found in the *Module Parameters* tab of the NX5001. This parameter is described in section "Module Parameters" Tab.
- The watchdog of each PROFIBUS remote has a minimum value to be configured in a redundant PLC. This subject is addressed in section Watchdog Configuration on PROFIBUS Remote Stations in Redundant PLCs.

6.4.3.4. NX5000 Modules Configuration

This topic has already been covered in the section Configuration of Ethernet Interfaces with NX5000 Modules.

There are some differences in the configuration of an NX5000 interface that apply only to redundant CPs:

- Multiple IP addresses must be configured on a redundant PLC, instead of a single IP address as is the case with a non-redundant PLC. The type and number of IP addresses depends on the *Cluster IP Addressing Method* parameter, which only exists in redundant PLC designs. For more details, see section Basic Ethernet Port Settings for the NX5000 and section IP Address Configuration on NX5000 Modules in a Redundant CPU Setup.
- There is a failure mode configuration, described in section Ethernet Port Failure Mode Configuration.

6.4.3.5. NX3035 CPU Configurations Related to Redundancy

The redundancy settings related to the variables %I, %Q, and %M can be accessed by double-clicking on the NX3035 CPU, followed by selecting the *Redundancy Parameters* tab, shown on the following screen:

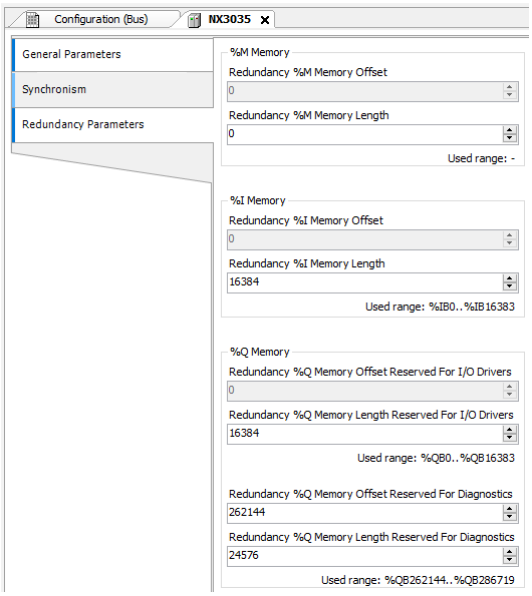


Figure 219: Redundancy parameters of the NX3035 CPU

To understand these parameters, you should read the sections Redundant and Non-redundant %I Variables, Redundant and Non-redundant %Q Variables e Redundant and Non-redundant %M Variables.

6.4.4. I/O Driver Configurations

The configuration of I/O drivers differs little from that of a non-redundant PLC. These differences generally consist of the use of special commands executed in the *NonSkippedPrg* program and some additional diagnostics.

Consider a redundant PLC that is a MODBUS TCP client of one or more MODBUS TCP server field devices. In this example, the following differentiated strategies can be adopted in a redundant PLC:

- It is recommended that client communication be disabled in the Non-Active PLC, so as not to overload field devices with unnecessary communications, since such communications only need to be carried out between the Active PLC and the field devices. To this end, in the Non-Active PLC, it is suggested to disable all MODBUS TCP client driver mappings for communication with field devices, by setting several mapping disable bits.
- It is important to verify that the Ethernet communication interfaces of the Non-Active PLC are intact, to avoid hidden failures that would only be discovered when this PLC switched to Active. A possible strategy for this is to define an additional test mapping, where the Active PLC as a client periodically performs a test communication where the Non-Active PLC is the server. This test mapping should only be enabled on the Active PLC and disabled on the Non-Active PLC. The failure diagnosis of this test mapping should be used by the maintenance team to repair a possible failure that prevents communication between the Active PLC and the Non-Active PLC.
- If the Active PLC loses communication with all field devices and also loses test communication with the Non-Active PLC, one option is to switch the Non-Active PLC to Active, provided that the Non-Active PLC is in Standby mode. There are commands that allow this switch-over.

The section Example of Managing a MODBUS TCP Client in a Redundant CPU shows a practical example of applying the strategies defined above.

In the case of a MODBUS TCP server, some strategies can also be used. Consider, for example, a SCADA system that is a MODBUS TCP client of the redundant PLC. Generally, the SCADA system will communicate using three different IP addresses (Active IP for communications related to the controlled process, IP of CPU A for diagnostics and commands exclusive to PLCA, and IP of CPU B for diagnostics and commands exclusive to PLCB). In this case, two strategies can be considered to address the problem:

1. In the first strategy, the SCADA operator will receive an alarm informing them that communication with the Active PLC has been lost. If communication continues to function with the Non-Active PLC (via CPU A IP or CPU B IP), the operator can send a special SCADA command to the Non-Active PLC, requesting that it switch to Active. To perform this switchover, the Non-Active PLC turns on the command bit *DG_NX3035.tRedundancy.RedCmdLoc.bStandbyRemote*.
2. The second strategy is fully automatic, requiring no operator intervention:
   a) The Active PLC detects that its MODBUS TCP server is no longer being accessed by the SCADA client. This is verified by freezing, for a few seconds, a MODBUS server diagnosis (*tStat.wRXRequests*) that counts the number of requests received from the SCADA client.
   b) The Non-Active PLC can make its MODBUS server diagnosis (*tStat.wRXRequests*) available to the Active PLC by copying this diagnosis to two bytes of the *DG_NX3035.tRedundancy.RedUsrLoc* structure. In the Active PLC, this diagnostic is read in two bytes of the *DG_NX3035.tRedundancy.RedUsrRem* structure.
   c) If the Active PLC realizes that it is not being accessed by SCADA, but the Non-Active PLC continues to be accessed, it can switch to Standby using the command *DG_NX3035.tRedundancy.RedCmdLoc.bStandbyLocal*.

In the case of the PROFIBUS network, there are also special commands differentiated for PLCs in Active and Non-Active states. In this case, however, redundancy management executes these commands automatically, without any management required on the part of the user.

6.4.5. MainTask Configurations

In a redundant PLC, the entire user application must be executed within the context of a cyclic task called MainTask (within the NonSkippedPrg and ActivePrg POUs).

The MainTask settings screen can be accessed by clicking on the MainTask object in the device tree. Two parameters must be adjusted on this screen:

- MainTask Interval
- Watchdog Time

Several considerations must be made to properly adjust the MainTask interval:

- The interval must be low enough to control the process effectively, observing the response times of all control loops.
- The interval must be high enough to accommodate, at a minimum, the sum of the following two times:
  - The maximum execution time of the NonSkippedPrg and ActivePrg POUs, together.
  - The time required to manage redundancy (redundancy overhead). An estimate of this time is calculated by Mastertool. This estimate is only reliable after the project is complete, with all POUs developed and redundant memory areas defined.
- In addition, the interval must have additional slack, necessary for other CPU tasks to have time to be executed (PROFIBUS communications, Ethernet communications, etc.).

The average cycle time of the MainTask can be measured by clicking on the Task Configuration object and then selecting the Monitor tab. It is recommended that this be done after the project is complete, so that the MainTask interval can be modified if necessary.

If the average cycle time measured for the MainTask exceeds 70% of the configured interval for MainTask, it is recommended to increase the interval so that there is a margin of 30% for the other tasks.

6.4.5.1. ActivePrg Program

In this POU, the user must create the main application, responsible for controlling the process. This POU is called by the main POU (MainPrg) and is only executed in the Active PLC.

The user can also create additional POUs (program, functions, or functional block) and call or instantiate them within the ActivePrg POU for the purpose of structuring the program. It is also possible to call functions and instantiate functional blocks defined in libraries.

It should be remembered that all symbolic variables defined in the ActivePrg POU, as well as instances of defined functional blocks, will be redundant variables. Symbolic variables defined in additional program-type POUs that are called within ActivePrg will also be redundant variables.

ATTENTION

VAR_TEMP type variables should not be used in the redundant program.

6.4.5.2. NonSkippedPrg Program

This POU is intended for controls that must be executed on both PLCs (PLCA and PLCB), regardless of their redundancy status. This POU is also called by the main POU (MainPrg).

It should be remembered that all symbolic variables defined in the NonSkippedPrg POU, as well as defined functional block instances, will be non-redundant variables. The user can also create additional POUs (program, functions, or functional block) and call or instantiate them within the NonSkippedPrg POU for the purpose of structuring their program. It is also possible to call functions and instantiate functional blocks defined in libraries.

ATTENTION

When calling additional program-type POUs within NonSkippedPrg, uncheck them in the Mastertool *Redundancy Configuration* object. By default, symbolic variables declared within these POUs will be redundant, and within NonSkippedPrg, non-redundant variables are usually desired.

Typical examples of controls executed in NonSkippedPrg are as follows:

- Create a compact structure of diagnostics (%Q) to be reported to a SCADA system, from a complete structure of diagnostics, where several diagnostics are not of interest to the SCADA system. These diagnostics can be extracted from data structures such as *RedDgnLoc*, *RedDgnRem*, *RedUsrLoc*, *RedUsrRem*, etc.
- Copy commands received from a SCADA to the respective fields of the *RedCmdLoc* data structure, and interlock these commands if necessary
- Manage user-controlled switchovers, depending on failures such as communication with a SCADA system or with MODBUS TCP server devices.
- Enable or disable communications depending on the redundancy status (Active or Inactive). For example, a MODBUS TCP client driver could disable its communications on the Inactive PLC so as not to generate useless requests to servers.

ATTENTION

It is not recommended to use the *TOF_RET*, *TON_RET*, *TOF*, and *TON* function blocks in the *NonSkippedPrg* program. See section Limitations on a Redundant PLC Programming.

6.4.6. *Redundancy Configuration* Object

This object, located in the project's device tree, is automatically created by the Wizard when creating a new redundant project. It is used to determine which POUs and GVLs will be redundant. Variables declared inside redundant POUs and GVLs will be redundant and therefore synchronized between the CPUs.

By default, user-created POUs and GVLs are marked as redundant in the *Redundancy Configuration* object, and it is up to the user to revert this setting when they should be non-redundant. For example, when creating a new POU that will be called inside the *NonSkippedPrg* POU, this new POU should normally be reverted to non-redundant within the *Redundancy Configuration* object.

Some POUs and GVLs have fixed (non-editable) configuration within the *Redundancy Configuration* object (the variables contained in these POUs and GVLs are counted as redundant data):

- POUs that are never redundant:
  - MainPrg
  - NonSkippedPrg
  - SpecialVariablesPrg
- GVLs that are never redundant:
  - System_Diagnostics
  - Module_Diagnostics
  - ReqDiagnostics
  - IOQualities
  - SpecialVariables

ATTENTION

The maximum number of redundant objects is limited to 1000. Objects marked as redundant that do not contain variable declarations are not counted toward this limit.

6.4.7. GVL *Module_Diagnostics* and GVL *System_Diagnostics*

These two special GVLs are created and filled automatically by the Wizard when creating a new redundant project, and also when inserting or removing objects in the device tree. They cannot be modified by the user.

Within these GVLs, several data structures are automatically created, containing diagnostics and commands. These diagnostics and commands reside in direct representation variables (%Q or %I).

These GVLs contain several statements with the AT directive to define symbolic names for these diagnostics and commands. This way, when the user needs to reference these variables, it is recommended to use a symbolic name instead of the numerical reference next to %Q or %I.

6.4.8. GVLs with Redundant or Non-Redundant Symbolic Variables

The user can create other GVLs, different from those mentioned above, to declare redundant or non-redundant symbolic variables.

For variables to be redundant, the GVL must be marked as redundant in the Redundancy Configuration object in the project device tree, meaning that this GVL is redundant.

By default, all GVLs created by the user will initially be redundant. If the user wants it to be non-redundant, they must uncheck it in the *Redundancy Configuration* object.

6.4.9. GVLs with AT Directives

GVLs with AT directives can be used to define symbolic names for direct representation variables (%I, %Q, or %M).

If there are variables defined with the AT directive within a GVL:

- This GVL must be marked as non-redundant in the *Redundancy Configuration* object. Otherwise, Mastertool will detect a compilation error.
- Even if a GVL with AT directives is marked as non-redundant, it may point to redundant %I, %Q, or %M variables. To check whether a %I, %Q, or %M variable is redundant or non-redundant, see section NX3035 CPU Configurations Related to Redundancy.

6.4.10. Program-Type POUs with Redundant or Non-Redundant Symbolic Variables

Symbolic variables declared within a program-type POU will be redundant or not, depending on how the POU is configured within the *Redundancy Configuration* object.

To define a new POU as redundant, after creating the POU, mark it in the *Redundancy Configuration* object configuration in the project device tree. By default, all POUs created by the user will initially be redundant.

If a POU is marked as redundant, it is not possible to use the AT directive within it to define symbolic names for direct representation variables (%I, %Q, or %M). Mastertool will detect a compilation error in this case.

6.4.11. Breakpoints Utilization in Redundant Systems

For redundant systems, breakpoints should only be used on the Active half-cluster while the other half-cluster is in a state other than Standby.

The user will be prevented from activating breakpoints on the Active PLC while the other PLC is in Standby.

In addition, the user must remember the following:

- Remove all breakpoints from the Active PLC when finishing debugging.
- Remove all breakpoints from the Active PLC before switching the other PLC to standby.

6.4.12. MODBUS Instances Managing in Redundant System

When the Failure Mode option for Ethernet interfaces is disabled, or in the case of MODBUS Server instances, the MODBUS instances are independent of the redundancy mechanism and therefore must be managed within the application. It is up to the user to decide which instances should be enabled or disabled when a CP enters a Non-Active state. When the Failure Mode option is enabled for Ethernet ports using MODBUS Client, no additional code implementation is required to control the switch-over.

The example below, inserted in the NonSkippedPrg program, checks the current CP state and, if it is in a Non-Active state, disables the MODBUS_RTU_Master, MODBUS_Server, and MODBUS_Symbol_Server instances:

```
VAR
eRedStateLocal : REDUNDANCY_STATE;
eRedStateLocal_old : REDUNDANCY_STATE;
END_VAR

// Read the current state of the local CP
eRedStateLocal := DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;

// Has the local CP state changed?
IF eRedStateLocal <> eRedStateLocal_old THEN
IF eRedStateLocal = REDUNDANCY_STATE.ACTIVE THEN
// Local CP entered Active state
System_Diagnostics.DG_MODBUS_RTU_Master.tCommand.bRestart := TRUE;
Module_Diagnostics.DG_MODBUS_Server.tCommand.bRestart := TRUE;
Module_Diagnostics.DG_MODBUS_Symbol_Server.tCommand.bRestart := TRUE;
ELSE
// Local CP entered a Non-Active state

System_Diagnostics.DG_MODBUS_RTU_Master.tCommand.bStop := TRUE;
Module_Diagnostics.DG_MODBUS_Server.tCommand.bStop := TRUE;
Module_Diagnostics.DG_MODBUS_Symbol_Server.tCommand.bStop := TRUE;
END_IF
// Save the last local CP state
eRedStateLocal_old:= eRedStateLocal;
END_IF
```

6.4.13. Limitations on a Redundant PLC Programming

In a redundant PLC, there are some limitations regarding CPU scheduling. These limitations are discussed in the subsections below.

6.4.13.1. Limitations in Redundant GVLs and POUs

In a redundant GVL or program-type POU, the following limitations must be observed for half-clusters to function correctly:

- Do not use VAR_TEMP variables.
- Do not mix variable types (VAR, VAR RETAIN, VAR PERSISTENT, VAR CONSTANT, etc.), only one type should be used in each GVL or POU.
- Do not mix symbolic variable declarations with ATs in GVLs and POUs. Separate GVLs or POUs should be created, declaring AT type variables in one and symbolic variables in the other. Normally, ATs should only be used in GVLs.
- Do not store the address of a variable in a redundant variable (make a redundant variable a pointer to an address), as variable addresses may differ in PLCA and PLCB.
- Do not use functional blocks for writing and reading the RTC clock in redundant POUs. More details can be found in section RTC Clock.

6.4.13.2. Limitations in Non-Redundant POUs

In a non-redundant program-type POU, the following limitations must be observed for the half-clusters to function correctly:

- Traditional TON and TOF timer function blocks cannot be used, as they use the IEC timer. When the Standby PLC enters the active state (with the other half-cluster leaving the Active state), the IEC timer will be synchronized, causing a discontinuity in the timer value. The TON_NR and TOF_NR functional blocks available in the NextoStandard library should be used. See section Non-Redundant Timer.
- Program-type POUs written in SFC (Sequential Function Chart) language cannot be used, as they use the IEC timer to time transitions.

6.4.14. Getting the Redundancy State of a Half-Cluster

It is possible to check the redundancy status of a half-cluster described in the section Structure of Diagnostics, Commands, and User Information for Redundancy:

```
VAR
  eRedStateLocal : IoDrvNextoRedundancy.REDUNDANCY_STATE;
END_VAR

eRedStateLocal := DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;
```

This allows the user to control logic that depends on the redundant state of the PLC.

6.4.15. Reading Non-Redundant Diagnostics

A redundant project, in addition to using redundant diagnostics (e.g., diagnostics from a remote PROFIBUS), can also use non-redundant diagnostics (diagnostics specific to the NX5000, NX5001, NX3035 modules, etc.). Such non-redundant diagnostics are generally used directly within the POU *NonSkippedPrg*, or in some other non-redundant POU called within the POU *NonSkippedPrg*.

Some of these non-redundant diagnostics may not be valid in the first moments after the PLC assumes the Active state, as they must be recalculated by the new PLC in the Active state, and this may take some time. Before using a non-redundant diagnostic, you must wait for the PLC to be in the Active state for some time, typically around 5 seconds. Here are some examples of such diagnostics:

- **DG_NX5001.tGeneral.bPbusCommFail:** the PROFIBUS network below this NX5001 has a communication failure.
- **DG_NX5001.tGeneral.bSlaveWithDiagnostic:** at least one slave below this NX5001 has active diagnostics failure.

The following shows an example of code in POU NonSkippedPrg to use a diagnosis of this type only after 5 seconds in the active state.

```
PROGRAM NonSkippedPrg
VAR
  Active_Time : NextoStandard.TON_NR; // Non-redundant timer: measures time in
    Active state
END_VAR


Active_Time.IN:=System_Diagnostics.DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag
    .eRedState = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE;
Active_Time.PT := T#5S;
Active_Time();

IF Active_Time.Q THEN
  ; // Code snippet where non-redundant diagnostics can be used (more than 5
    seconds in Active)
ELSE
  ; // Code snippet where it is not possible to use non-redundant diagnostics (
    less than 5 seconds in Active)
END_IF
```

6.4.16. Example of Managing a MODBUS TCP Client in a Redundant CPU

This section describes an example of managing a MODBUS TCP client on a redundant PLC, considering the architecture shown in the following figure.

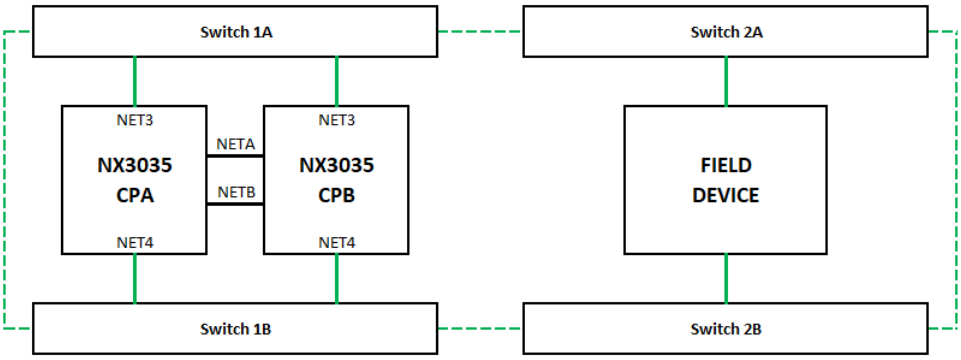


Figure 220: Architecture of the MODBUS TCP Client communication example

In the architecture, it can be seen that the two redundant half-clusters (PLCA and PLCB), as MODBUS TCP clients, must communicate with a field device, which is a MODBUS TCP server. This is a high-availability architecture due to the following characteristics:

- The PLCA and PLCB half-clusters use a NIC Teaming pair (NET3+NET4), with the NET3 ports connecting to switch 1A and the NET4 ports connecting to switch 1B. This allows for individual port failure (NET3 or NET4) as well as individual switch failure (1A or 1B).
- It is assumed that the field device also has the ability to connect via NIC Teaming, and its two ports connect to two different switches (2A and 2B).
- Finally, the four switches are connected in a ring, to tolerate a single interconnection failure between switches.

The application developed to control this communication has the following characteristics, already mentioned in section I/O Driver Configurations:

- Mappings are disabled in the Non-Active PLC, so that it does not generate unnecessary communication requests to the field device. Only the Active PLC generates communication requests to the field device. This prevents unnecessary communication overload on the network and for the field device.
- The Active PLC will calculate the following fault diagnoses:
  - Failure of the mappings used for communication with the field device. In the example that will be presented, there are two mappings. Mapping 0 is used to read 120 holding registers from the field device, while mapping 1 is used to write 120 holding registers to the field device.
  - Communication failure against the Non-Active PLC. In this case, the Active PLC acts as a client and the Non-Active PLC as a server. The purpose of this communication is to test the communication integrity of the Non-Active PLC, to prevent hidden failures in the Non-Active PLC. This is necessary because the Non-Active PLC does not generate communication requests as a client.
  - General communication failure, which occurs when there is a simultaneous failure in all communication mappings with the field device, and in addition there is a failure in communication with the Non-Active PLC.
- A switch-over may take the Active PLC to the Standby state, and the Standby PLC to the Active state. For this to happen, three conditions must occur simultaneously:
  1. The Active PLC, before going to Standby, must have remained Active for a minimum of 15 seconds.
  2. The Non-Active PLC must be in Standby state, i.e., able to switch to Active state.
  3. There must be a general communication failure lasting at least 5 seconds, or simultaneous link-downs on the NET3 and NET4 ports of the Active PLC.

The following sections describe details of the application developed for this example.

6.4.16.1. Configuration of NET3 / NET4 Interfaces in the Redundant CPU

The configuration of the NET3 and NET4 interfaces of the Redundant PLC is shown in the following figure:

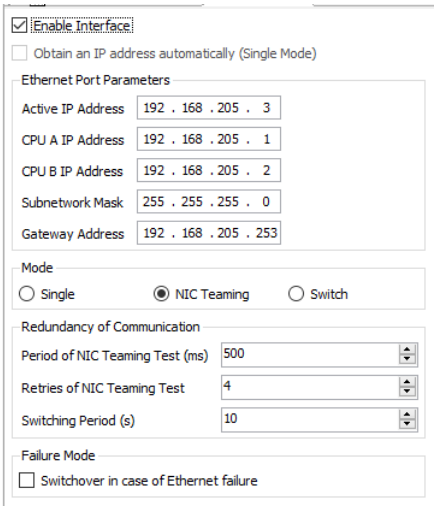


Figure 221: Configurations of NET3 and NET4

Note that NET3+NET4 form a NIC Teaming pair.

6.4.16.2. MODBUS TCP Server Configuration for Test Communication

It is necessary to configure a MODBUS TCP server below the NET3+NET4 ports of the redundant PLC, so that test communication can be performed where the Active PLC is the client and the Non-Active PLC is the server.

The following figure shows the configuration of this server, which provides a single holding register to be read by the client.

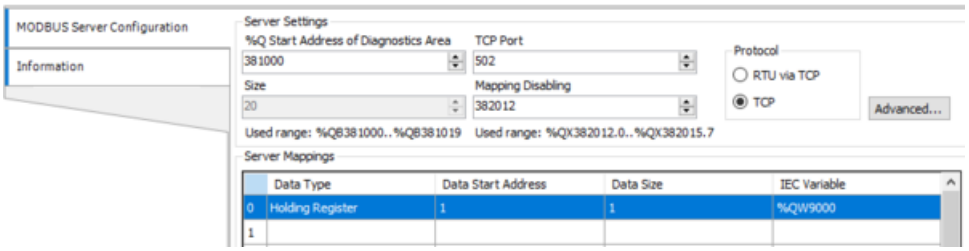


Figure 222: MODBUS TCP Server Settings under NET3 and NET4

6.4.16.3. MODBUS TCP Client Configuration

The following figure shows an overview of the MODBUS TCP client configuration below the NET3+NET4 ports of the redundant PLC.

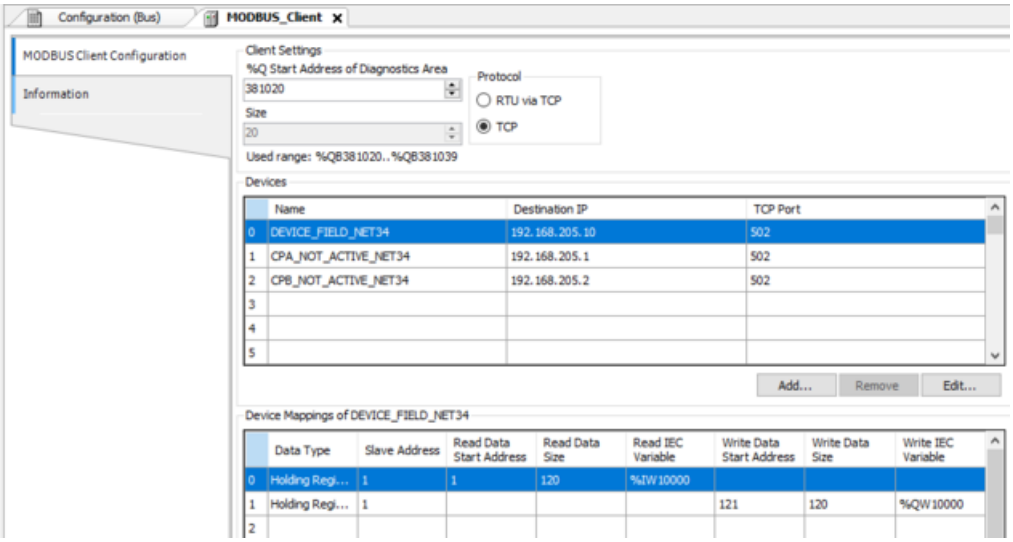


Figure 223: MODBUS TCP Client Settings under NET3 and NET4

The following observations apply to the three server devices configured on this MODBUS TCP client:

- The first server device (FIELD_DEVICE_NET34) corresponds to the field device (IP address = 192.168.205.10) and contains two mappings (0 and 1). Mapping 0 configures the reading of 120 holding registers, and mapping 1 configures the writing of 120 holding registers.
- The second server device (PLCA_NOT_ACTIVE_NET34) corresponds to the PLCA (IP address = 192.168.205.1), and contains a single mapping (0), which configures the reading of 1 holding register. This mapping should only be enabled when the PLCB is the Active PLC (client), and the PLCA is the Non-Active PLC (server). The purpose of this mapping is to test the integrity of the PLCA's communication while it is in the Non-Active state.
- The third server device (PLCB_NOT_ACTIVE_NET34) corresponds to the PLCB (IP address = 192.168.205.2) and contains a single mapping (0), which configures the reading of 1 holding register. This mapping should only be enabled when the PLCA is the Active PLC (client), and the PLCB is the Non-Active PLC (server). The purpose of this mapping is to test the integrity of the PLCB communication while it is in the Non-Active state.

6.4.16.4. GVLs and POUs for Managing the MODBUS TCP Client

The application for managing the MODBUS TCP client in this example consists of two GVLs and one POU.
The following figure shows the *GVL_AT_MODBUS_CLIENT_NET34*:

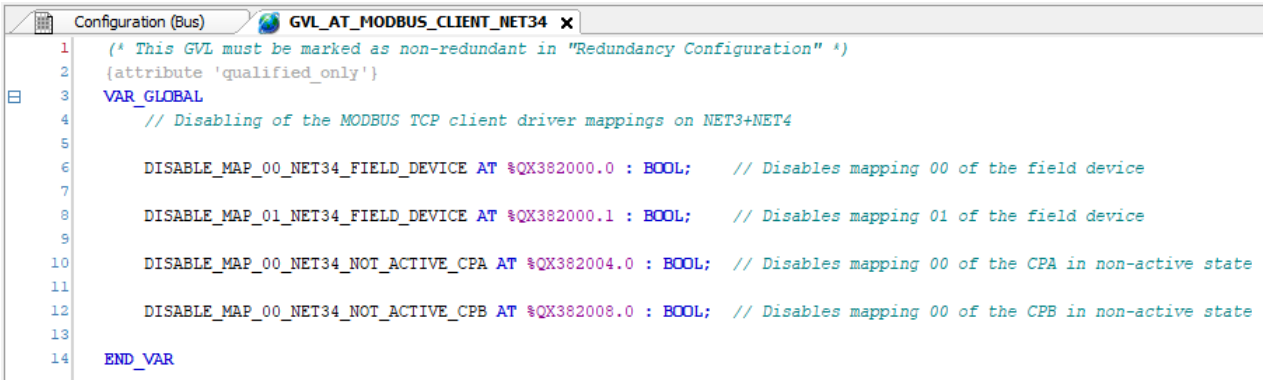


Figure 224: GVL_AT_MODBUS_CLIENT_NET34

```
(* This GVL must be marked as non-redundant in "Redundancy Configuration" *)
{attribute 'qualified_only'}
VAR_GLOBAL
```

```
  // Disabling of the MODBUS TCP client driver mappings on NET3+NET4

  DISABLE_MAP_00_NET34_FIELD_DEVICE AT %QX382000.0 : BOOL;
  // Disables mapping 00 of the field device

  DISABLE_MAP_01_NET34_FIELD_DEVICE AT %QX382000.1 : BOOL;
  // Disables mapping 01 of the field device

  DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA AT %QX382004.0 : BOOL;
  // Disables mapping 00 of the PLCA in non-active state

  DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB AT %QX382008.0 : BOOL;
  // Disables mapping 00 of the PLCB in non-active state
END_VAR
```

The GVL in the previous figure simply uses AT directives to define symbolic names for the disable bits of the four mappings described above. It is important to note that this GVL must be marked as non-redundant in the *Redundancy Configuration* object.

The following figure shows the GVL_MODBUS_CLIENT_NET34:

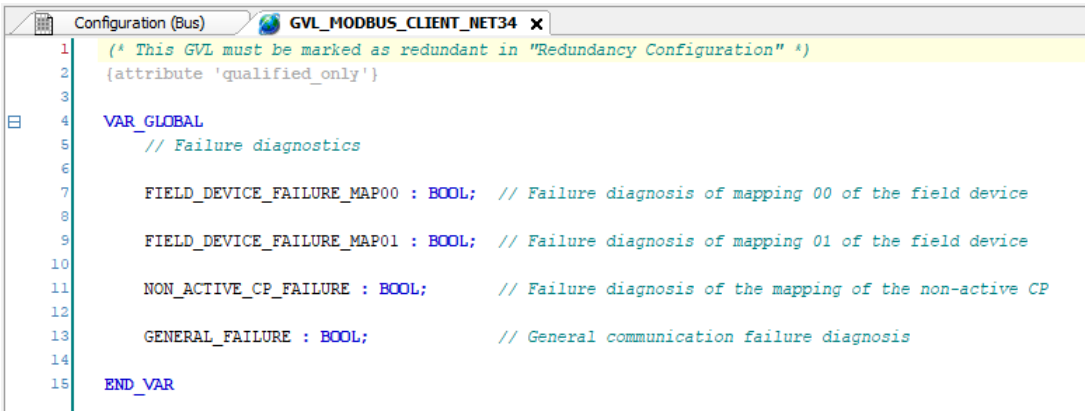


Figure 225: GVL_MODBUS_CLIENT_NET34

```
(* This GVL must be marked as redundant in "Redundancy Configuration" *)
{attribute 'qualified_only'}
VAR_GLOBAL
  // Failure diagnostics

  FIELD_DEVICE_FAILURE_MAP00 : BOOL;  // Failure diagnosis of mapping 00 of the
    field device

  FIELD_DEVICE_FAILURE_MAP01 : BOOL;  // PT: Failure diagnosis of mapping 01 of
    the field device

  NON_ACTIVE_CP_FAILURE : BOOL;   // PT: Failure diagnosis of the mapping of the
     non-active PLC

  GENERAL_FAILURE : BOOL;       // PT: General communication failure diagnosis

END_VAR
```

The GVL in the previous figure defines variables for fault diagnostics. It is important to note that this GVL must be marked as redundant in the *Redundancy Configuration* object.

Finally, the following figure shows the POU *Manage_MODBUS_CLIENT_NET34*:

Configuration (Bus) | Manage_MODBUS_CLIENT_NET34

```
1  (* This POU must be called in NonSkippedPrg and marked as non-redundant in "Redundancy Configuration" *)
2
3  PROGRAM Manage_MODBUS_CLIENT_NET34
4  VAR
5      eRedStateLocal : IoDrvNextoRedundancy.REDUNDANCY_STATE; // Redundancy state of the local CP
6
7      eRedStateRemote : IoDrvNextoRedundancy.REDUNDANCY_STATE;// Redundancy state of the remote CP
8
9      eID_CP : NextoStandard.CLUCONF_ID;                      // Identification of the local CP (PLC_A or PLC_B)
10
11     LinkDownsSimultaneos : BOOL;                            // Simultaneous link downs on NET3 and NET4
12
13     TON_ActiveTime : NextoStandard.TON_NR;                  // Non-redundant timer: time of the local CP in active state
14
15     TON_GeneralFailure : NextoStandard.TON_NR;              // Non-redundant timer: duration of the general communication failure
16
17     TON_Reserved_Failure : NextoStandard.TON_NR;            // Non-redundant timer: duration of the communication failure with the standby CP
18
19 END_VAR
```

Figure 226: Manage_MODBUS_CLIENT_NET34– variables section

Configuration (Bus) | Manage_MODBUS_CLIENT_NET34

```
// Captures the redundancy state of local and remote CPs, and identifies the local CP
eRedStateLocal := System_Diagnostics.DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;
eRedStateRemote := System_Diagnostics.DG_NX3035.tRedundancy.RedDgnRem.sGeneral_Diag.eRedState;
eID_CP := System_Diagnostics.DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.ePLC_ID;
// Enabling and disabling MODBUS client mappings
IF (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE) THEN
    // On the active CP, enables mappings with field device
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_FIELD_DEVICE := FALSE;
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_01_NET34_FIELD_DEVICE := FALSE;
    IF eID_CP = PLC_A THEN
        // When CPA is active, disables mapping for CPA diagnostics and enables mapping for CPB diagnostics
        GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA := TRUE;
        GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB := FALSE;
    ELSE
        // When CPB is active, enables mapping for CPA diagnostics and disables mapping for CPB diagnostics
        GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA := FALSE;
        GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB := TRUE;
    END_IF
ELSE
    // On the non-active CP, disables all mappings
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_FIELD_DEVICE := TRUE;
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_01_NET34_FIELD_DEVICE := TRUE;
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA := TRUE;
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB := TRUE;
END_IF
// Calculates diagnostics for MODBUS client mappings that are enabled on the active CP
IF (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE) THEN
    GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP00 := NOT(DG_FIELD_DEVICE_NET34_Mapping_000.byStatus.bCommOk);
    GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP01 := NOT(DG_FIELD_DEVICE_NET34_Mapping_001.byStatus.bCommOk);

    IF eID_CP = PLC_A THEN
        GVL_MODBUS_CLIENT_NET34.NON_ACTIVE_CP_FAILURE := NOT(DG_CPB_NOT_ACTIVE_NET34_Mapping_000.byStatus.bCommOk);
    ELSE
        GVL_MODBUS_CLIENT_NET34.NON_ACTIVE_CP_FAILURE := NOT(DG_CPA_NOT_ACTIVE_NET34_Mapping_000.byStatus.bCommOk);
    END_IF

    GVL_MODBUS_CLIENT_NET34.GENERAL_FAILURE := GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP00 AND
                                               GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP01 AND
                                               GVL_MODBUS_CLIENT_NET34.NON_ACTIVE_CP_FAILURE;
END_IF
// Calculates simultaneous link-downs on NET3 and NET4 of the local CP
LinkDownsSimultaneos := System_Diagnostics.DG_NX3035.tDetailed.Ethernet.NET[3].bLinkDown AND DG_NX3035.tDetailed.Ethernet.NET[4].bLinkDown;
// Commands switchover of the local CP from active to standby when all three of the following conditions are true:
// 1) The local CP has been in an active state for more than 15 seconds.
// 2) The remote CP is in a standby state and is therefore capable of switching to active.
// 3) General failure for more than 5 seconds, or simultaneous link-downs on NET3 and NET4.

TON_ActiveTime(IN := (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE), PT := T#15S);
TON_GeneralFailure(IN := (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE) AND GVL_MODBUS_CLIENT_NET34.GENERAL_FAILURE, PT := T#5S);
IF TON_ActiveTime.Q AND (eRedStateRemote = IoDrvNextoRedundancy.REDUNDANCY_STATE.STANDBY) AND(TON_GeneralFailure.Q OR LinkDownsSimultaneos) THEN
    // Command to switch from active to standby
    System_Diagnostics.DG_NX3035.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
END_IF
```

Figure 227: Manage_MODBUS_CLIENT_NET34 – code section

```
(* This POU must be called in NonSkippedPrg and marked as non-redundant in "
   Redundancy Configuration" *)
PROGRAM Manage_MODBUS_CLIENT_NET34
VAR
  eRedStateLocal : IoDrvNextoRedundancy.REDUNDANCY_STATE;   // Redundancy state
   of the local PLC

  eRedStateRemote : IoDrvNextoRedundancy.REDUNDANCY_STATE;  // Redundancy state
   of the remote PLC

  eID_CP : NextoStandard.CLUCONF_ID;              // Identification of the local
    PLC (PLC_A or PLC_B)

  LinkDownsSimultaneos : BOOL;                  // Simultaneous link downs on NET3
    and NET4

  TON_ActiveTime : NextoStandard.TON_NR;            // Non-redundant timer: time
    of the local PLC in active state

  TON_GeneralFailure : NextoStandard.TON_NR;          // Non-redundant timer:
```

```
    duration of the general communication failure

  TON_Reserved_Failure : NextoStandard.TON_NR;        // Non-redundant timer:
    duration of the communication failure with the standby PLC

END_VAR



// Captures the redundancy state of local and remote CPs, and identifies the
    local PLC
eRedStateLocal := System_Diagnostics.DG_NX3035.tRedundancy.RedDgnLoc.
    sGeneral_Diag.eRedState;
eRedStateRemote := System_Diagnostics.DG_NX3035.tRedundancy.RedDgnRem.
    sGeneral_Diag.eRedState;
eID_CP := System_Diagnostics.DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.
    ePLC_ID;
//  Enabling and disabling MODBUS client mappings
IF (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE) THEN
  // On the active PLC, enables mappings with field device
  GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_FIELD_DEVICE := FALSE;
  GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_01_NET34_FIELD_DEVICE := FALSE;
  IF eID_CP = PLC_A THEN
    // When PLCA is active, disables mapping for PLCA diagnostics and enables
    mapping for PLCB diagnostics
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA := TRUE;
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB := FALSE;
  ELSE
    // When PLCB is active, enables mapping for PLCA diagnostics and disables
    mapping for PLCB diagnostics
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA := FALSE;
    GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB := TRUE;
  END_IF
ELSE
  // On the non-active PLC, disables all mappings
  GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_FIELD_DEVICE := TRUE;
  GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_01_NET34_FIELD_DEVICE := TRUE;
  GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPA := TRUE;
  GVL_AT_MODBUS_CLIENT_NET34.DISABLE_MAP_00_NET34_NOT_ACTIVE_CPB := TRUE;
END_IF
// Calculates diagnostics for MODBUS client mappings that are enabled on the
    active PLC
IF (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.ACTIVE) THEN
  GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP00 := NOT(
    DG_FIELD_DEVICE_NET34_Mapping_000.byStatus.bCommOk);
  GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP01 := NOT(
    DG_FIELD_DEVICE_NET34_Mapping_001.byStatus.bCommOk);
  IF eID_CP = PLC_A THEN
    GVL_MODBUS_CLIENT_NET34.NON_ACTIVE_CP_FAILURE := NOT(
    DG_CPB_NOT_ACTIVE_NET34_Mapping_000.byStatus.bCommOk);
  ELSE
    GVL_MODBUS_CLIENT_NET34.NON_ACTIVE_CP_FAILURE := NOT(
    DG_CPA_NOT_ACTIVE_NET34_Mapping_000.byStatus.bCommOk);
  END_IF
  GVL_MODBUS_CLIENT_NET34.GENERAL_FAILURE := GVL_MODBUS_CLIENT_NET34.
    FIELD_DEVICE_FAILURE_MAP00 AND
```

```
                        GVL_MODBUS_CLIENT_NET34.FIELD_DEVICE_FAILURE_MAP01 AND
                        GVL_MODBUS_CLIENT_NET34.NON_ACTIVE_CP_FAILURE;
END_IF
// Calculates simultaneous link-downs on NET3 and NET4 of the local PLC
LinkDownsSimultaneos := System_Diagnostics.DG_NX3035.tDetailed.Ethernet.NET[3].
    bLinkDown AND DG_NX3035.tDetailed.Ethernet.NET[4].bLinkDown;
// Commands switchover of the local PLC from active to standby when all three of
     the following conditions are true:
// 1) The local PLC has been in an active state for more than 15 seconds.
// 2) The remote PLC is in a standby state and is therefore capable of switching
     to active.
// 3) General failure for more than 5 seconds, or simultaneous link-downs on
    NET3 and NET4.
TON_ActiveTime(IN:= (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.
    ACTIVE), PT:= T#15S);
TON_GeneralFailure(IN:= (eRedStateLocal = IoDrvNextoRedundancy.REDUNDANCY_STATE.
    ACTIVE) AND GVL_MODBUS_CLIENT_NET34.GENERAL_FAILURE, PT:= T#5S);
IF TON_ActiveTime.Q AND (eRedStateRemote = IoDrvNextoRedundancy.REDUNDANCY_STATE
    .STANDBY) AND (TON_GeneralFailure.Q OR LinkDownsSimultaneos) THEN
  // Command to switch from active to standby
  System_Diagnostics.DG_NX3035.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
END_IF
```

It is important to note that this POU must be marked as non-redundant in the *Redundancy Configuration* object, and that it must be called within the *NonSkippedPrg* POU. The previous figures contain comments that sufficiently explain how this POU works.

6.5. Redundant CPU Program Downloading

The section Redundant CPU Programming dealt with aspects related to the development of a project for a redundant PLC with CPU NX3035.

In this section, we discuss methods and steps to download this project in a redundant PLC, considering the following situations:

- Downloading the project in a brand new NX3030 CPU or in a CPU with an unknown project.
- Online downloading of modifications.
- Offline downloading of modifications with interruption of process control during a scheduled process shutdown.

In addition, there is another more complex situation, which will be discussed in a separate chapter: Offline Program Download Without Interrupting Process Control. In this case, even if the project modification requires offline downloading, special redundancy features are used so as not to interrupt process control.

6.5.1. Initial Downloading of a Redundant Project

This section describes a safe procedure for performing the download of a redundant project in a NX3035 CPU. This is necessary, for example, for a new factory CPU just taken out of the box, or for a CPU containing an unknown project.

The worst case is assumed, in which the CPU may contain an unknown project, and its inclusion in an automation network may cause unpredictable effects, such as IP addresses conflicting with other equipment installed in this automation network. For this reason, the safe procedure described below contains precautions such as completely isolating this CPU from the automation network until a known project is downloaded onto it.

The following steps must be performed for both half-clusters (PLCA and PLCB) that make up a redundant PLC. First, perform all steps for one of the half-clusters (e.g., PLCA), and then for the other (e.g., PLCB).

1. Start the procedure with the CPU powered down.
2. Disconnect all Ethernet ports from the CPU (NET1 ... NET6), as well as the NETA and NETB synchronization network ports.

3. Start the CPU without application, performing the procedure in Not Downloading the Application at Startup.
4. Use a computer with the Mastertool programmer and open the project that must be downloaded on the redundant PLC that will use this CPU.
5. Connect an RJ45 Ethernet port on this computer to the NET1 port of the CPU using a point-to-point cable.
6. Temporarily change the IP address and subnet mask of this computer port so that it is on the same Ethernet subnet configured for the NET1 port of the CPU in the project opened in step 4. Consider, for example, that in the project, the NET1 port of the CPU has Active IP Address = 192.168.17.100, IP Address of the CPU A = 192.168.17.101, IP Address of the CPU B = 192.168.17. 102, and Subnet Mask = 255.255.255.0. In this example, the computer port can be temporarily set to IP = 192.168.17.10 and mask = 255.255.255.0.
7. Temporarily disconnect any other Ethernet ports on this computer and disable any Wi-Fi connections. This will prevent Mastertool from making unwanted connections to other CPUs via other Ethernet or Wi-Fi ports. The goal is for Mastertool to only be able to locate the CPU where the open project should be downloaded.
8. Use Mastertool's *Easy Connection* function—described in the Mastertool manual—to set the IP address, subnet mask, and gateway address for the NET1 port. These parameters must be the same as those configured in the open project in step 4. There are two possible addresses:
   a) If this CPU is to be installed in the PLCA half-cluster, it must be the *IP address of CPU A* of the NET1 port.
   b) If this CPU is to be installed in the PLCB half-cluster, it must be the *IP address of CPU B* of the NET1 port.
   c) Click *Device* in the device tree, select the *Communication Settings* tab, and finally the *Map Network* button.
9. A list with a single detected CPU should appear. Double-click this CPU to define an active path to it.

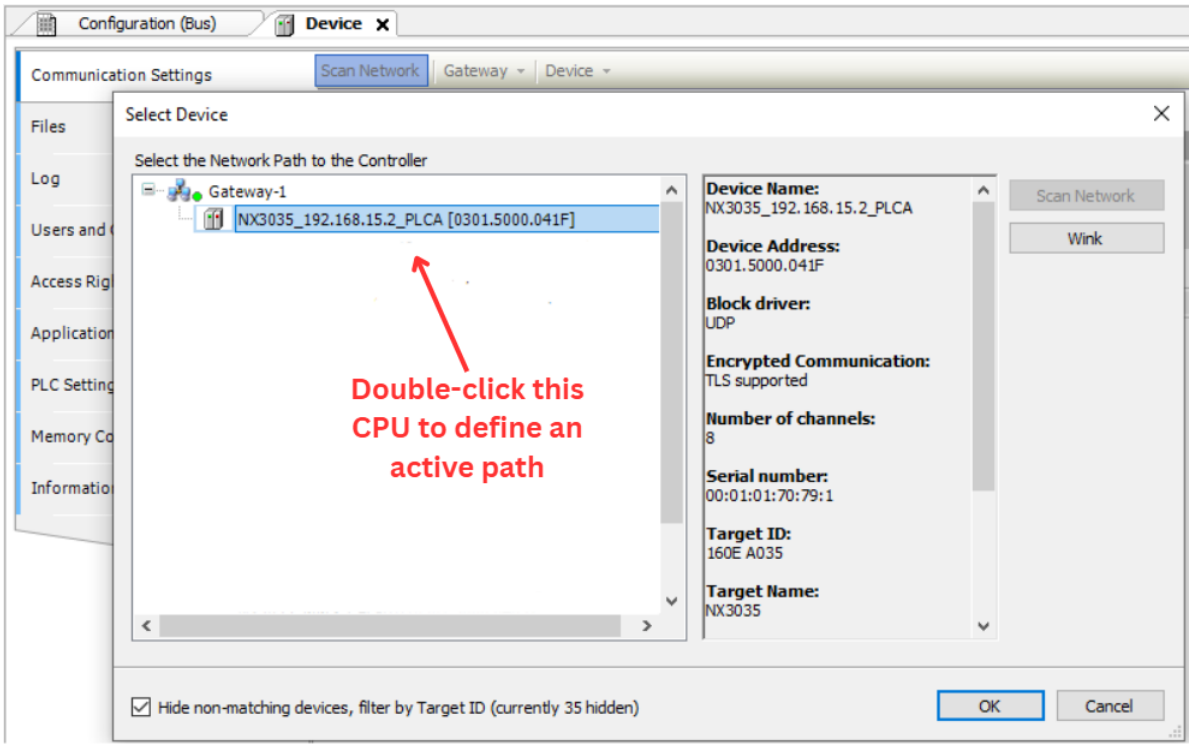


Figure 228: Selecting the Device

10. At this point, it is advisable to give this CPU a meaningful name, for example, GENERATOR_B for the PLCB half-cluster of the GENERATOR cluster. This can be done in *Device*, *Communication Settings* tab, *Device*, *Rename active device*.

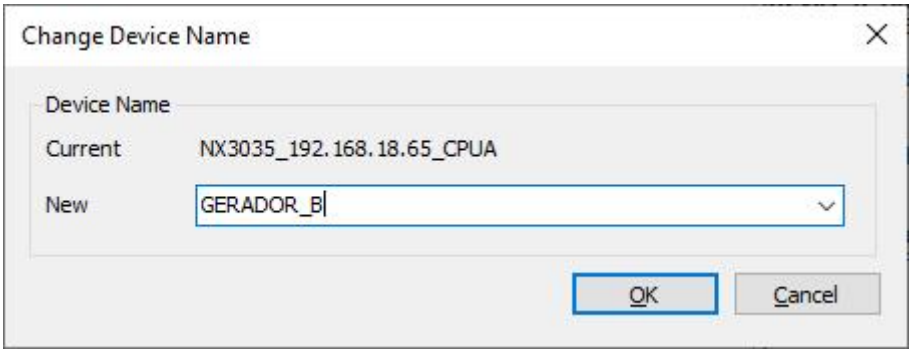


Figure 229: Changing the Device Name

11. Through the *Communication* menu, *Redundancy Configuration*, you must adjust the PLC identification, which in a redundant PLC can be PLCA or PLCB. Then press the *Write* button. This will cause the CPU to reboot.

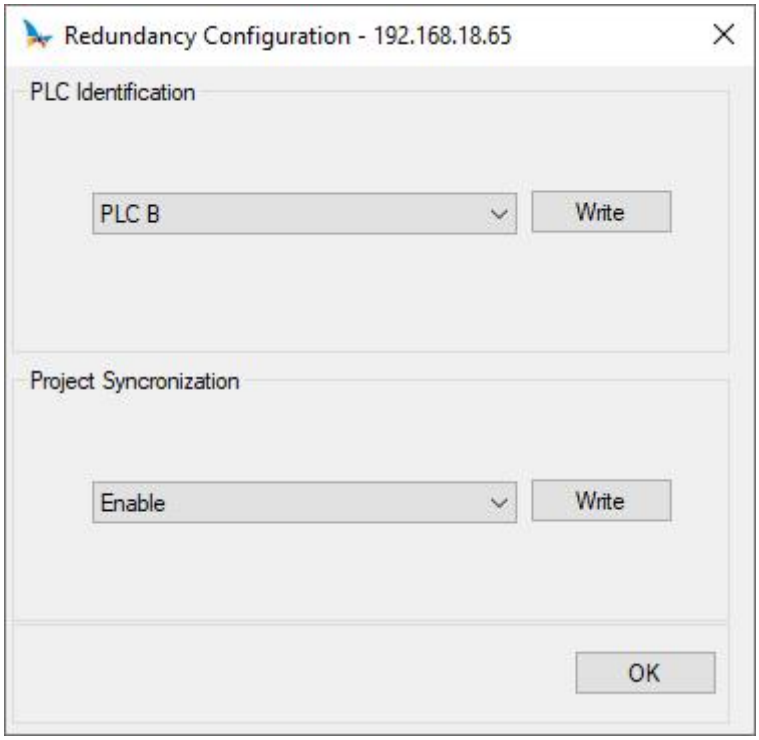


Figure 230: Redundancy Configuration

12. After the CPU restarts, use the OTD button and display on the NX3035 to check:
    a) If the IP address is correct, in the NETWORK menu.
    b) If the PLC identification (PLCA or PLCB) is correct, in the REDUNDANCY menu.
13. If the checks in the previous step are correct, you can proceed. Otherwise, restart this procedure from step 1.
14. If necessary, make the relevant settings for this PLC on the product's web page.
15. Set the active path of Mastertool for this PLC, repeating the procedure described in step 9.
16. Use the *Communication / Login* menu to download the project onto the PLC.

After performing this procedure on the PLCA and PLCB CPUs, remember to undo the temporary changes made on the computer in steps 6 and 7.

6.5.2. Mastertool Connection with an NX3035 CPU in a Redundant CPU Setup

After performing the procedure described in section Initial Downloading of a Redundant Project on both PLCs (PLCA and PLCB), the connection to Mastertool, through the CPU's NET 1 interface, can be made using one of the following addresses:

- IP address of CPU A: IP address of NET 1 exclusive to the CPU installed in PLCA
- IP address of CPU B: IP address of NET 1 exclusive to the CPU installed in PLCB

To connect to a specific PLC, first double-click on *Device* (NX3035) in the device tree, go to the *Communication Settings* tab, and press the *Map Network* button to list all PLCs detected by Mastertool on the network. The following figure shows an example of such a list.

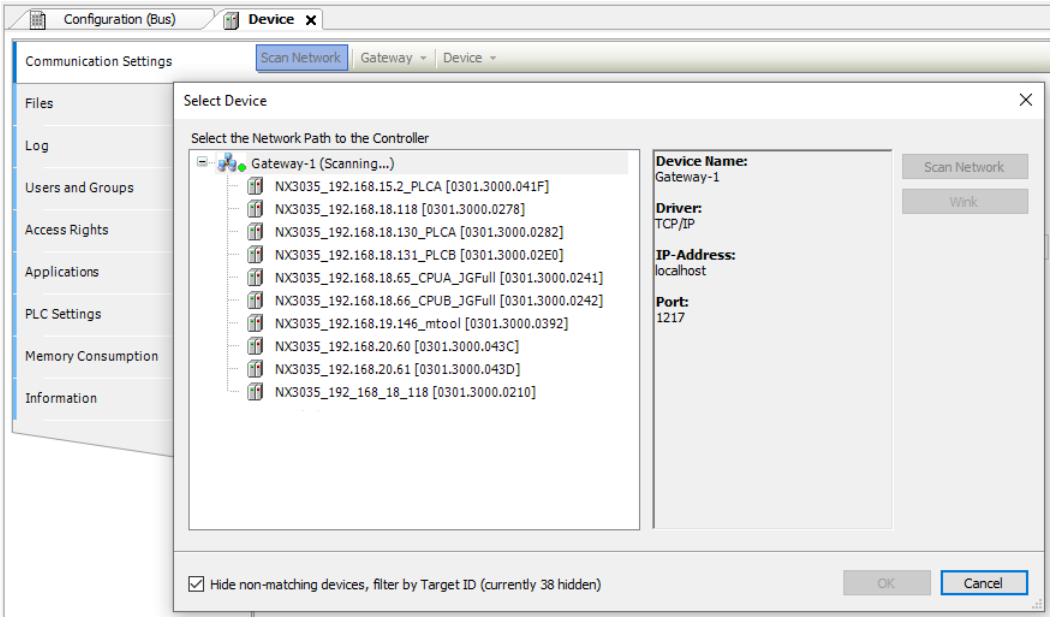


Figure 231: NX3035 CPUs Detected on the Network

The correct PLC in this list can be located by the meaningful name that was previously assigned to it (for example: GENERATOR_A – see step 10 of section Initial Downloading of a Redundant Project).

If you are unsure about the correct CPU in this list, which may occur if a meaningful name has not yet been assigned to the PLC, you can select the most likely PLC with a single click and press the *Flash* button. This will cause the CPU display on the selected PLC to flash, allowing you to determine whether you have selected the correct PLC.

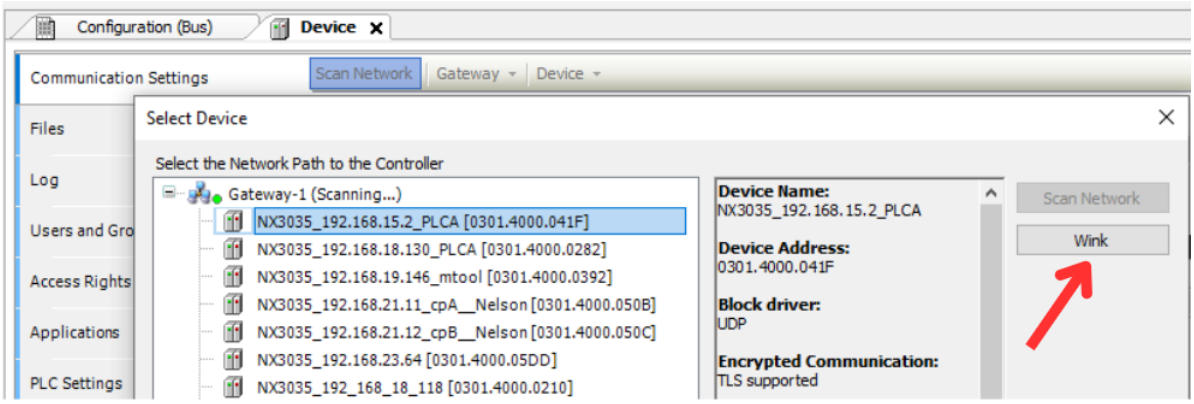


Figure 232: Using the Blink button to verify the selected CPU

By double-clicking on the PLC selected in the list, you define an active path that connects Mastertool to this PLC. The active path allows Mastertool to connect to a PLC to perform various functions. The following list shows just a few of these functions:

- Log in to the PLC. When doing so, if the project in Mastertool is different from the project in the PLC, the user will have the option to download the project.
- Give the PLC a meaningful name. If this has not already been done, it is recommended to do so as soon as possible (see step 10 of section Initial Downloading of a Redundant Project).
- Execute commands from the *Communication*, *Redundancy Configuration* menu, such as:
  - Change the PLC identification (PLCA, PLCB, or Non-Redundant)
  - Enable project synchronization

ATTENTION

Mastertool can only connect to one PLC at a time (there is only one active path in Mastertool at any given moment). To connect to multiple PLCs, you must open multiple instances of Mastertool, always taking care to open the correct project in each instance. It is not possible to open the same project with two different instances of Mastertool. If this is desired (for example, one instance connected to PLCA and another to PLCB for debugging purposes), you must create an additional copy of the project on your computer so that each instance of Mastertool opens a different project.

6.5.3. Modification Download in a Redundant Project

After the two PLCs (PLCA and PLCB) that form a redundant PLC have received an initial download, as described in the section Initial Downloading of a Redundant Project, it is possible to download successive modifications to the project as such modifications become necessary.

The connection of Mastertool to a PLC to execute the modifications download must be made as described in the section Mastertool Connection with an NX3035 CPU in a Redundant CPU Setup. This section explains how to connect to a specific PLC (PLCA or PLCB).

Normally, modifications must be downloaded on the Active PLC, and then they will be automatically synchronized to the Non-Active PLC through the NETA/NETB synchronization channels. Therefore, normally the user must find out which of the PLCs is in Active status and log in to it to download modifications to a project. The redundancy status can be found through the CPU display or through diagnostics that may be shown in the SCADA system, for example.

ATTENTION

There are some situations where it may be necessary to download design changes in Non-Active PLC. Such situations are described in the chapter Offline Program Download Without Interrupting Process Control.

6.5.4. Offline and Online Modification Download

Design modifications can be downloaded offline or online.

Offline downloads require the PLC where the modification is to be downloaded to be stopped. On the other hand, online downloads allow the PLC to continue running its application while the modification is downloaded.

Some types of modifications require offline download, meaning they cannot be loaded online on the PLC where Mastertool is connected. In this case, there are two options:

- Interrupt process control by performing the procedure described in section Offline Download of Modifications with Process Control Interruption.
- Use the redundancy of the PLC so as not to interrupt process control, even with the need to perform offline download on each of the half-clusters (PLCA or PLCB). The procedures for this case are described in chapter Offline Program Download Without Interrupting Process Control.

6.5.4.1. Modifications which Demand Offline Download and the Interruption of the Process Control

The following modifications to a project will make it impossible to download it into a redundant system without interrupting process control.

- Modifications to redundant memory areas for direct representation variables (%Q, %I, and %M), described in section NX3035 CPU Configurations Related to Redundancy.
- Modifications to task configurations.

ATTENTION

Modifications of this type prevent the execution of the procedures described in chapter Offline Program Download Without Interrupting Process Control. To avoid this situation, the configuration of these areas must be carefully planned with sufficient room for expansion.

6.5.4.2. Modifications which Demand Offline Download

The following modifications require offline downloads in the PLC, but do not prevent the execution of the procedures described in chapter Offline Program Download Without Interrupting Process Control:

- Add or remove devices in the device tree, for example:
  - Modules in the main backplane (PROFIBUS NX5001 masters, NX5000 Ethernet interfaces, etc.)
  - Remotes in PROFIBUS networks
  - I/O modules in PROFIBUS network remotes
  - MODBUS instances
- Modify parameters within existing devices in the device tree, for example:
  - IP addresses and other Ethernet interface parameters
  - PROFIBUS master parameters
  - PROFIBUS remote parameters
  - I/O module parameters within PROFIBUS network remotes
- Project update due to Mastertool programmer update.
- NX3035 CPU firmware updates, provided that the new version remains compatible with the previous one (example: version 1.a.b.c to version 1.e.f.g).

6.5.4.3. Modifications which Allow Online Download

In principle, modifications not mentioned in sections Modifications which Demand Offline Download and the Interruption of the Process Control and Modifications which Demand Offline Download, allow online download.

Even so, the main modifications that allow online download in the PLC where Mastertool is connected will be mentioned below. The modifications mentioned below apply to variables, POUs, and GVLs, whether redundant or not.

- Add program-type POUs, provided that these POUs do not need to be associated with any task and are not marked as redundant.
- Remove program-type POUs, provided that these POUs are not associated with any task.
- Add or remove function or functional block type POUs.
- Modify the code of any type of POU (program, function, or functional block).
- Add or remove symbolic variables in any type of POU (program, function, or functional block), whether redundant or not.
- Add or remove functional block instances in program or functional block POUs.
- Add or remove GVLs.
- Add or remove symbolic variables or functional block instances in GVLs.

6.5.5. Online Download of Modifications

An online upload must be performed by connecting Mastertool to an Active PLC interface. Therefore, the user must find out which of the PLCs is in Active status and log in to it to upload project modifications online. The redundancy status can be found through the CPU display or through diagnostics that may be shown in the SCADA system.

The procedure for connecting to a PLC was described in the section Mastertool Connection with an NX3035 CPU in a Redundant CPU Setup.

Next, the user must use the *Communication / Login* menu. For example, if Mastertool is connected to PLC GENERATOR_A, the following sequence of messages will be displayed:

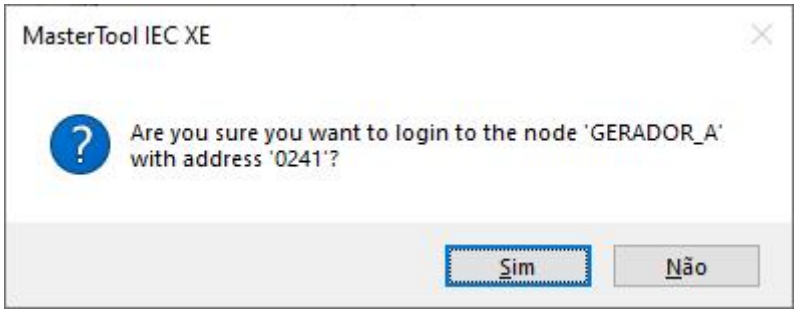


Figure 233: Device Confirmation Message

- Press the *Yes* button.

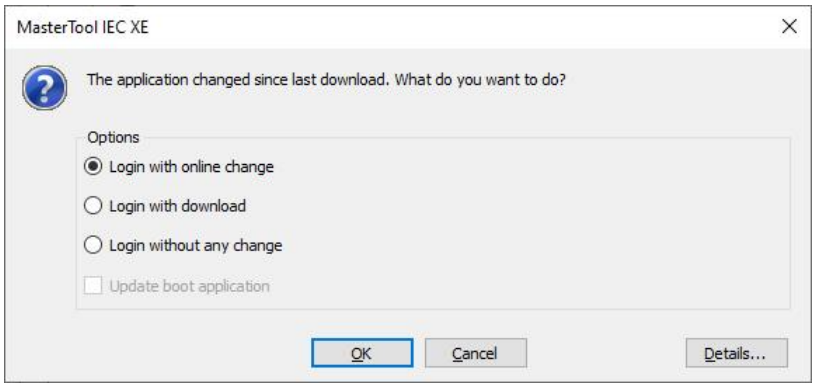


Figure 234: Download Option Message

- Select *Login with online change* and then the *OK* button.

After the online download finishes on the Active PLC, when the Non-Active PLC realizes that it has a different project from the Active PLC, it performs the following actions:

- Negotiates automatic project synchronization with the Active PLC.
- If it is in the Standby or Initializing state, it switches to the Unconfigured state and remains in this state until the projects are synchronized again. After that, it automatically returns to the Standby state.
- If it is in the Inactive state, a command must be executed to request that the PLC switch to the Standby state. When this occurs, the PLC first goes to the Unconfigured state, synchronizes the project, and finally attempts to go to the Standby state. The Standby state will be reached if no critical diagnostics prevent this.

6.5.6. Offline Download of Modifications with Process Control Interruption

This section defines the procedure for performing an offline download that will interrupt process control. This situation is acceptable in certain types of processes and during scheduled shutdowns of this process.

An offline Download must be performed by connecting Mastertool to channel NET 1 of the Active PLC. Therefore, the user must find out which of the PLCs is in Active status and log in to it to Download online modifications to a project. The redundancy status can be found through the CPU display or through diagnostics that may be shown in the SCADA system.

The procedure for connecting to a PLC was described in the section Mastertool Connection with an NX3035 CPU in a Redundant CPU Setup.

Next, the user must use the *Communication / Login* menu. For example, if Mastertool is connected to PLC GENERATOR_A, the following sequence of messages will be displayed:

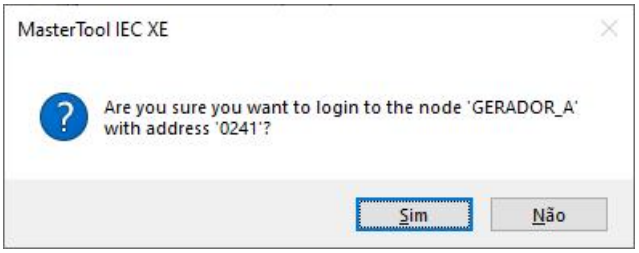


Figure 235: Login Confirmation Message

- Answer *Yes*.

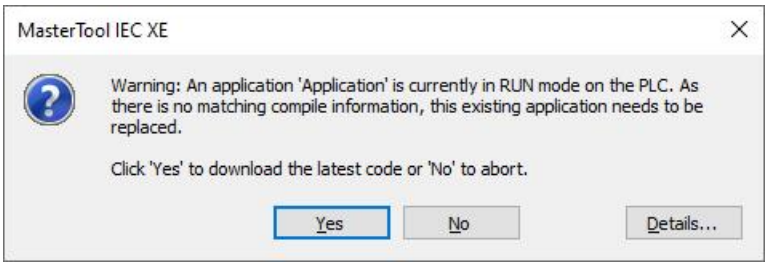


Figure 236: Unknown Application Message

- Answer *Yes*.
- After that, if the other PLC is in Standby status, the following message will appear:

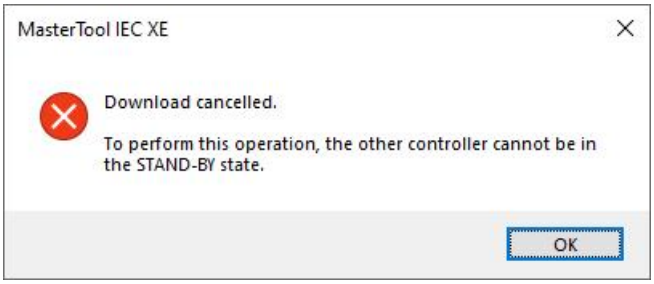


Figure 237: Message Download Cancelled

If the previous message appeared, change the status of the Standby PLC to Inactive, and restart the offline download process.

If the previous message did not appear, offline download is performed, and the Active PLC goes to the Unconfigured status in STOP mode, interrupting process control.

After that:

1. You must restart the application of this PLC by putting it in RUN mode. After some time initializing, this PLC should return to the Active status.
2. If the other PLC is in the Inactive status, you must request that it go to the Standby status. At this point, automatic project synchronization will occur, and when it is complete, the PLC will go to the Standby state.

6.6. Redundancy Maintenance

6.6.1. Hot Swapping of Modules in a Redundant CPU

In case of failure in any module of one of the PLCs (PLCA or PLCB), it may be necessary to replace the module without interrupting process control. To do so, the following steps must be followed:

1. Set the other PLC, which will not undergo maintenance, to Active status, if it is not already in this status.
2. Set the PLC that will undergo maintenance to Inactive status.
3. Perform the necessary hot swaps on the Inactive PLC, as indicated in chapter CPU Settings - General Parameters - How to do the Hot Swap.
4. Finally, return the Inactive PLC to the Standby state.
5. If desired, switch this Standby PLC to the Active state.

6.6.2. Mastertool Warning Messages

When Mastertool has a redundant project open, or when it is connected to an NX3035 CPU identified as PLCA or PLCB, some special warning messages may occur, as described in the following subsections.

6.6.2.1. Blocking the Loading of a Redundant or Non-Redundant Project

In some situations, Mastertool will block the loading of a project in the CPU to which it is connected via the active path:

- The loading of a redundant project will be blocked if the CPU model is not NX3035 or this CPU is not identified as PLCA or PLCB (see section Identification of an NX3035 CPU).
- The loading of a non-redundant project will be blocked if this CPU is identified as PLCA or PLCB (see section Identification of an NX3035 CPU).
- The loading of a redundant project will be blocked if the other redundant CPU is in Standby status.

If an attempt is made to perform any of these illegal actions, Mastertool will warn you with an appropriate message.

6.6.2.2. Alerts Before Commands That Can Stop the Active CPU

The following commands can stop a PLC:

- Put the PLC in STOP mode
- Debug / Stop
- Communication / Reset (Hot, Cold, Source)

If Mastertool is logged into the active PLC and an attempt is made to execute one of these commands, before sending them, Mastertool sends the following message and waits for authorization to send the command (OK) or to cancel it (Cancel):



Figure 238: Active PLC Stop Alert

A PLC may also stop if the user attempts to load a project offline on the Active PLC, and the other PLC is not in Standby mode. In this case, Mastertool sends the following message and waits for authorization to perform the offline load (Yes) or to cancel it (No):

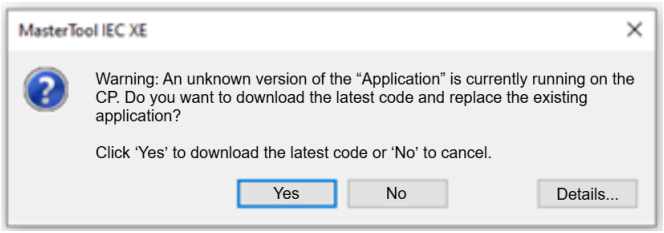


Figure 239: Active PLC Stop Alert after offline download

Another situation that can stop the PLC is the activation of a breakpoint in the Active PLC. In this case, the activation of the breakpoint in the Active PLC must be blocked if the other PLC is in Standby state, as described in the section Breakpoints Utilization in Redundant Systems.

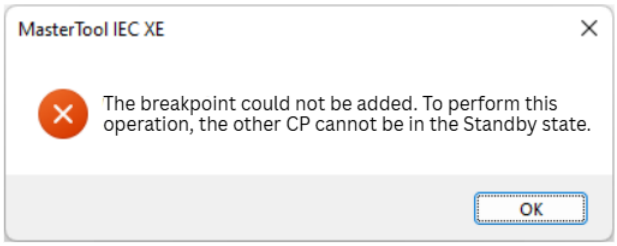


Figure 240: Breakpoint Activation Lock

6.6.2.3. Alert Before Logging into the Non-Active CPU

Under normal circumstances, it is not usual for Mastertool to log into the Non-Active PLC. When an attempt is made to execute a command of this type, Mastertool issues the following warning to confirm the operation (OK) or cancel it (Cancel).

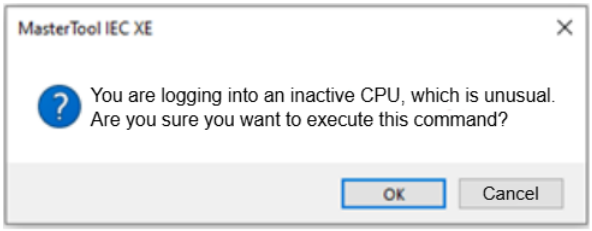


Figure 241: Warning of login attempt on the Non-Active PLC

There are circumstances (not very common) in which it is necessary to log in to the Non-Active PLC, and in these cases the user must authorize the login. Such circumstances may occur, for example:

- For initial configurations, as described in section Initial Downloading of a Redundant Project.
- To load a different project offline on the Non-Active PLC, as described in chapter Offline Program Download Without Interrupting Process Control.
- To monitor or force non-redundant variables on the Non-Active PLC.

6.6.3. Redundancy Diagnostics on the NX3035 CPU Graphical Display

Various diagnostics related to redundancy can be viewed on the NX3035 CPU display.

6.6.3.1. CPU Redundancy Status

The redundancy status of the PLC, described in Redundant CPU States, is shown in the first three characters of the second line of the main screen, as shown in section Graphic Display. The main screen of the display is shown at startup and reappears a few seconds after navigation is complete (without pressing the NX3035 CPU key).

6.6.3.2. Screens Below the REDUNDANCY Menu

There is a menu called REDUNDANCY, under which there are several screens. The description and access to the redundancy screens are available in section CPU's Informative and Configuration Menu of chapter Configuration.

6.6.4. Structure of Diagnostics, Commands, and User Information for Redundancy

The diagnostics, commands, and user information area for redundancy is mapped onto direct representation variables %Q, defined symbolically through AT directives, in GVL *System_Diagnostics*.

The data structure *DG_NX3035.tRedundancy* contains redundancy-specific diagnostics, commands, and user information, which will be described in this section and its subsections. This structure is divided into the following 6 data substructures:

- **RedDgnLoc:** Contains redundancy diagnostics related to the local PLC, for example, the redundancy status of the local PLC. This structure is described in section Redundancy Diagnostics.
- **RedDgnRem:** Contains redundancy diagnostics related to the remote PLC, for example, the redundancy status of the remote PLC. It is a copy of RedDgnLoc from the other PLC, received via NETA / NETB synchronization channels. This way, the local PLC has access to the remote PLC diagnostics. This structure is described in section Redundancy Diagnostics.
- **RedCmdLoc:** Contains redundancy commands that can be written to the local PLC, for example, through writes from a SCADA system, or generated in POUs executed on this PLC (e.g., ActivePrg or NonSkippedPrg). This structure is described in section Redundancy Commands.
- **RedCmdRem:** This is a copy of RedCmdLoc from the remote PLC, received via NETA / NETB synchronization channels. On the local PLC, this structure can only be read (attempts to write to it are discarded). Some of the commands received from the remote PLC may be executed on the local PLC. This structure is described in section Redundancy Commands.
- **RedUsrLoc:** Contains user-selectable information that can be exchanged between PLCA and PLCB. In the local PLC, this structure can be written to transmit information to the remote PLC. This structure is described in section User Information Exchanged between Local and Remote CPUs.
- **RedUsrRem:** This is a copy of RedUsrLoc from the remote PLC, received via NETA / NETB synchronization channels. On the local PLC, this structure can only be read (attempts to write to it are discarded). The information read on the local PLC via RedUsrRem is the same as that written in the RedUsrLoc structure of the remote PLC. This structure is described in section User Information Exchanged between Local and Remote CPUs.

It is important to note that the structures copied from the remote PLC are only updated when the Diagnostic and Command Exchange service, described in section Cyclic Synchronization Services through NETA and NETB, is successfully completed. This service is executed every cycle of the MainTask. In case of failure in this service:

- Remote PLC diagnostics may have obsolete values in the local PLC.
- Commands sent to the remote PLC may not be executed in the remote PLC.
- User information will not be exchanged between the local and remote PLCs.

The following subsections contain tables that define in detail each of the data structures for diagnostics, commands, and user information. The following observations should be considered when analyzing these tables:

- For a detailed view of the data types for each variable, access the *Library Manager* item in the device tree of the Mastertool window. Within the *Library Manager*, these types can be found in the *IoDrvNextoRedundancy* library.
- The variables are direct representation (%Q), but have symbolic names assigned through AT directives in GVL *System_Diagnostics*.
- The address %Q(n) corresponds to the starting address of the *DG_NX3035.tRedundancy* data structure.

6.6.4.1. Redundancy Diagnostics

Redundancy diagnostics are found in the following structures:

- **DG_NX3035.tRedundancy.RedDgnLoc:** local PLC diagnostics
- **DG_NX3035.tRedundancy.RedDgnRem:** remote PLC diagnostics

These diagnostics have several uses, for example:

- They can be consulted to check for any problems that need to be remedied.
- Whenever variations occur in them, such variations are inserted as events in the user logs. By consulting the historical sequence of such events, one can discover, for example, the cause of a switch-over.
- They can be referenced in the user application (ActivePrg or NonSkippedPrg). For example, one can test the redundancy status of the PLC and take specific actions based on this status.

ATTENTION

The *DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync* diagnosis indicates whether the Diagnostic and Command Exchange service was completed successfully, if it has the value TRUE. If the value of this diagnosis is FALSE, remote PLC diagnoses have not been copied and may contain obsolete values.

Since RedDgnRem is a copy of RedDgnLoc from the remote PLC, the two structures have the same format.

These structures are divided into five substructures:

- **sGeneral_Diag:** General redundancy diagnostics.
- **sNETA_Diag:** NETA synchronization channel diagnostics.
- **sNETB_Diag:** NETB synchronization channel diagnostics.
- **sNET_Stat:** Common statistics for the NETA and NETB synchronization channels, for counting successes and failures of synchronization services.
- **sGeneral_DiagExt:** General redundancy diagnostics, extended part (continuation of *sGeneral_Diag*).

The following table details the *sGeneral_Diag* substructure.

Variable AT DG_NX3035.tRedundancy. RedDgnLoc.sGeneral_Diag.*	**Description**
bConfigDone	TRUE – The configuration process, executed in the Non-Configured state, has completed.
	FALSE – The configuration process, executed in the Non-Configured state, has not yet finished or has not been executed.
bConfigError	TRUE – The configuration process, executed in the Non-Configured state, finished with errors. This is a system error, which is not normally expected. Contact Altus support to report it. Also provide the value of the *wConfigErrorCode* diagnostic to Altus support.
	FALSE – The configuration process was successful or was not performed.
bTooManyRedAreas	TRUE – The number of redundant areas has exceeded the maximum allowed number. This is a system error, which is not normally expected. Contact Altus support to report it.
	FALSE – The number of redundant areas is as expected.
bTemporaryBufferTooSmall	TRUE – Intermediate data structure with insufficient size. This is a system error, which is not normally expected. Contact Altus support to report it.
	FALSE – The data structure size is within the expected range.
bExchangeSync	TRUE – The Diagnostics and Commands Exchange synchronization service was successfully executed in this MainTask cycle.
	FALSE – The Diagnostics and Commands Exchange synchronization service was not successfully executed in this MainTask cycle. Invalid or outdated values may exist in the data structures and commands copied from the remote CP.
bRedDataSync	TRUE – The Redundant Data Synchronization service was successfully executed in this MainTask cycle.
	FALSE – The Redundant Data Synchronization service was not successfully executed in this MainTask cycle.
bRedForceSync	TRUE – The Redundant Forcing List Synchronization service was successfully executed in this MainTask cycle.
	FALSE – The Redundant Forcing List Synchronization service was not successfully executed in this MainTask cycle.
bApplicationIncompatible	TRUE – The application is not compatible between the two PLCs. A new application was downloaded to one of the PLCs with one of the following changes: - Modification of the redundant data area; - Modification of redundant symbolic variables. While this diagnostic is active, one of the PLCs will remain in the Inactive state until the same application is present on both CPs. This requires either reloading the previous application on both PLCs or updating both PLCs with the new application. For further information on how to proceed, refer to chapter Redundant CPU Program Downloading.
	FALSE – The application is compatible between the two PLCs.

Variable AT DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
bProjectSyncDisable	TRUE – The synchronization of the application project and the *Project archive* between the two PLCs will not be performed. This is a copy of the non-volatile variable used to disable project synchronization, as described in section Project Synchronization Disabling. Project synchronization can be disabled either on the local PLC or on the remote PLC; that is, executing the disable command on one of the PLCs is sufficient to disable project synchronization. The commands for disabling project synchronization are described in section Project Synchronization Disabling.
	FALSE – The synchronization of the application project and the *Project archive* between the two PLCs will be performed.
bIncompatibleFirmware	TRUE – The firmware version is incompatible between this PLC and the remote PLC, or there is a failure in the redundancy links.
	FALSE – The firmware version is compatible between this PLC and the remote PLC.
bApplicationProjectDiff	TRUE – The application project of this PLC is different from the application project of the remote PLC, or there is a failure in the redundancy links.
	FALSE – The application project of this PLC is the same as that of the remote PLC.
bProjectArchiveDiff	TRUE – The *Project archive* of this PLC is different from that present on the remote PLC, or there is a failure in the redundancy links.
	FALSE – The *Project archive* of this PLC is the same as that of the remote PLC
bOnlineChangeApply	TRUE – An online change has been performed in the application and has not yet been synchronized with the standby PLC.
	FALSE – No online changes have been performed in the application, or they have already been synchronized with the standby PLC.
bFailedPBUS1A	TRUE – This PLC is unable to communicate in the role of master (active or passive mode) on the PROFIBUS 1A network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This condition may also be reported if the NX5001 module has a microprocessor failure or if it is unable to communicate with the CPU via backplane.
	FALSE – There are no faults on the PROFIBUS 1A network.
bFailedPBUS1B	TRUE – This PLC is unable to communicate in the master role (active or passive mode) on the PROFIBUS 1B network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a fault in its microprocessor or if it is unable to communicate with the CPU via backplane.
	FALSE – There are no faults on the PROFIBUS 1B network.
bFailureProfibus_1	TRUE – This PLC cannot communicate in the master role (active or passive mode) on the PROFIBUS 1 network. If the PROFIBUS 1 network is redundant, *bFailureProfibus_1* results from a logical "AND" between *bFailedPBUS1A* and *bFailedPBUS1B*. If the PROFIBUS 1 network is not redundant, *bFailureProfibus_1* is a copy of *bFailedPBUS1A*.
	FALSE – There are no faults on the PROFIBUS 1 network.
bFailedPBUS2A	TRUE – This PLC cannot communicate in the master role (active or passive mode) on the PROFIBUS 2A network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it cannot communicate with the CPU via backplane.
	FALSE – There are no faults on the PROFIBUS 2A network.
bFailedPBUS2B	TRUE – This PLC cannot communicate in the master role (active or passive mode) on the PROFIBUS 2B network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it cannot communicate with the CPU via backplane.
	FALSE – There are no faults on the PROFIBUS 2B network.

Variable AT DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
bFailureProfibus_2	TRUE – This PLC cannot communicate in the master role (active or passive mode) on the PROFIBUS 2 network. If the PROFIBUS 2 network is redundant, *bFailureProfibus_2* results from a logical AND between *bFailedPBUS2A* and *bFailedPBUS2B*. If the PROFIBUS 2 network is not redundant, *bFailureProfibus_2* is a copy of *bFailedPBUS2A*.
	FALSE – There are no faults on the PROFIBUS 2 network.
bProfibusVitalFailureAny	TRUE – This PLC cannot communicate in the master role (active or passive mode) with at least one PROFIBUS network configured in vital failure mode.
	FALSE – There are no failures in the PROFIBUS networks configured in vital failure mode.
bProfibusVitalFailureAll	TRUE – This PLC cannot communicate in the master role (active or passive mode) with all PROFIBUS networks configured in vital failure mode.
	FALSE – At least one of the PROFIBUS networks configured in vital failure mode has no failures.
ePLC_ID	This diagnostic reports the identification of this PLC: - 0 = non-redundant - 2 = PLC_A - 3 = PLC_B This is a copy of the non-volatile variable used to identify the PLC, as described in section Identification of an NX3035 CPU. Section Initial Downloading of a Redundant Project describes the Mastertool command used to write to this non-volatile variable.
eRedState	Reports the redundancy state of this PLC: - Non-Configured = 0 - Initializing = 2 - Standby = 3 - Active = 4 - Inactive = 5
ePreviousRedState	Value that the RedState diagnostic had prior to the last state transition.
wRedStateDuration	Measures how long (in milliseconds) the current redundancy state has been active. This time stops incrementing when it reaches 65,535 ms.
wConfigErrorCode	Error code detected during the configuration process while in the Non-Configured state. See the *bConfigError* diagnostic described earlier.
dwApplicationCRC	32-bit CRC of the application, used to detect differences between the applications of the two PLCs.
dwArchiveCRC	32-bit CRC of the *project archive*, used to detect differences between the *project archives* of the two PLCs.
dwFirmwareVersion	Firmware version of this PLC, used to verify compatibility between the firmware of the two PLCs.
dwIECTimer	Synchronization of the IEC Timer is required for collision-free operation of certain function blocks, such as TON and TOF. Through this diagnostic, the IEC Timer of the Active PLC is received and updated in the Non-Active PLC, provided that the Diagnostics and Commands Exchange service has been successfully executed. Its count starts at 0 and increments up to 4,294,967,295. After counter overflow, it restarts at 0.
wCycleCounter	16-bit counter used as auxiliary sequence information in the Redundancy Event Logs. In the Active PLC, it is incremented on each MainTask cycle. In the Non-Active PLC, it receives a copy of the value present in the Active PLC, provided that the Diagnostics and Commands Exchange service has been successfully executed. Its count starts at 0 and increments up to 65,535. After counter overflow, it restarts at 0.

Table 157: Substructure *sGeneral_Diag*

ATTENTION

When the variable *DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync* has a value equal to FALSE, it is not possible to define the status of some other diagnostics in the previous table that depend on communication with the remote PLC, such as *bIncompatibleFirmware*, *bApplicationProjectDiff*, and *bProjectArchiveDiff*.

The following table details the *sNETA_Diag* substructure, which contains diagnostics for the NETA synchronization channel:

Variable AT DG_NX3035.tRedundancy.RedDgnLoc.sNETA_Diag.*	Description
bGeneralFailure	TRUE – The synchronization channel has some type of failure. The next three diagnostics will indicate the specific fault.
	FALSE – The synchronization channel is operating correctly.
bInternalFailure	TRUE – The detected fault has its cause located within this PLC. Such faults are handled differently.
	FALSE – There are no internal faults on the synchronization channel.
bLinkDownFailure	TRUE – Failure in the fiber-optic connection on the receive channel, or a fault in the SFP module (NX9500 or NX9501).
	FALSE – Absence of these faults.
bTimeoutFailure	TRUE – This fault is reported when a synchronization service does not complete successfully within a specified timeout, and no faults of type *bInternalFailure* or *bLinkDownFailure* are detected that would justify it.
	FALSE – No timeout faults.

Table 158: Substructure *sNETA_Diag*

The following table details the *sNETB_Diag* substructure, which contains diagnostics for the NETB synchronization channel:

Variable AT DG_NX3035.tRedundancy.RedDgnLoc.sNETB_Diag.*	Description
bGeneralFailure	TRUE – The synchronization channel has a failure. The next 3 diagnostics will indicate the specific failure.
	FALSE – The synchronization channel is functioning correctly.
bInternalFailure	TRUE – The detected fault has its cause located within this PLC. Such faults are handled differently.
	FALSE – There are no internal faults in the synchronization channel.
bLinkDownFailure	TRUE – Failure in the optical fiber connection on the receive channel, or a fault in the SFP module (NX9500 or NX9501).
	FALSE – Absence of these faults.
bTimeoutFailure	TRUE – This fault is reported when a synchronization service does not complete successfully within a specified timeout and no faults of type *bInternalFailure* or *bLinkDownFailure* are detected that would justify it.
	FALSE – No timeout faults.

Table 159: Substructure *sNETB_Diag*

The following table details the *sNET_Stat* substructure, which contains statistical counters for failures and successes for services executed through the NETA and NETB synchronization channels.

Variable AT DG_NX3035.tRedundancy.RedDgnLoc.sNET_Stat.*	Description
wSuccessExchDgCmdSync	Count of successful executions of the Diagnostics and Commands Exchange service. (0 a 65535)
wFailedExchDgCmdSync	Count of failures of the Diagnostics and Commands Exchange service. (0 a 65535)
wSuccessRedDataSync	Count of successes of the Redundant Data Synchronization service. (0 a 65535)
wFailedRedDataSync	Count of failures of the Redundant Data Synchronization service. (0 a 65535)
wSuccessRedForceSync	Count of successful executions of the Redundant Forcing List Synchronization service. (0 a 65535)
wFailedRedForceSync	Count of failures of the Redundant Forcing List Synchronization service. (0 a 65535)

Table 160: Substructure *sNET_Stat*

Notes on the counters in the previous table:

- All counters return to zero when their maximum value (65535) is exceeded.
- It is possible to reset all statistical counters shown in the previous table by writing TRUE the following commands:
  - **DG_NX3035.tRedundancy.RedCmdLoc.bResetNETStatisticsLocal:** resets local PLC counters.
  - **DG_NX3035.tRedundancy.RedCmdLoc.bResetNETStatisticsRemote:** resets remote PLC counters.

The following table details the *sGeneral_DiagExt* substructure, which is a continuation of the diagnostics of the *sGeneral-Diag* substructure described above.

Variable AT DG_NX3035.tRedundancy. RedDgnLoc.sGeneral_DiagExt.*	Description
bFailedPBUS3A	TRUE – This PLC is unable to communicate as a master (active or passive mode) on the PROFIBUS 3A network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or is unable to communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 3A network.
bFailedPBUS3B	TRUE – This PLC is unable to communicate in the master role (active or passive mode) on the PROFIBUS 3B network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it is unable to communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 3B network.
bFailureProfibus_3	TRUE – This PLC is unable to communicate in the master role (active or passive mode) on the PROFIBUS 3 network. If the PROFIBUS 3 network is redundant, *bFailureProfibus_3* results from a logical "AND" between *bFailedPBUS3A* and *bFailedPBUS3B*. If the PROFIBUS 3 network is not redundant, *bFailureProfibus_3* is a copy of *bFailedPBUS3A*.
	FALSE – There are no failure on the PROFIBUS 3 network.
bFailedPBUS4A	TRUE – This PLC is unable to communicate in the role of master (active or passive mode) on the PROFIBUS 4A network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a failure in its microprocessor, or if it is unable to communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 4A network.
bFailedPBUS4B	TRUE – This PLC is unable to communicate in the role of master (active or passive mode) on the PROFIBUS 4B network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the master operating in active mode) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a failure in its microprocessor, or if it is unable to communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 4B network.
bFailureProfibus_4	TRUE – This PLC is unable to communicate in the role of master (active or passive mode) on the PROFIBUS 4 network. If the PROFIBUS 4 network is redundant, *bFailureProfibus_4* results from a logical "AND" between *bFailedPBUS4A* and *bFailedPBUS4B*. If the PROFIBUS 4 network is not redundant, *bFailureProfibus_4* is a copy of *bFailedPBUS4A*.
	FALSE – There are no failure on the PROFIBUS 4 network.
bFailureCommBusAny	TRUE – There is a vital failure on at least one Ethernet communication network configured for vital failure.
	FALSE – No Ethernet communication network configured for vital failure presents a failure.
bFailureCommBusAll	TRUE – There is a failure in all Ethernet communication networks configured for vital failure.
	FALSE – There is no failure in all Ethernet communication networks configured for vital failure.
byFailedCommBusCount	Number of Ethernet field communication networks reporting failures.
bRemCpuKeepAliveNet1	TRUE – The CPU is receiving Keep Alive packets from the CPU of the other half-cluster, through the NET1 Ethernet port. Keep Alive packets are only sent in projects that do not have any PROFIBUS network.

Variable AT DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_DiagExt.*	Description
	FALSE – Keep Alive packets are not being received through the NET1 Ethernet port.
bRemCpuKeepAliveNet2	TRUE – The CPU is receiving Keep Alive packets from the CPU of the other half-cluster, through the NET2 Ethernet port. Keep Alive packets are only sent in projects that do not have any PROFIBUS network.
	FALSE – Keep Alive packets are not being received through the NET2 Ethernet port.
bFailedPBUS5A	TRUE – This PLC cannot communicate as a master (active or passive mode) on the PROFIBUS 5A network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active mode master) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it cannot communicate with the CPU via backplane
	FALSE – There are no failure on the PROFIBUS 5A network.
bFailedPBUS5B	TRUE – This PLC cannot communicate as a master (active or passive mode) on the PROFIBUS 5B network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active mode master) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it cannot communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 5B network.
bFailureProfibus_5	TRUE – This PLC cannot communicate as a master (active or passive mode) on the PROFIBUS 5 network. If the PROFIBUS 5 network is redundant, *bFailureProfibus_3* results from a logical "AND" between *bFailedPBUS5A* and *bFailedPBUS5B*. If the PROFIBUS 5 network is not redundant, *bFailureProfibus_5* is a copy of *bFailedPBUS5A*.
	FALSE – There are no failure on the PROFIBUS 5 network.
bFailedPBUS6A	TRUE – This PLC cannot communicate as a master (active or passive mode) on the PROFIBUS 6A network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active mode master) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it cannot communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 6A network6
bFailedPBUS6B	TRUE – This PLC cannot communicate as a master (active or passive mode) on the PROFIBUS 6B network. The active mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active mode master) is assumed by the Non-Active PLC. This failure may also be indicated if the NX5001 module has a microprocessor failure or if it cannot communicate with the CPU via backplane.
	FALSE – There are no failure on the PROFIBUS 6B network.
bFailureProfibus_6	TRUE – This PLC cannot communicate as a master (active or passive mode) on the PROFIBUS 6 network. If the PROFIBUS 6 network is redundant, *bFailureProfibus_4* results from a logical "AND" between *bFailedPBUS6A* and *bFailedPBUS6B*. If the PROFIBUS 6 network is not redundant, *bFailureProfibus_6* is a copy of *bFailedPBUS6A*.
	FALSE – There are no failure on the PROFIBUS 6 network.
bAdditionalSettingsDiff	TRUE – The web configurations for Firewall, SNMP, FTP, SysLog, Memory Card, or the user access rights in this PLC differ from the configurations in the remote PLC, or there is a failure in the redundancy links.
	FALSE – The configurations and user access rights of this PLC are identical to those of the remote PLC.
dwAdditionalSettingsCRC	32-bit CRC of the web configurations for Firewall, SNMP, FTP, SysLog, Memory Card, and user access rights, used to detect differences between the configurations of the two PLCs.

Table 161: Substructure *sGeneral_DiagExt*

6.6.4.2. Redundancy Commands

The redundancy commands are in the following structures:

- **DG_NX3035.tRedundancy.RedCmdLoc:** local PLC commands

- **DG_NX3035.tRedundancy.RedCmdRem:** remote PLC commands

These commands have several uses, for example, requesting a change in the redundancy status of the local PLC or remote PLC.

ATTENTION

The *DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync* diagnosis indicates whether the Diagnostic and Command Exchange service was completed successfully, if it has the value TRUE. If the value of this diagnosis is FALSE, commands from the local PLC have not been copied to the remote PLC and therefore will not be executed on the remote PLC.

As *RedCmdRem* is a copy of *RedCmdLoc* from the remote PLC, the two structures have the same format.

The commands of the *RedCmdLoc* and *RedCmdRem* structures always have a suffix that can be Local or Remote. Commands with the Local suffix must be executed on the local PLC, while commands with the Remote suffix must be executed on the remote PLC. For example:

- By writing TRUE in the *RedCmdLoc.bInactiveLocal* command of the local PLC, we are requesting that the local PLC change its redundancy status to Inactive.
- By writing TRUE in the *RedCmdLoc.bInactiveRemote* command of the local PLC, we are requesting that the remote PLC change its redundancy state to Inactive. This command will be received at the remote PLC in RedCmdRem.bInactiveRemote, and then the remote PLC will execute the state change to Inactive.

It should be remembered that, in each cycle of MainTask, the *RedCmdLoc* structure of the local PLC is transmitted to the *RedCmdRem* structure of the remote PLC.

To trigger a command, the corresponding bit in *RedCmdLoc* must always be set. This can be done by a SCADA system, writing via Mastertool, or even by turning on the command bit within a POU such as *ActivePrg* or *NonSkippedPrg*.

The user does not need to worry about turning off the command bit, as this will be done automatically by the redundancy manager:

- In the case of commands executed on the local PLC itself (RedCmdLoc + command with Local suffix), the bit is turned off as soon as the command is recognized and executed by the local PLC itself.
- In the case of commands sent to the remote PLC to be executed on the remote PLC (*RedCmdLoc* + command with Remote suffix):
  - On the remote PLC where the command is received in *RedCmdRem*, the command is executed when the redundancy manager perceives a rising edge on the command bit.
  - At the local PLC where the command was written to *RedCmdLoc*, the bit is automatically turned off in the next MainTask cycle.

It is important to note that all command activations or deactivations are recorded in the redundancy event logs, allowing historical evaluation, for example, to identify the causes of a switch-over.

The following table details the *RedCmdLoc* and *RedCmdRem* structures, which have identical formats.

Variable AT DG_NX3035.tRedundancy. RedCmdLoc.*	**Description**
bStandbyLocal	TRUE – Requests the local PLC to switch to the Standby state.
	FALSE – No command.
bInactiveLocal	TRUE – Requests the local PLC to transition to the Inactive state.
	FALSE – No command.
bResetNETStatisticsLocal	TRUE – This command resets the statistics of NETA / NETB (see sub-structure *SNET_Stat* in *RedDgnLoc* and *RedDgnRem*).
	FALSE – No command.
bStandbyRemote	TRUE – Requests that the remote PLC change to the Standby state.
	FALSE – No command.
bInactiveRemote	TRUE – Requests that the remote PLC change to the Inactive state.
	FALSE – No command.
bResetNETStatisticsRemote	TRUE – This command resets the NETA / NETB statistics of the remote PLC (see the *SNET_Stat* sub-structure in *RedDgnRem*).
	FALSE – No command.

Table 162: *RedCmdLoc* and *RedCmdRem* Structures

6.6.4.3. User Information Exchanged between Local and Remote CPUs

The user information exchanged between local and remote PLCs is in the following structures:

- **DG_NX3035.tRedundancy.RedUsrLoc:** information transmitted from the local PLC to the remote PLC
- **DG_NX3035.tRedundancy.RedUsrRem:** information transmitted from the remote PLC to the local PLC

These data structures can be used to bring non-redundant information from the local PLC to the remote PLC, or vice versa. In each structure (*RedUsrLoc* and *RedUsrRem*), the user has a 128-byte array available to transmit the information they want. As an example of such information, we can mention non-redundant diagnostics that are not mapped in the *RedDgnLoc* and *RedDgnRem* structures.

ATTENTION

The *DG_NX3035.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync* diagnostic indicates whether the Diagnostic and Command Exchange service was completed successfully, if it has the value TRUE. If the value of this diagnosis is FALSE, the remote PLC user information has not been copied and may contain obsolete values.

Since *RedUsrRem* is a copy of *RedUsrLoc* from the remote PLC, both structures have the same format. This is the array:

```
abyUser : ARRAY [1 .. 128] OF BYTE;
```

6.6.4.4. MODBUS Diagnostics Used in Redundancy

To check for failures in all MODBUS Servers configured in a MODBUS Client instance, there is a specific diagnostic in each configured MODBUS Client instance. The table below shows the diagnostic 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 on this client are reporting an error.
	FALSE – There is at least one server configured on this client that is not reporting an error.

Table 163: MODBUS Client Diagnostics

When configured with Ethernet Failure Mode, this diagnostic is consulted and 3 seconds after its status changes from FALSE to TRUE, a switch-over occurs if the other conditions for this event are met.

6.6.4.5. Redundancy Event Log

Mastertool allows users to view various logs for a PLC Nexto, including the Redundancy Event Log. These messages, specific to redundancy, record relevant changes in fields of diagnostic data structures and redundancy commands in the *System Log*.

Each line shown in the log has the following columns:

- **Severity:** information, warning, error, or exception.
- **Timestamp:** date and time of the event.
- **Description:** text describing the event.
- **Component:** component that generated the event, which in the case of the Redundancy Event Log will be *IoDrvNextoRedundancy*.

The text in the *Description* column contains information about the event that occurred.

An example of the *Description* column could be the following:

Redundancy new state (local): Starting

To access this screen, double-click on the device (NX3035) in the device tree, and then select the *Log* tab. There is a filter that allows you to select only the *IoDrvNextoRedundancy* component, to display only redundancy events.

ATTENTION

Some diagnostics may indicate possible failures during the initialization of the redundant system and in the first cycles of the tasks. However, when the system is functioning correctly, these diagnostics will indicate the absence of errors shortly after the system initialization.

7. Maintenance

One feature of the Nexto Series is the abnormality diagnostic generation, whether they are failures, errors or operation modes, allowing the operator to identify and solve problems which occurs in the system easily.

The Nexto CPUs permit many ways to visualize the diagnostics generated by the system, which are:

- One Touch Diag
- Diagnostics via LED
- Diagnostics via System Web Page
- Diagnostics via Variables
- Diagnostics via Function Blocks

The first one is an innovating feature of Nexto Series, which allows a fast access to the application abnormal conditions. The second is purely visual, generated through two LEDs placed on the panel (DG and WD) and also through the LEDs placed in the RJ45 connector (exclusive for Ethernet connection). The next feature is the graphic visualization in a WEB page of the rack and the respective configured modules, with the individual access allowed of the operation state and the active diagnostics. The diagnostics are also stored directly in the CPU variables, either direct representation (%Q) or attributed (AT variable), and can be used by the user application, for instance, being presented in a supervisory system. The last ones present specific conditions of the system functioning.

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

7.1. Module Diagnostics

7.1.1. One Touch Diag

The One Touch Diag (OTD), or single touch diagnostics, is an exclusive feature the Nexto Series brings for the programmable controllers. With this new concept the user can verify the diagnostics of any module connected to the system straight on the CPU graphic display with a single touch on the module Diagnostic Switch. This is a powerful diagnostic tool which can be used offline (with no need of supervisory or programming software) making easier to find and solve quickly possible problems.

The diagnostics key is placed on the CPU upper part, in an easy access place and, besides giving active diagnostics, allows the access to the navigation menu, described in the Configuration – CPU's Informative and Configuration Menu section.

The figure below shows the CPU switch placement:

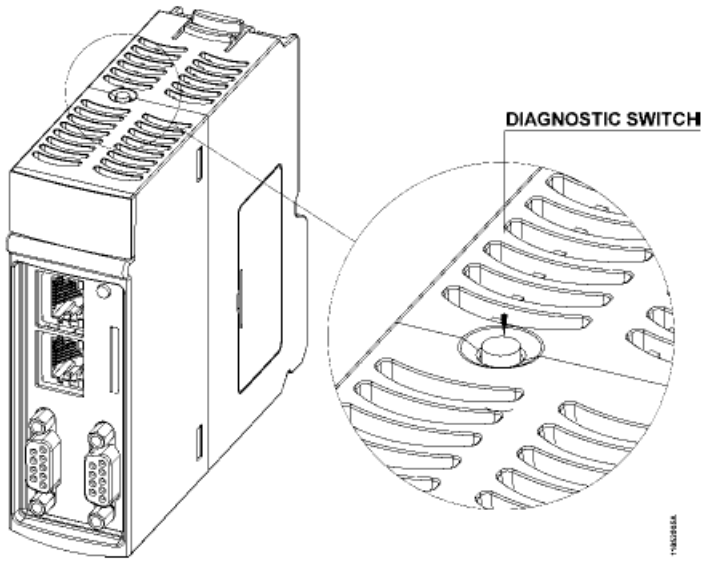


Figure 242: Diagnostic Switch

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 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.

The figure below shows the process starting with the short touch, with the conditions and the CPU times presented in smaller rectangles. It is important to stress the diagnostics may have more than one screen, in other words, the specified time in the block diagram below is valid for one of them.

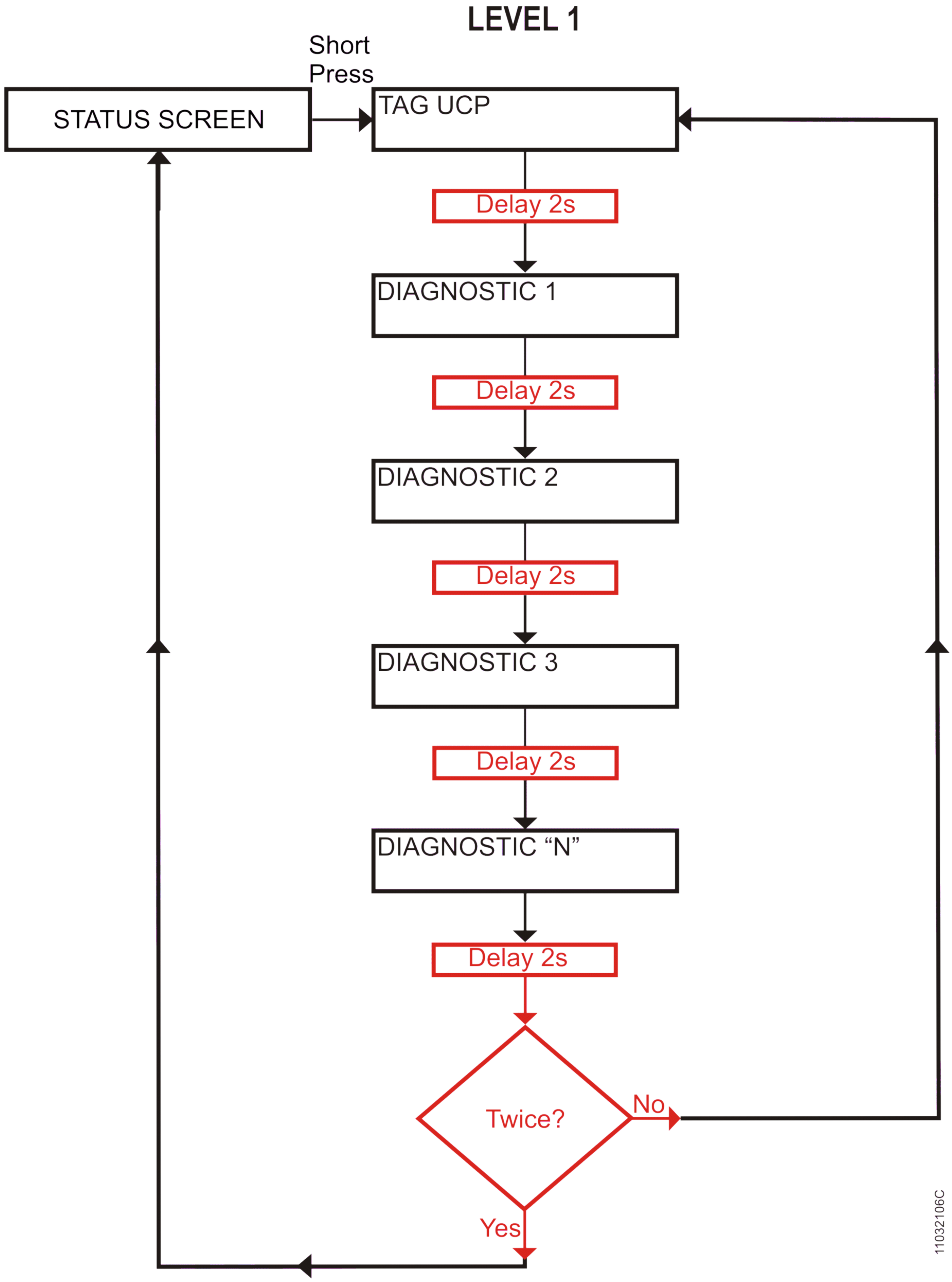


Figure 243: CPU Diagnostics Visualization

Before all visualization process be concluded, it is just to give a short touch on the diagnostic switch, at any moment, or press the diagnostic switch from any I/O module connected to the bus. Also, it is important to observe that the One Touch Diag could be available when the module could be in Operational Mode.

In case a long touch is executed, the CPU goes to navigation menu, which is described in the Configuration – CPU's Informative and Configuration Menu section.

The table below shows the difference between the short touch time, the long touch time and stuck button.

Touch type	Minimum time	Maximum time	Indication condition
No touch	-	59.99 ms	-
Short touch	60 ms	0.99 s	Release
Long touch	1 s	20 s	More than 1 s till 20 s
Locked Switch	20.01 s	(∞)	Diagnostics indication, see on Table 168

Table 164: 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 168.

If any situation of stuck button occur in one of the I/O modules, the diagnostic button of this module will stop of indicate the diagnostics on CPU graphic display when is pressed. In this case, the CPU will indicate that there is a module with active diagnostics. To remove this diagnostic from the CPU, a hot swap must be done in the module where the diagnostic is active.

For further details on the procedure for viewing the diagnostics of the CPU or other bus modules, see description in the User Manual Nexto Series – MU214600.

7.1.2. Diagnostics via LED

This product have a LED for diagnostic indication (LED DG) and a LED for watchdog event indication (LED WD). The Tables 165 and 166 show the meaning of each state and its respective descriptions.

7.1.2.1. DG (Diagnostic)

Green	Red	Description	Causes	Priority
Off	**Off**	Not used	No power supply. Hardware problem	-
On	**Off**	All applications in execution mode (Run)	-	3 (Low)
Off	**On**	All applications in stopping mode (Stop)	-	3 (Low)
Blinking 2x	**Off**	Bus modules with diagnostic	At least, a bus module, including the CPU, is with an active diagnostic	1
Blinking 3x	**Off**	Data forcing	Some memory area is being forced by the user through Mastertool	2
Off	**Blinking 4x**	Configuration or hardware error in the bus	The bus is damaged or is not properly configured	0 (High)

Table 165: Description of the Diagnostic LEDs States

7.1.2.2. WD (Watchdog)

Red LED	Description	Causes	Priority
Off	No watchdog indication	Normal operation	3 (Low)
Blinking 1x	Software watchdog	User application watchdog	2
On	Hardware watchdog	Damaged CPU and/or corrupted operational system	1 (High)

Table 166: 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 172.

Hardware Watchdog: In order to reset any watchdog indication, as in the WD LED or in the Reset.bWatchdogReset operand, the CPU must be disconnected from the power supply and wait until the watchdog LED (WD LED) turns off.

In order to verify the application conditions in the CPU restart, see configurations on Table 36.

7.1.2.3. RJ45 Connector LEDs

Both LEDs placed in the RJ45 connectors, help the user in the installed physical network problem detection, indicating the network Link speed and the existence of interface communication traffic. The LEDs meaning is presented on table below.

Yellow	Green	Description
○	○	Network LINK absent
○	●	1000 Mbytes/s network LINK
●	○	100 Mbytes/s network LINK
●	●	10 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 167: Ethernet LEDs Meaning

7.1.3. Diagnostics via System Web Page

In addition to the features previously presented, the Nexto Series provides users with an innovative tool for accessing diagnostics and system operating states through the *Bus Information* tab of the System Web Page.

Its use is not only dynamic, but also highly intuitive, making user operations easier. In other words, it can replace the use of a supervisory system when the purpose is limited to monitoring the system status.

To access the System Web Page of the desired CPU, simply use a desktop browser and enter the IP address corresponding to the CPU in the address bar (e.g., http://192.168.1.1). For more details, refer to the section System Web Page.

7.1.4. Diagnostics via Variables

The Nexto Series CPUs have many variables for diagnostic indication. There are data structures with the diagnostics of all modules declared on the bus, mapped on the variables of direct representation %Q, and defined symbolically through the AT directive, in the GVL System_Diagnostics created automatically by the Mastertool.

The CPU diagnostics are divided in the categories bellow:

- Summarized Diagnostics

- Detailed Diagnostics
- Redundancy Diagnostics

The size in bytes of each diagnostic structure shall be determined using the SIZEOF instruction.

The detailed description of the redundancy diagnostics is available in section Structure of Diagnostics, Commands, and User Information for Redundancy of this manual.

7.1.4.1. Summarized Diagnostics

The table below shows the meaning of each CPU summarized diagnostic bit:

Diagnostic Message	Variable DG_Modulo.tSummarized.*	Description
NO DIAG	-	There is no active diagnostic.
CONFIG. MISMATCH	bConfigMismatch	TRUE – There is a configuration issue on the bus, such as a module installed in the wrong slot.
		FALSE – The bus is correctly configured.
ABSENT MODULES	bAbsentModules	TRUE – One or more declared modules are missing.
		FALSE – All declared modules are present on the bus.
SWAPPED MODULES	bSwappedModules	TRUE – Two modules are swapped in their positions on the bus.
		FALSE – No modules are swapped on the bus.
UNDECLARED MODULES	bNonDeclaredModules	TRUE – One or more modules present on the bus are not declared.
		FALSE – All modules present on the bus are declared.
MODULES WITH DIAGNOSTIC	bModulesWithDiagnostic	TRUE – One or more bus modules have an active diagnostic.
		FALSE – No active diagnostics are present in the bus modules.
MODULES WITH FATAL ERROR	bModuleFatalError	TRUE – One or more bus modules are in a non-functional state.
		FALSE – All modules present on the bus are functional.
MODULES WITH PARAMETER ERROR	bModuleParameterError	TRUE – One or more bus modules have a parameterization error.
		FALSE – All modules are properly parameterized.
BUS ERROR	bWHSBBusError	TRUE – The master indicates a failure in the WHSB bus.
		FALSE – The WHSB bus is operating correctly.
HARDWARE FAILURE	bHardwareFailure	TRUE – CPU hardware failure.
		FALSE – The hardware is operating correctly.
SOFTWARE EXCEPTION	bSoftwareException	TRUE – One or more software exceptions were generated.
		FALSE – No software exceptions have been generated.
MEMORY CARD ERROR	bMemoryCardError	TRUE – The memory card is inserted in the CPU but is not operating correctly.
		FALSE – The memory card is operating correctly.
COM1 CONFIG. ERROR	bCOM1ConfigError	TRUE – An error occurred during or after configuration of the serial interface COM 1.
		FALSE – The COM 1 serial interface configuration is correct.
NET 1 CONFIG. ERROR	bNET1ConfigError	TRUE – An error occurred during or after configuration of the Ethernet interface NET 1.

Diagnostic Message	Variable DG_Modulo.tSummarized.*	Description
		FALSE – The NET 1 Ethernet interface configuration is correct.
NET 2 CONFIG. ERROR	bNET2ConfigError	TRUE – An error occurred during or after configuration of the Ethernet interface NET 2.
		FALSE –The NET 2 Ethernet interface configuration is correct.
NET 3 CONFIG. ERROR	bNET3ConfigError	TRUE – An error occurred during or after configuration of the Ethernet interface NET 3.
		FALSE – The NET 3 Ethernet interface configuration is correct.
NET 4 CONFIG. ERROR	bNET4ConfigError	TRUE – An error occurred during or after configuration of the Ethernet interface NET 4.
		FALSE – The NET 4 Ethernet interface configuration is correct.
NET 5 CONFIG. ERROR	bNET5ConfigError	TRUE – An error occurred during or after configuration of the Ethernet interface NET 5.
		FALSE – The NET 5 Ethernet interface configuration is correct.
NET 6 CONFIG. ERROR	bNET6ConfigError	TRUE – An error occurred during or after configuration of the Ethernet interface NET 6.
		FALSE – The NET 6 Ethernet interface configuration is correct.
INVALID DATE/TIME	bInvalidDateTime	TRUE – The date or time is invalid.
		FALSE – The date and time are valid.
RUNTIME RESET	bRuntimeReset	TRUE – The RTS (Runtime System) has been restarted at least once. This diagnostic is cleared only at system restart.
		FALSE – The RTS (Runtime System) is operating normally.
RETENTIVITY LOST	bRetentivityLost	TRUE – An error occurred while saving retentive data.
		FALSE – Valid retentive data was found during initialization.
OTD SWITCH ERRROR	bOTDSwitchError	TRUE – A key remained stuck for more than 20 seconds at least once while the CPU was powered. This diagnostic is cleared only at system restart.
		FALSE – No key has remained stuck while the CPU was powered.
ABSENT RACKS	bAbsentRacks	TRUE – One or more declared racks are missing.
		FALSE – All declared racks are present.
DUPLICATE RACKS	bDuplicatedRacks	TRUE – There is a rack with a duplicated identification number.
		FALSE – No racks have duplicated identification numbers.
INVALID RACKS	bInvalidRacks	TRUE – There is a rack with an invalid identification number.
		FALSE – No racks have invalid identification numbers.
UNDECLARED RACKS	bNonDeclaredRacks	TRUE – There is a rack with an undeclared identification number.
		FALSE – No racks have undeclared identification numbers.
DUPLICATED SLOTS	bDuplicatedSlots	TRUE – There is a duplicated slot address.
		FALSE – No duplicated slot addresses were found.

Table 168: CPU Summary Diagnostics

Notes:

Incompatible configuration: The incompatible configuration diagnostic is generated when one or more modules on the bus do not match what was declared in the project, i.e., in cases of missing or different modules. Modules physically inserted on the bus but not declared in the project are not considered.

Swapped modules: If exactly two modules are swapped with each other on the respective bus, the swapped-modules diagnostic can be identified. Otherwise, the condition is treated as an "Incompatible Configuration."

Modules with fatal error: If the Modules with Fatal Error diagnostic is active, identify the faulty module on the bus and contact Altus Technical Support, since this indicates a hardware failure.

Modules with parameterization error: If the Module Parameterization Error diagnostic is active, verify that the bus modules are configured correctly and that the firmware and Mastertool software versions are appropriate. If the problem occurs when inserting a module into the bus, check whether that module supports hot-swap.

Bus error: Considered a fatal error that interrupts access to the bus modules. If the Bus Error diagnostic is active, there may be a hardware problem on the bus communication lines; contact Altus Technical Support.

Hardware failure: If the Hardware Failure diagnostic is active, send the CPU to Altus Technical Support, since this indicates problems with the RTC, auxiliary processor, or other hardware resources.

Software exception: If a software exception is diagnosed, review your application to ensure it is not accessing memory improperly. If the problem persists, consult Altus Support. Software exception codes are listed after the CPU detailed diagnostics table.

Diagnostic Message: Diagnostic messages can be viewed on the CPU graphical display using the OTD key or via the web interface on the CPU diagnostics page.

7.1.4.2. Detailed Diagnostics

The tables below contain Nexto Series' CPUs detailed diagnostics. It is important to have in mind the observations below before consulting them:

- **Visualization of the Diagnostics Structures:** The Diagnostics Structures added to the Project can be seen at the item "*Library Manager*" of Mastertool tree view. There, it is possible to see all data types defined in the structure.
- **Counters:** All CPU diagnostics counters return to zero when their limit value is exceeded.
- *DG_Module*, where the word Module must be replaced by the product being used.

7.1.4.2.1. Target Detailed Diagnostics Group

DG_Module.tDetailed.Target.*	Size	Description
dwCPUModel	DWORD	CPU model.
abyCPUVersion	BYTE ARRAY(4)	Firmware version.
abyBootloaderVersion	BYTE ARRAY(4)	Bootloader version.
abyAuxprocVersion	BYTE ARRAY(4)	Auxiliary processor version.

Table 169: Target Detailed Diagnostics Group Description

7.1.4.2.2. Hardware Detailed Diagnostics Group

DG_Module.tDetailed.Hardware.*	Size	Description
bAuxprocFailure	BIT	Failure in the communication between the auxiliary processor and the principal processor.
bRTCFailure	BIT	The main processor is not enabled to communicate with the RTC (CPU's clock).
bThermometerFailure	BIT	Failure in the communication between the thermometer and the main processor.
bLCDFailure	BIT	Failure in the communication between the LCD screen and the main processor.

Table 170: Hardware Detailed Diagnostics Group Description

7.1.4.2.3. Exception Detailed Diagnostics Group

DG_Module.tDetailed.Exception.*	Size	Description
wExceptionCode	WORD	Exception code generated by the RTS. See Table 172.
byProcessorLoad	BYTE	Level, in percentage (%), of charge in the processor.

Table 171: Exception Detailed Diagnostics Group Description

Note:
Exception Code: the code of the exception generated by the RTS (Runtime System) can be consulted below:

Code	Description	Code	Description
0x0000	There is no exception code.	**0x0051**	Access violation.
0x0010	Watchdog time of the IEC task expired (Software Watchdog).	**0x0052**	Privileged instruction.
0x0012	I/O configuration error.	**0x0053**	Page failure.
0x0013	Checksum error after the program download.	**0x0054**	Stack overflow.
0x0014	Fieldbus error.	**0x0055**	Invalid disposition.
0x0015	I/O updating error.	**0x0056**	Invalid maneuver.
0x0016	Cycle time (execution) exceeded.	**0x0057**	Protected page.
0x0017	Program online updating too long.	**0x0058**	Double failure.
0x0018	External references not resolved.	**0x0059**	Invalid OpCode.
0x0019	Download rejected.	**0x0100**	Data type misalignment.
0x001A	Project not loaded, as the retentive variables cannot be reallocated.	**0x0101**	Arrays limit exceeded.
0x001B	Project not loaded and deleted.	**0x0102**	Division by zero.
0x001C	Out of memory stack.	**0x0103**	Overflow.
0x001D	Retentive memory is corrupted and cannot be mapped.	**0x0104**	Cannot be continued.
0x001E	Project can be loaded but causes a drop later on.	**0x0105**	Watchdog in the processor load of all IEC task detected.
0x0021	Target of startup application does not match to the current target.	**0x0150**	FPU: Not specified error.
0x0022	Scheduled tasks error.	**0x0151**	FPU: Operand is not normal.
		0x0152	FPU: Division by zero.
0x0023	Downloaded file Checksum with error.	**0x0153**	FPU: Inexact result.
0x0024	Retentive identity is not correspondent to the current identity of the boot project program	**0x0154**	FPU: Invalid operation.
0x0025	IEC task configuration failure.	**0x0155**	FPU: Overflow.
0x0026	Application working with wrong target.	**0x0156**	FPU: Stack verification.
0x0050	Illegal instruction.	**0x0157**	FPU: Underflow.

Table 172: RTS Exception codes

7.1.4.2.4. RetainInfo Detailed Diagnostics Group

DG_Module.tDetailed.RetainInfo.*	Size	Descrição
byCPUInitStatus	BYTE	CPU Startup Status: 01: Not used 02: Warm Start 03: Cold Start Note: These variables are reset in every powerup.
wCPUColdStartCounter	WORD	Increments when the CPU starts with loss of retentivity. (0 to 65535)
wCPUWarmStartCounter	WORD	Increments when the CPU starts normally with valid retain data. (0 to 65535)

DG_Module.tDetailed.RetainInfo.*	Size	Descrição
wRTSResetCounter	WORD	Counter of resets performed by RTS (Runtime System). (0 to 65535).

Table 173: RetainInfo Group Detailed Diagnostics

7.1.4.2.5. Reset Detailed Diagnostics Group

DG_Module.tDetailed.Reset.*	Size	Description
bBrownOutReset	BIT	The CPU was restarted due a failure in the power supply in the last startup.
bWatchdogReset	BIT	The CPU was restarted due the active watchdog in the last startup.

Table 174: 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.

7.1.4.2.6. Thermometer Detailed Diagnostics Group

DG_Module.tDetailed.Thermometer.*	Size	Description
bOverTemperatureAlarm	BIT	Alarm generated due internal temperature at 85 °C or above it.
bUnderTemperatureAlarm	BIT	Alarm generated due internal temperature at 0 °C or under it.
diTemperature	DINT	Temperature read in the internal sensor of the CPU.

Table 175: 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.

7.1.4.2.7. Serial Detailed Diagnostics Group

DG_Module.tDetailed.Serial.COM1.*	Size	Description
byProtocol	BYTE	Protocol selected in the COM 1: 00: Without protocol 01: MODBUS RTU Master 02: MODBUS RTU Slave 03: Other protocol
dwRXBytes	DWORD	Counter of characters received from COM 1 (0 to 4294967295).
dwTXBytes	DWORD	Counter of characters transmitted from COM 1 (0 to 4294967295).
wRXPendingBytes	WORD	Number of characters left in the reading buffer in COM 1 (0 to 4096).
wTXPendingBytes	WORD	Number of characters left in the transmission buffer in COM 1 (0 to 1023).
wBreakErrorCounter	WORD	The transmitter is holding the data line at zero for too long, according to the databit configured.

DG_Module.tDetailed.Serial.COM1.*	Size	Description
wParityErrorCounter	WORD	The received frame has the mismatched parity bit.
wFrameErrorCounter	WORD	The received frame has the wrong start point, usually caused by a noise or baud rate mismatch.
wRXOverrunCounter	WORD	When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device).

Table 176: Serial COM 1 Detailed Diagnostics Group Description

Note:

Parity Error Counter: When the serial COM 1 is configured Without Parity, this error counter won't be incremented when it receives a message with a different parity. In this case, a frame error will be indicated.

7.1.4.2.8. Ethernet Detailed Diagnostics Group

DG_Module.tDetailed.Ethernet.*	Size	Description
NET[x].bLinkDown	BIT	Indicates link state on NET[x].
NET[x].byOperatingMode	BYTE	Indicates the operating mode of the interface NET[x].
NET[x].byOperatingState	BYTE	Indicates the operating state of the interface NET[x].
NET[x].szIP	STRING (15)	NET[x] IP Address.
NET[x].szMask	STRING (15)	NET[x] Subnet Mask.
NET[x].szGateway	STRING (15)	NET[x] Gateway Address.
NET[x].szMAC	STRING (17)	NET[x] MAC Address.
NET[x].abyIP	BYTE ARRAY(4)	NET[x] IP Address.
NET[x].abyMask	BYTE ARRAY(4)	NET[x] Subnet Mask.
NET[x].abyGateway	BYTE ARRAY(4)	NET[x] Gateway Address.
NET[x].abyMAC	BYTE ARRAY(6)	NET[x] MAC Address.
NET[x].NICTeaming.bStandbyState	BIT	Indicates the interface is in standby state.
NET[x].NICTeaming.bActiveState	BIT	Indicates the interface is in active state.
NET[x].NICTeaming.bLinkDown	BIT	Indicates the network link is down.
NET[x].NICTeaming.InterMsgTimeout	BIT	Indicates no communication with the other interface.
NET[x].NICTeaming.GeneralRxMsgTimeout	BIT	Indicates no network communication activity.
NET[x].dwTransmittedBytes	DWORD	Counter of bytes sent via the NET[x] port (0 to 4294967295).
NET[x].dwTransmittedPackets	DWORD	Counter of packets sent via the NET[x] port (0 to 4294967295).
NET[x].dwTransmittedDropErrors	DWORD	Connection loss counter on transmission over the NET[x] port (0 to 4294967295).
NET[x].dwTransmittedCollisionErrors	DWORD	Transmission collision error counter over NET[x] port (0 to 4294967295).
NET[x].dwReceivedBytes	DWORD	Counter of bytes received via the NET[x] port (0 to 4294967295).
NET[x].dwReceivedPackets	DWORD	Counter of packets received via NET[x] port (0 to 4294967295).
NET[x].dwReceivedDropErrors	DWORD	Connection loss counter on reception via the NET[x] port (0 to 4294967295).
NET[x].dwReceivedFrameErrors	DWORD	Frame error counter on reception over NET[x] port (0 to 4294967295).

Table 177: Ethernet Group Detailed Diagnostics NET[x]

Note:

[x] is the ethernet interface number of the CPU, where for example, NET 1 represents the value 1.

DG_Module.tDetailed.Ethernet.RSTP.*	Size	Description
RSTP[x].bProtocolEnabled	BIT	Indicates if the RSTP protocol is enabled.
RSTP[x].bBridgeIsRoot	BIT	Indicates if the Bridge is root.
RSTP[x].eProtocol	BYTE	Indicates the protocol version.
RSTP[x].sBridgeId	STRING (22)	Local bridge ID (priority and MAC).

DG_Module.tDetailed.Ethernet.RSTP.*	Size	Description
RSTP[x].sRootId	STRING (22)	Root bridge ID (priority and MAC).
RSTP[x].eRootPort	BYTE	Indicates which is the root port.
RSTP[x].dwRootPathCost	DWORD	Path Cost to the bridge root.
RSTP[x].byBridgeHelloTime	BYTE	Hello Time configured.
RSTP[x].byRootHelloTime	BYTE	Hello Time learned.
RSTP[x].byBridgeMaxAge	BYTE	Max Age configured.
RSTP[x].byRootMaxAge	BYTE	Max Age learned.
RSTP[x].byBridgeForwDelay	BYTE	Forward Delay configured.
RSTP[x].byRootForwDelay	BYTE	Forward Delay learned.
RSTP[x].dwTxHoldCount	DWORD	Limit of BPDUs transmitted per cycle.
RSTP[x].dwTimeSinceTC	DWORD	Time, in seconds, since the last topology change.
RSTP[x].dwTCCount	DWORD	Number of times the topology has been changed.
RSTP[x].aPort[y].eName	BYTE	Port name.
RSTP[x].aPort[y].eRole	BYTE	Port function.
RSTP[x].aPort[y].eState	BYTE	Port state.
RSTP[x].aPort[y].wIdentity	WORD	Port ID.
RSTP[x].aPort[y].dwPathCost	DWORD	Port cost.
RSTP[x].aPort[y].bOperEdge	BIT	Indicates if the port is working as Edge.
RSTP[x].aPort[y].bOperP2P	BIT	Indicates if the port is working as Point-to-Point.
RSTP[x].aPort[y].dwNumTxBPDU	DWORD	Number of BPDUs frames transmitted on the port.
RSTP[x].aPort[y].dwNumRxBPDU	DWORD	Number of frames BPDUs received on the port.

Table 178: RSTP Detailed Diagnostics Group Description

Note:

[x]: is the bridge number.

[y]: is the port number within the bridge.

DG_Module.tDetailed.Ethernet.MRP.*	Size	Description
MRP[x].eConfiguredRingRole	BYTE	Indicates the device configuration (Manager or Client)
MRP[x].eCurrentRingRole	BYTE	Indicates the current device configuration
MRP[x].eRingState	BYTE	Indicates the ring status
MRP[x].ePrimaryPort	BYTE	Indicates the main MRP NET
MRP[x].eSecondaryPort	BYTE	Indicates secondary MRP NET
MRP[x].eManagerDiagnostics	BYTE	Indicates Manager diagnostic type

Table 179: MRP Detailed Diagnostics Group Description

Note:

[x]: is the MRP number.

7.1.4.2.9. UserFiles Detailed Diagnostics Group

DG_Module.tDetailed.UserFiles.*	Size	Description
byMounted	BYTE	Indicates if the memory used for recording user files is able to receive data.
dwFreeSpacekB	DWORD	Free memory space for user files in Kbytes.
dwTotalSizekB	DWORD	Storage capacity of the memory of user files in Kbytes.

Table 180: 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.

7.1.4.2.10. UserLogs Detailed Diagnostics Group

DG_Module.tDetailed.UserLogs.*	Size	Description
byMounted	BYTE	Status of the memory where user logs are inserted.
wFreeSpacekB	WORD	User log memory free space in Kbytes.
wTotalSizekB	WORD	User logs memory storage capacity in Kbytes.

Table 181: UserLogs Detailed Diagnostics Group Description

7.1.4.2.11. MemoryCard Detailed Diagnostics Group

DG_Module.tDetailed.MemoryCard.*	Size	Description
byMounted	BYTE	Status of the Memory Card: 00: Memory card not mounted 01: Memory card inserted and mounted
bEnable	BIT	Memory Card enabled.
dwFreeSpacekB	DWORD	Free space in the Memory Card in Kbytes.
dwTotalSizekB	DWORD	Storage capacity of the Memory Card in Kbytes.

Table 182: MemoryCard Detailed Diagnostics Group Description

7.1.4.2.12. WHSB Detailed Diagnostics Group

DG_Module.tDetailed.WHSB.*	Size	Description
byHotSwapAndStartupStatus	BYTE	Informs the abnormal situation in the bus which caused the application stop for each mode of hot swapping. See Table 184 for more information.
adwRackIOErrorStatus	DWORD ARRAY (32)	Identification of errors in I/O modules, individually. For more information about this diagnostic, see the notes below.
adwModulePresenceStatus	DWORD ARRAY (32)	Status of presence of declared I/O modules in buses, individually. For more information about this diagnostic, see the notes below.
byWHSBBusErrors	BYTE	Counter of failures in the WHSB bus. This counter is restarted in the energization (0 to 255).

Table 183: WHSB Detailed Diagnostics Group Description

Notes:

Bus modules error diagnostic: Each DWORD from this diagnostic array represents a rack, whose positions are represented by the bits of these DWORDS. So, Bit-0 of the DWORD-0 is equivalent to position zero of the rack with address zero. Each one of these Bits is the result of an OR logic operation between the Incompatible Configuration (bConfigMismatch), absent modules (bAbsentModules), swapped modules (bSwappedModules), module with fatal error (bModuleFatalError) diagnostics and the operational state of the module in a certain position.

Module presence status: Each DWORD from this diagnostic array represents a rack, whose positions are represented by the bits of these DWORDS. So, Bit-0 from DWORD-0 is equivalent to position zero of the rack with address zero. So, if a module is present, this bit will be true. It's important to notice that this diagnostic is valid for all modules, except power supplies, CPUs and non-declared modules, e.g. those that are not in the rack on the respective position (bit remains in false).

Situations in which the Application Stops: The codes for the possible situations in which the application stops can be consulted below:

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.

Code	Enumerable	Description
02	ABSENT_MODULES_HOT_SWAP_ DISABLED	Application in Stop Mode due to Absent Modules diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
03	CFG_MISMATCH_HOT_SWAP_ DISABLED	Application in Stop Mode due to Configuration Mismatch diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
04	ABSENT_MODULES_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Absent Modules diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
05	CFG_MISMATCH_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Incompatible Configuration diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
06	APPL_STOP_ALLOWED_TO_RUN	Application in Stop Mode and all consistencies executed successfully. The application can be set to Run Mode.
07	APPL_STOP_MODULES_NOT_READY	Application in Stop Mode and all consistencies executed successfully, but the I/O modules are not able to start the system. It is not possible to set the application to Run Mode.
08	APPL_STOP_MODULES_GETTING_ READY_TO_RUN	Application in Stop Mode and all consistencies executed successfully. The I/O modules are being prepared to start the system. It is not possible to set the application to Run Mode.
09	NORMAL_OPERATING_STATE	Application in Run Mode.
10	MODULE_CONSISTENCY_OK	Internal usage.
11	APPL_STOP_DUE_TO_EXCEPTION	Application in Stop Mode due to an exception in the CPU.
12	DUPLICATED_SLOT_HOT_SWAP_ DISABLED	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
13	DUPLICATED_SLOT_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
14	DUPLICATED_SLOT_HOT_SWAP_ ENABLED	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Enabled, without startup consistency"*.
15	NON_DECLARED_MODULE_HOT_ SWAP_STARTUP_CONSISTENCY	Application in Stop Mode due to Non Declared Modules diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"*.
16	NON_DECLARED_MODULE_HOT_ SWAP_DISABLED	Application in Stop Mode due to Non Declared Modules diagnostic being set when the Hot Swap Mode is *"Disabled"*.

Table 184: Codes of the Situations in which the Application Stops

7.1.4.2.13. Application Detailed Diagnostics Group

DG_Module.tDetailed.Application.*	Size	Description
byCPUState	BYTE	Informs the operation state of the CPU: 01: All user applications are in Run Mode 03: All user applications is in Stop Mode
bForcedIOs	BIT	There is one or more forced I/O points.
bNetDefinedByWeb	BIT	The IP address is set by the System Web Page.

Table 185: Application Detailed Diagnostics Group Description

7.1.4.2.14. SNTP Detailed Diagnostics Group

DG_Module.tDetailed.SNTP.*	Size	Description
bServiceEnabled	BIT	SNTP service enabled.

DG_Module.tDetailed.SNTP.*	Size	Description
byActiveTimeServer	BYTE	Indicates which server is active: 00: No active server. 01: Primary server active. 02: Secondary server active.
wPrimaryServerDownCount	WORD	Count of times the primary server was unavailable (0 to 65535).
wSecondaryServerDownCount	WORD	Count of times the secondary server was unavailable (0 to 65535).
dwRTCTimeUpdatedCount	DWORD	Count of times the RTC was updated by the SNTP service (0 to 4294967295).
byLastUpdateSuccessful	BYTE	Indicates status of last update: 00: Not updated. 01: Last update failed. 02: Last update was successful.
byLastUpdateTimeServer	BYTE	Indicates which server was used in the last update: 00: No updates. 01: Primary server. 02: Secondary server.
sLastUpdateTime.byDayOfMonth	BYTE	Day of last RTC update.
sLastUpdateTime.byMonth	BYTE	Month of last RTC update.
sLastUpdateTime.wYear	WORD	Year of last update of RTC.
sLastUpdateTime.byHours	BYTE	Hour of last RTC update.
sLastUpdateTime.byMinutes	BYTE	Minute of last RTC update.
sLastUpdateTime.bySeconds	BYTE	Second of last RTC update.
sLastUpdateTime.wMilliseconds	WORD	Millisecond of last RTC update.

Table 186: SNTP Group Detailed Diagnostics

7.1.4.2.15. Rack Detailed Diagnostics Group

DG_Module.tDetailed.Rack.*	Size	Description
dwAbsentRacks	DWORD	Each bit represents an identification number of a rack, if any bit is TRUE, it means that the rack, with that identification number, is absent.
dwDuplicatedRacks	DWORD	Each bit represents an identification number of a rack, if any bit is TRUE, it means that more than one rack is configured with the same identification number.
dwNonDeclaredRacks	DWORD	Each bit represents a rack identification number, if any bit is TRUE, it means that there is a rack configured with an identification number that is not declared in the project.

Table 187: Rack Group Detailed Diagnostics

7.1.4.2.16. ApplicationInfo Detailed Diagnostics Group

DG_Module.tDetailed.ApplicationInfo.*	Size	Description
dwApplicationCRC	DWORD	32-bit CRC of the application. When the application is modified and sent to the CPU, a new CRC is calculated.

Table 188: ApplicationInfo Detailed Diagnostics Group Description

7.1.4.2.17. Firewall Detailed Diagnostics Group

DG_Module.tDetailed.Firewall.*	Size	Description
bServiceEnabled	BIT	The Firewall service is enabled.

DG_Module.tDetailed.Firewall.*	Size	Description
sLatsModificationDateUTC	NextoStandard. EXTENDED_DATE_AND_TIME	Date and time of the last configuration change in UTC.

Table 189: Firewall Detailed Diagnostics Group Description

7.1.4.2.18. OpenVPN Detailed Diagnostics Group

DG_Module.tDetailed.OpenVPN.*	Size	Description
byOperationMode	BYTE	VPN operating mode: SERVER(0): Device operating as a server. CLIENT(1): Device operating as a client.
bServiceEnabled	BIT	VPN service enabled
bServiceRunning	BIT	VPN service running.
byConnectionState	BYTE	VPN connection status: DISCONNECTED(0): Device disconnected. CONNECTING(1): Device connecting. CONNECTED(2): Device connected.
uliConnectionTime	ULINT	Current connection time in seconds.
sIPAddress	STRING(15)	VPN IP address.
dwConnectedClients	DWORD	Number of connected clients.
dwTransmittedBytes	DWORD	Number of bytes transmitted through the VPN.
dwReceivedBytes	DWORD	Number of bytes received through the VPN.
CACertificate.CertificateName	STRING(64)	CA certificate name.
CACertificate.CommonName	STRING(64)	CA certificate common name.
CACertificate.sStartDate	EXTENDED_DATE_AND_TIME	CA certificate start date and time.
CACertificate.sExpirationDate	EXTENDED_DATE_AND_TIME	CA certificate expiration date and time.
CACertificate.dwValidDaysLeft	DWORD	Days until CA certificate expiration.
DeviceCertificate.CertificateName	STRING(64)	Device certificate name.
DeviceCertificate.CommonName	STRING(64)	Device certificate common name.
DeviceCertificate.sStartDate	EXTENDED_DATE_AND_TIME	Device certificate start date and time.
DeviceCertificate.sExpirationDate	EXTENDED_DATE_AND_TIME	Device certificate expiration date and time.
DeviceCertificate.dwValidDaysLeft	DWORD	Days until device certificate expiration.
sDeviceKeyName	STRING(64)	Device private key name.

Table 190: Detailed Diagnostics OpenVPN Group

7.1.4.2.19. FTP Detailed Diagnostics Group

DG_Module.tDetailed.FTP.*	Size	Description
bServiceEnabled	BIT	FTP service enabled.
dwConnectedClients	DWORD	Number of connected clients.
sLastUpdateTime.byDayOfMonth	BYTE	Day of the last FTP update.
sLastUpdateTime.byMonth	BYTE	Month of the last FTP update.
sLastUpdateTime.wYear	WORD	Year of the last FTP update.
sLastUpdateTime.byHours	BYTE	Hour of the last FTP update.
sLastUpdateTime.byMinutes	BYTE	Minute of the last FTP update.
sLastUpdateTime.bySeconds	BYTE	Second of the last FTP update.
sLastUpdateTime.wMilliseconds	WORD	Millisecond of the last FTP update.

Table 191: Detailed Diagnostics FTP Group

7.1.5. Diagnostics via Function Blocks

The function blocks allow the visualization of some parameters which cannot be accessed otherwise. The function regarding advanced diagnostics is in the *NextoStandard* library and is described below.

7.1.5.1. GetTaskInfo

This function returns the task information of a specific application.



Figure 244: 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 192: GetTaskInfo Input Parameters

The data returned by the function, through the pointer informed in the input parameters are described on table below.

Returned Parameters	Size	Description
dwCurScanTime	DWORD	Task cycle time (execution) with 1 μs resolution.
dwMinScanTime	DWORD	Task cycle minimum time with 1 μs resolution.
dwMaxScanTime	DWORD	Task cycle maximum time 1 μs resolution.
dwAvgScanTime	DWORD	Task cycle average time with 1 μs resolution.
dwLimitMaxScan	DWORD	Task cycle maximum time before watchdog occurrence.
dwIECCycleCount	DWORD	IEC cycle counter.

Table 193: GetTaskInfo Output Parameters

Possible ERRORCODE:

- NoError: success execution;
- TaskNotPresent: the desired task does not exist.

Example of utilization in ST language:

```
PROGRAM UserPrg
VAR
sAppName : STRING;
psAppName : POINTER TO STRING;
sTaskName : STRING;
psTaskName : POINTER TO STRING;
pstTaskInfo : POINTER TO NextoStandard.stTaskInfo;
TaskInfo :NextoStandard. stTaskInfo;
Info : NextoStandard.ERRORCODE;
```

```
END_VAR
//INPUTS:
sAppName := 'Application';    //Variable receives the application name.
psAppName := ADR(sAppName); //Pointer with application name.
sTaskName := 'MainTask';    //Variable receives task name.
psTaskName := ADR(sTaskName); //Pointer with task name.
pstTaskInfo := ADR(TaskInfo); //Pointer that receives task info.
//FUNCTION:
//Function call.
Info := GetTaskInfo (psAppName, psTaskName, pstTaskInfo);
//Variable Info receives possible function errors.
```

7.2. Graphic Display

The graphic display available in this product has an important tool for the process control, as through it is possible to recognize possible error conditions, active components or diagnostics presence. Besides, all diagnostics including the I/O modules are presented to the user through the graphic display. For further information regarding the diagnostic key utilization and its visualization see One Touch Diag section.

On figure below, it is possible to observe the available characters in this product graphic display and, next, its respective meanings.

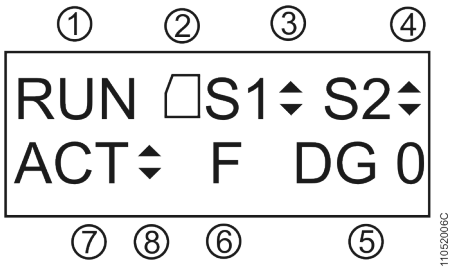


Figure 245: CPU Status Screen

Legend:

- 1. Indication of the CPU operating state. If the CPU application is running, the state will be RUN. If the CPU application is stopped, the state will be STOP, and when it is stopped at application debug breakpoints, the state will be BRKP. If the CPU is in exception, the state will be EXCP. For more details, see section CPU Operating States.
- 2. Indication of the presence of the Memory Card. For more details on card installation, see section Memory Card Installation.
- 3. Indication of traffic on COM 1. The upward arrow (▲) indicates data transmission and the downward arrow (▼) indicates data reception. For more information about the COM 1 interface, see section Serial Interfaces.
- 4. Indication of traffic on COM 2. The upward arrow (▲) indicates data transmission and the downward arrow (▼) indicates data reception. For more information about the COM 2 interface, see section Serial Interfaces.
- 5. Indication of the number of active diagnostics in the CPU. If the displayed number is different from 0 (zero), there are active diagnostics in the CPU. For more details on viewing them on the CPU graphical display using the diagnostics key, see section One Touch Diag.
- 6. Indication of forced variables in the CPU. If the character F is displayed on the graphical display, at least one variable is being forced by the user, either a symbolic variable or a direct representation variable mapped to a symbolic variable. For more details on forcing variables, see section Writing and Forcing Variables.
- 7. Identification of the redundancy state in the CPU (message valid only for the NX3035 in redundant mode). If the CPU is the Active CP, the ACT indication will be displayed. The other possible states are NCF (Not-Configured), STR (Starting), INA (Inactive), and SBY (Standby).
- 8. Indication that project synchronization is being executed. The upward arrow (▲) indicates transmission of project data and the downward arrow (▼) indicates reception of project data.

In addition to the characters described above, Nexto CPUs may display some messages on the graphical display corresponding to a process that is currently being executed.

The table below shows 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.
NO TAG	There is no configured tag for the CPU in the Mastertool.
NO DESC	There is no configured description for the CPU in the Mastertool.
MSG. ERROR	Indicates that there are error(s) on diagnostics message(s) of the requested module(s).
SIGNATURE MISSING	Indicates the product presented an unexpected problem. Get in contact with Altus Technical Support sector.
APP. ERROR RESTARTING	Indicates that occurred an error in the application and the Runtime is restarting the application.
APP. NOT LOADED	Indicates that the runtime will not load the application.
LOADING APP.	Indicates that the runtime will load the application.
WRONG SLOT	Indicates that the CPU is in an incorrect position in the rack.
FATAL ERROR	Indicates that there are serious problems in the CPU startup such as CPU partitions that were not properly mounted. Please, contact Altus customer support.
HW-SW MISMATCH	Indicates that the CPU hardware and software are not compatible because the product presented an unexpected problem. Please, contact Altus customer support.
UPDATING FIRMWARE	Indicates the firmware is being updated in the CPU.
RECEIVING FIRMWARE	Indicates the updating file is being transferred to the CPU.
UPDATED	Shows the firmware version updated in the CPU.
UPDATE ERROR	Indicates an error has occurred during the CPU firmware updating, caused by communication failure or configuration problems.
REBOOTING SYSTEM...	Indicates the CPU is being restarted for the updating to have effect.

Table 194: Other Messages of the Graphic Display

7.3. System Log

The *System Log* is an available feature in the Mastertool programmer. It is an important tool for process control, as it makes it possible to find events on CPU that may indicate error conditions, presence of active components or active diagnostics. Such events can be viewed in chronological order with a resolution of milliseconds, with a storage capacity of up to one thousand log entries stored in the CPU internal memory, that can't be removed.

In order to access these Logs, just go to the *Device Tree* and double-click on *Device*, then go to the *Log* tab, where hundreds of operations can be seen, such as: task max cycles, user access, online change, application download and upload, application synchronization between CPUs, firmware update between another events and actions.

In order to view the *Logs*, just need to be connected to a CPU (selected Active Path) and click on . When this button is pressed the Logs are displayed and updated instantly. When the button is not being pressed the Logs will be hold in the screen, it means, these button has two stages, one hold the logs state being updated and in the state the updating is disabled. To no longer show the Logs, press .

It is possible to filter the Logs in 4 different types: warning(s), error(s), exception(s) and information.

Another way to filter the messages displayed to the user is to select the component desired to view.

The Log tab's *Time Stamp* is shown by Mastertool after information provided by the device (CPU). Mastertool can display the Time Stamp in local time (computer's time) or UTC, if *UTC time* checkbox is marked.

ATTENTION

If the device's time or time zone parameter are incorrect, the Time Stamp shown in Mastertool also won't be correct.

For further information about the System Logs please check the Mastertool User Manual and the RTC Clock and Time Synchronization subsection of this manual.

ATTENTION

The system logs of the CPUs, starting in firmware version 1.4.0.33 (Nexto) and 1.14.36.0 (Xtorm), are reloaded in the cases of a restart of the CPU or a reboot of the Runtime System, that is, it will be possible to check the older logs when one of these situations occurs.

7.4. Not Downloading the Application at Startup

If necessary, the user can choose to not load an existing application on the CPU during its startup. Just power the CPU with the diagnostics button pressed and keep it pressed for until the message "APP. NOT LOADED" is shown in the screen. If a login attempt is made, Mastertool 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.

7.5. Power Failure

The Nexto Series Power Supply (NX8000) has a failure detection system according to the levels defined in its technical features (see Power Supply 30 W 24 Vdc Technical Characteristics - CE114200). There are two ways to diagnose a failure:

1. In case the NX8000 power supply is on with voltage lower than the required minimum limit, a power supply failure diagnostic is generated, which is recognized by the CPU and the message "POWER FAILURE" is shown on the display. When the supply is within the established limits, the CPU recognizes it and automatically is restarted with the user application. The diagnostic will still be active to show to the user that the last initialization suffered a power supply failure.
2. In case the NX8000 has a voltage drop to an inferior value than the minimum required limit and it returns to a higher value within 10 ms, the power supply failure is not recognized by the CPU and the diagnostic is not generated as the system remains intact during this time. But if the voltage drop takes longer than 10 ms, the "POWER FAILURE" message is shown on the CPU screen and the diagnostic is activated.



Figure 246: Power Supply Failure Message

The user can change the value of the variable attributed to the power supply failure to FALSE during the application execution, facilitating the verification and treatment of this diagnostic.

The POWER FAILURE diagnostic is already mapped in a specific memory region, defined as CPU Detailed Diagnostic. This way it is just to use it as global variable. The variable name is described in the detailed diagnostic list in the Diagnostics via Variables section.

7.6. Most Common Problems

If, at power on the CPU, it does not work, the following items must be verified:

- Is the room temperature within the device supported range?
- Is the rack power supply being fed with the correct voltage?
- Is the power supply module inserted on the far left in the rack (observing the rack by the front view) followed by the Nexto Series CPU?
- Are there network devices, as hubs, switches or routers, powered, interconnected, configured and working properly?
- Is the Ethernet network cable properly connected to the Nexto CPU NET 1 or NET 2 port and to the network device?
- Is the Nexto Series CPU on, in execution mode (Run) and with no diagnostics related to hardware?

If the Nexto CPU indicates the execution mode (Run) but it does not respond to the requested communications, whether through Mastertool 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.

7.7. Troubleshooting

The table below shows the symptoms of some problems with their possible causes and solutions. If the problem persists, consult the Altus Technical Support.

Symptom	Possible Cause	Solution
Does not power on	Lack of power supply or incorrectly powered.	Verify if the CPU is connected properly in the rack. Power off and take off all modules from the bus, but the power supply and the CPU. Power on the bus and verify the power supply functioning, the external and the one in the rack. Verify if the supply voltage gets to the Nexto power supply contacts and if is correctly polarized.
CPU Screen shows the message WRONG SLOT	CPU in a wrong position.	The CPU must be placed in slots 2 and 3 of rack 0. Put it in the correct slots. CPUs must be placed in slots 2 and 3 of rack 0. Put it in the correct slots.
Does not communicate	Bad contact or bad configuration.	Verify every communication cable connection. Verify the serial and Ethernet interfaces configuration in the Mastertool 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 195: Troubleshooting

7.8. Preventive Maintenance

- It must be verified, each year, if the interconnection cables are connected firmly, without dust accumulation, mainly the protection devices.
- In environments subjected to excessive contamination, the equipment must be periodically cleaned from dust, debris, etc.
- The TVS diodes used for transient protection caused by atmospheric discharges must be periodically inspected, as they might be damaged or destroyed in case the absorbed energy is above limit. In many cases, the failure may not be visual. In critical applications, is recommendable the periodic replacement of the TVS diodes, even if they do not show visual signals of failure.
- Bus tightness and cleanness every six months.
- For further information, see Nexto Series Manual - MU214600.

8. Appendixes

8.1. TLS Key and Certificate Management

This section covers the generation of security files, certificates, and keys using TLS. The certificates commented on below are signed by CA. This type of certificate considers an entity, called Certificate Authority (CA), to generate the certificates. This entity can be an official authority service or a simple computer. It is only necessary to restrict access to the CA to avoid any security breach since this entity can generate certificates for any device. The image below shows how each device interacts with the files.

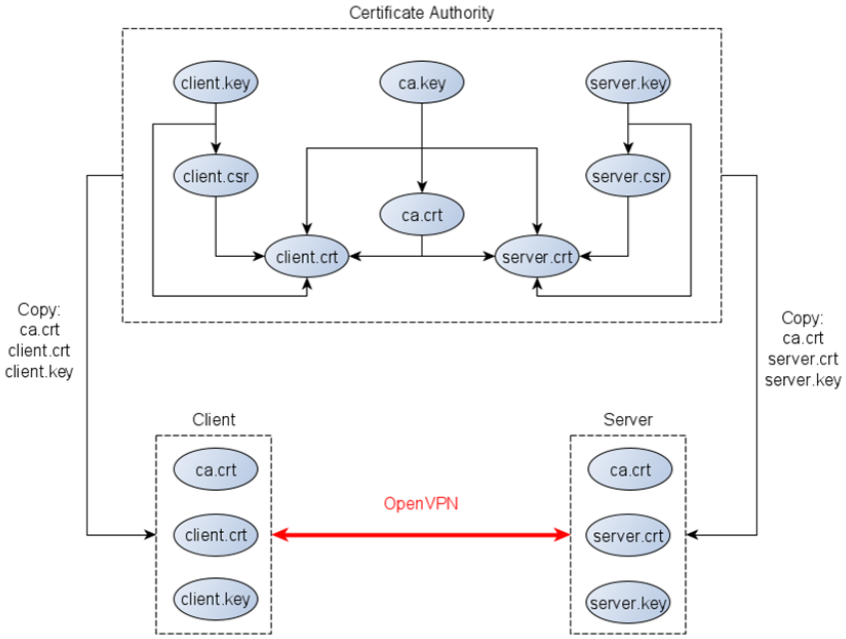


Figure 247: TLS Certificate Generation Flow

First of all, the generated files are private keys. Each device has its key file, created either by the CA entity or the device itself. The most important file is the CA private key *ca.key*, which must not leave the entity. The CA entity generates its certificate based on its private key *ca.crt*. This certificate is a public file used by the devices to validate the VPN connection. Generating certificates from the device first requires a request file (*.csr* or *.req* depending on the tool) based on the device's private key. This document presents two possible tools for generating certificate files: Easy-RSA and OpenSSL.

Make sure you have the date and time set correctly in the CA entity so that the generation of the certificates is based on a current setting.

8.1.1. Easy-RSA Certificate Generation

The OpenVPN project provides this tool to help with the certificate and keys. Easy-RSA is available for Windows and Linux. See below for step-by-step instructions to generate the files in a Windows configuration:

1- Open a Windows prompt in the Easy-RSA folder and run the following command to enter the tool shell:

```
.\EasyRSA-Start.bat
```

```
C:\Users\igor.franco\Downloads\EasyRSA-3.0.8-win64\EasyRSA-3.0.8>.\EasyRSA-Start.bat

Welcome to the EasyRSA 3 Shell for Windows.
Easy-RSA 3 is available under a GNU GPLv2 license.

Invoke './easyrsa' to call the program. Without commands, help is displayed.

EasyRSA Shell
# _
```

Figure 248: Certificate Generation using Easy-RSA (step 1)

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

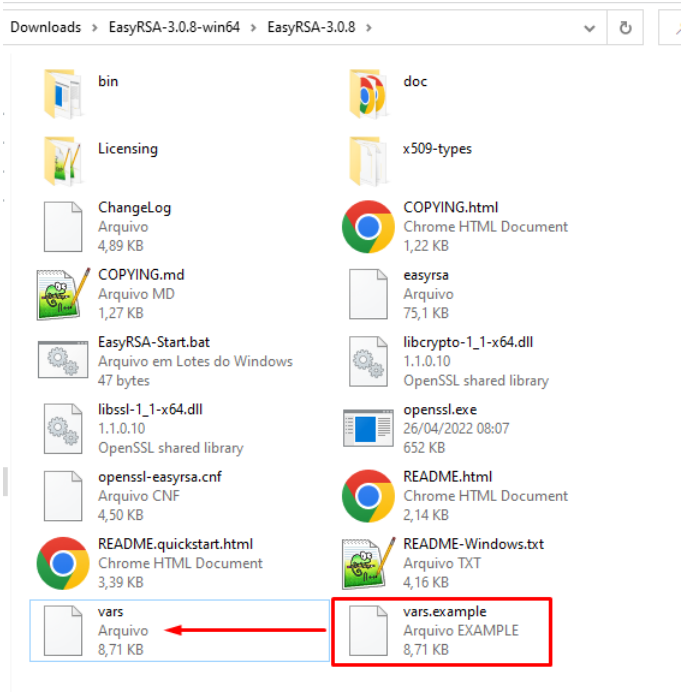


Figure 249: Certificate Generation using Easy-RSA (step 2)

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

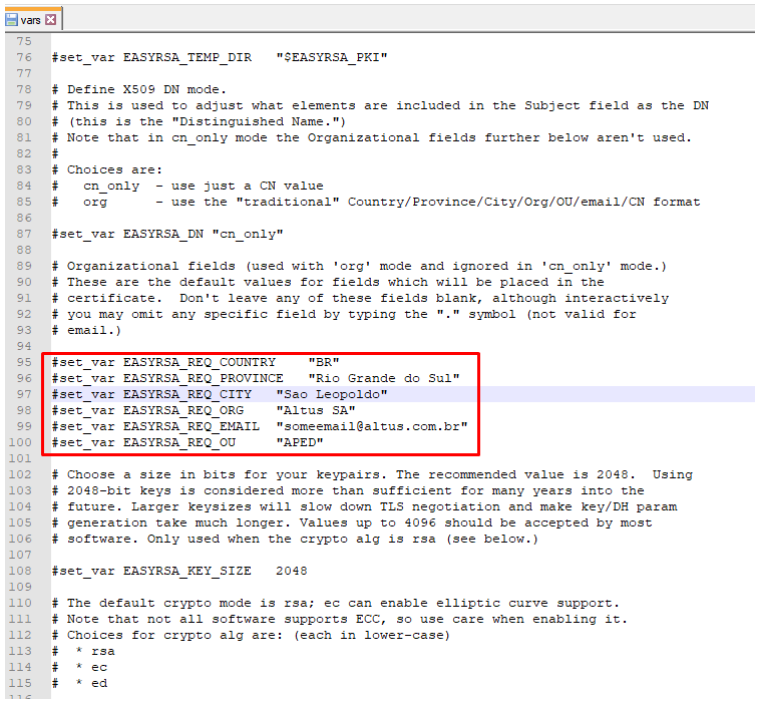


Figure 250: Certificate Generation using Easy-RSA (step 3)

4- Use the following command to prepare the configuration.

```
./easyrsa init-pki
```



Figure 251: Certificate Generation using Easy-RSA (step 4)

5- Then type the following to generate the CA certificate.

```
./easyrsa build-ca nopass
```

Remove the *nopass* argument if you want to set a password for the file. Enter the common name of the CA certificate when prompted (press enter to use the default *Easy-RSA CA* as the common name).

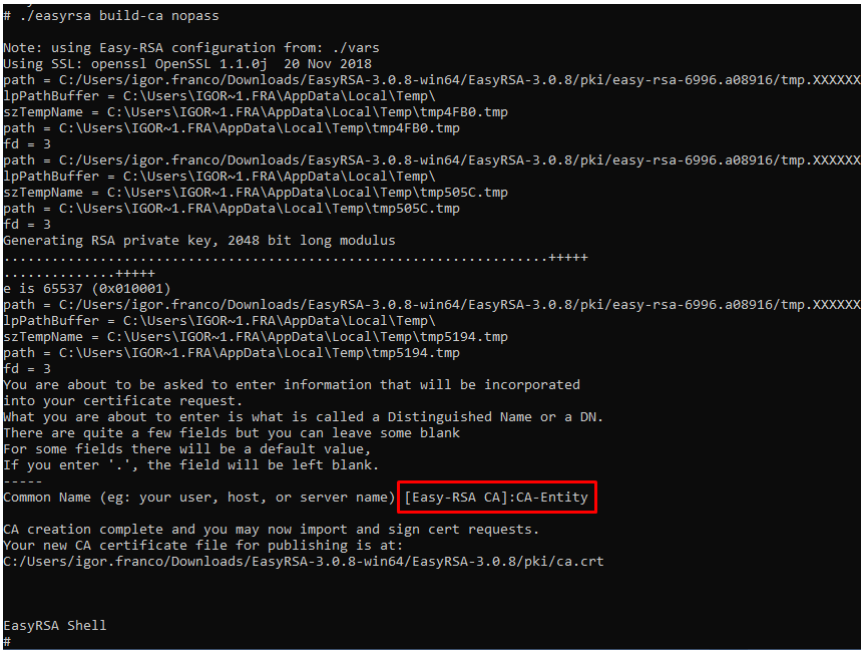


Figure 252: Certificate Generation using Easy-RSA (step 5)

6 - Generate the device key and request files using the following command (change the *DeviceName* with the desired common name):

```
./easyrsa gen-req DeviceName nopass
```

Again, remove the *nopass* argument to use a password for the certificate file. When entering the Common Name as an argument, simply press Enter when prompted (red square).

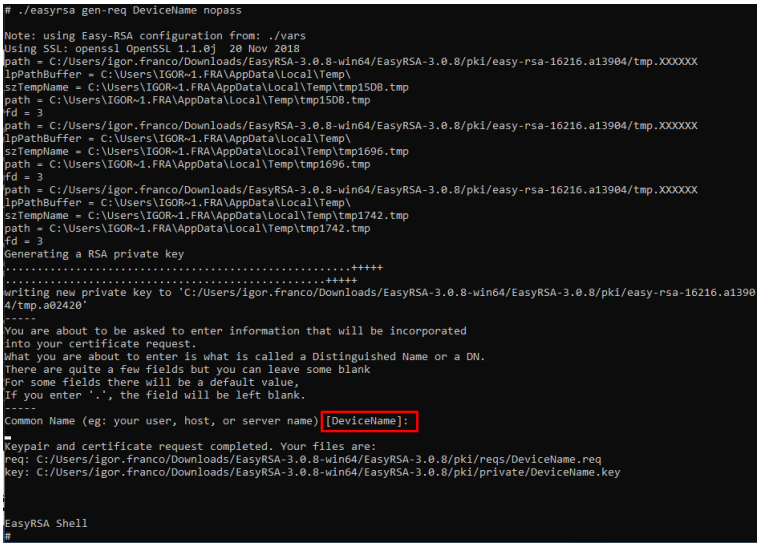


Figure 253: Certificate Generation using Easy-RSA (step 6)

7- Finally, type the following command to generate the device certificate (the *DeviceName* is the desired common name, and the *server* is the type; use *client* if you are generating for a VPN client).

```
./easyrsa sign-req server DeviceName
```

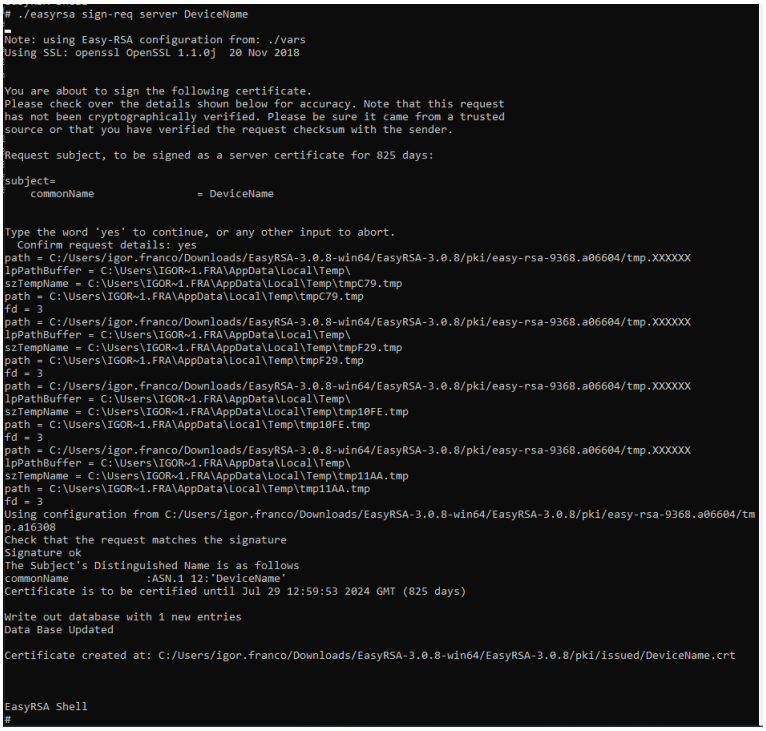


Figure 254: Certificate Generation using Easy-RSA (step 7)

8- Repeat steps 6 and 7 to generate more device certificates.

9- Find the *ca.crt* in the *pki* folder, the device private keys in the *pki/private* path, and the device certificates in the *pki/issued* directory.

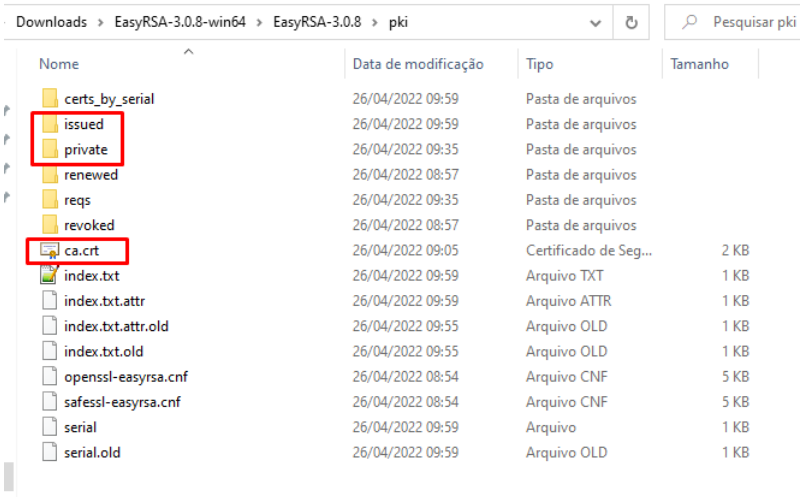


Figure 255: Certificate Generation using Easy-RSA (step 9)

8.1.2. OpenSSL Certificate Generation

OpenSSL is an open-source package with tools that help generate many files and security features. This package is native to most Linux distributions and is available for Windows. Just remember to set the OpenSSL folder in the PATH (environment variable) to allow you to use the command from anywhere via the prompt. Find below the step-by-step using this feature (all files can have any name as desired, the steps consider only an example):

1- Open a prompt in the certificate folder (where you will create the files).

2- Generate the CA private key with the following command:

```
openssl genrsa -out ca.key 4096
```

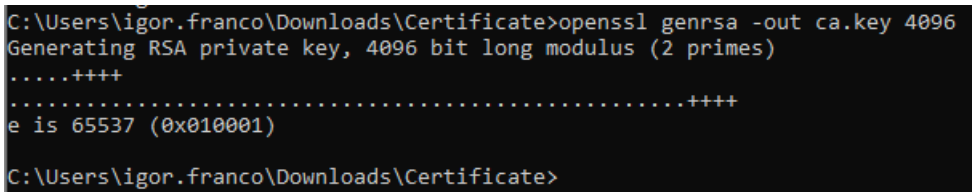


Figure 256: Certificate Generation using OpenSSL (step 2)

3- Then generate the CA certificate based on the private key, using the following command.

```
openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

The parameter *-days* represents the expiration time for the certificate. Set it as desired. In this example, the certificate is valid for one year. Fill in the values requested at the prompt as needed (press enter to use the default, which is enclosed in square brackets []). It is mandatory to define a Common Name for the certificate work.

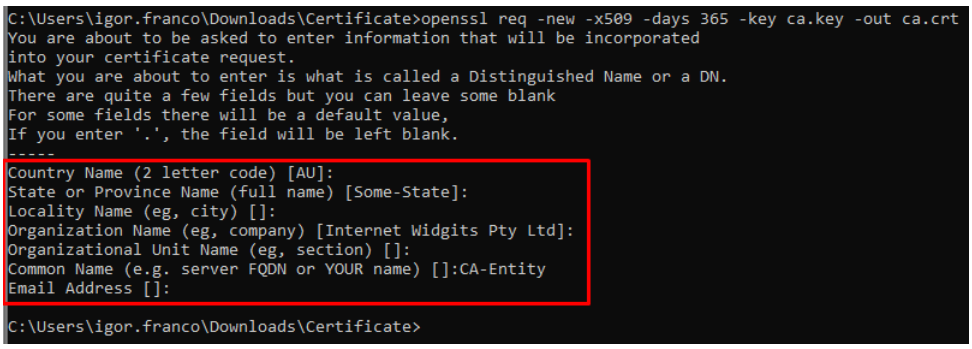


Figure 257: Certificate Generation using OpenSSL (step 3)

4- Now generate the device's private key, similar to step 2, using the following command:

```
openssl genrsa -out DeviceName.key 2048
```

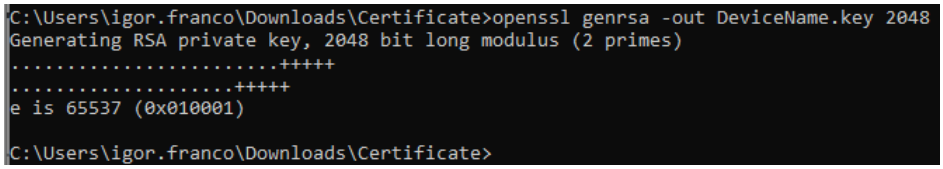


Figure 258: Certificate Generation using OpenSSL (step 4)

5- After that, generate the certificate request file based on the private key using the following command:

```
openssl req -new -key DeviceName.key -out DeviceName.csr
```

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

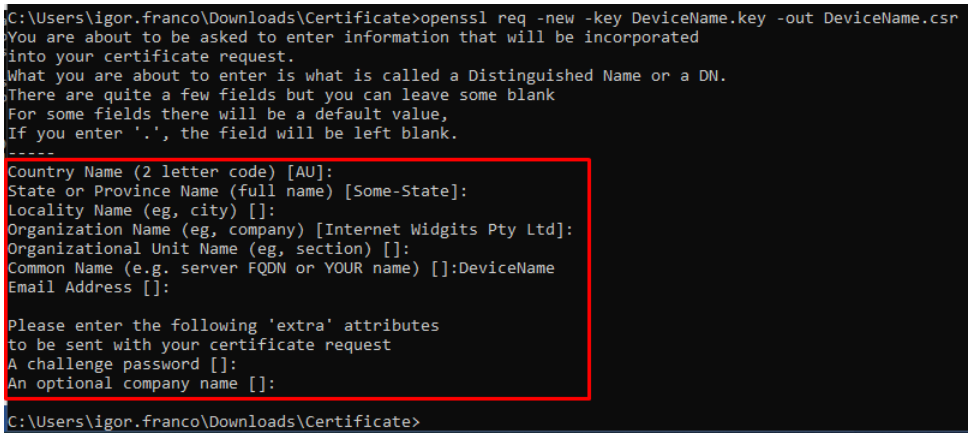


Figure 259: Certificate Generation using OpenSSL (step 5)

6- Finally, generate the device certificate using the CA private key, the CA certificate, and the device certificate request file using the following command:

```
openssl x509 -req -days 365 -in DeviceName.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out DeviceName.crt
```

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

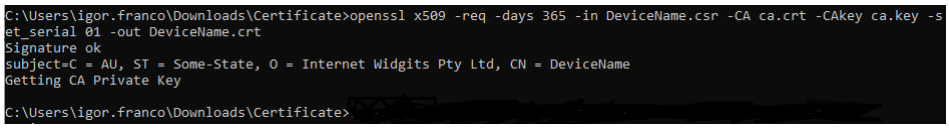


Figure 260: Certificate Generation using OpenSSL (step 6)

7- Repeat steps 4 to 6 for any new device.

8- (Optional) OpenSSL provides a tool to verify that the device certificate works with CA:

Use the following command:

```
openssl verify -purpose sslserver -CAfile ca.crt DeviceName.crt
```

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

Figure 261: Certificate Generation using OpenSSL (step 8)

8.1.3. TA Key Generation by OpenVPN

The OpenVPN project provides a tool for generating a TLS key, commonly called ta.key. This key is an extra layer of protection on OpenVPN's UDP/TCP communication ports, so the use of this key can be interpreted as an HMAC Firewall for VPN communication, requiring the existence of the parameter on both sides of the communication for it to be established.

The key generation in Windows can be done with the following command:

```
openvpn --genkey secret ta.key
```

```
C:\Program Files\OpenVPN\bin>openvpn --genkey secret C:\Users\bruno.berwanger\Desktop\Chaves\ta.key

C:\Program Files\OpenVPN\bin>
```

Figure 262: TA Key Generation in Windows example

To execute the command, we used the executable installed with the OpenVPN package. The directory used in the image above is an example and is optional. You can use only the desired file name too.

The command can be used to generate the key from Linux, but there is a minor change in the command compared to Windows. To generate the key on Linux, use the following command in the terminal:

```
openvpn --genkey --secret ta.key
```

```
developer@developer:~$ openvpn --genkey --secret ta.key
developer@developer:~$
```

Figure 263: TA Key Generation in Linux example

This parameter is not mandatory for VPN communication, but if the server uses it, all its clients must also use it, and the key for the server and the clients must be the same.

8.2. Offline Program Download Without Interrupting Process Control

The redundancy of programmable controller CPUs is primarily intended to increase the availability of the automation system. For modifications that require offline download without interrupting process control, specific redundancy features are used to avoid interrupting process control.

8.2.1. General Recommendations

Here are some best practices to follow when making programming changes to redundant systems:

- **Before making important changes:**
  1. Ensure that you can log in to any PLC without the need for a download or online change;
  2. Save the project using the commands in the "*File / Project File / Save/Send File...*" menu, generating a *projectarchive* that allows you to return to the previous backup point.

- **In case of problems during the change:**
  1. Recover the project using the "*File / Project File / Extract File...*" menu;
  2. Open the recovered project. Even if it is done on another computer or by another user, it will be possible to log in without additional changes.
- **Consistency guarantee:** before initiating any online changes, confirm that the project opened in Mastertool is identical to the one running in PLC Active (ACT).
- **Device configuration changes** (any Treeview object in Mastertool except GVL or POU) must first be downloaded to PLC Standby (SBY), with project synchronization disabled.
- **Do not mix configuration changes and redundant data:**
  - First download the new configuration to the Standby PLC with synchronization disabled;
  - Then, after the PLC returns to Standby (SBY) status, perform the switchover;
  - Only then make redundant data changes online on the new Active PLC.
- **Complex changes** (deletion/insertion of redundant variables, movement of blocks in POU, configuration changes): do this step by step, with intermediate compilations and backups, ensuring the possibility of returning to the previous version.
- **Code or variable deletions:** it is recommended to initially just comment out the lines, rather than deleting them directly.

8.2.2. PROFIBUS Changes

Among the modifications that affect a PROFIBUS network and require offline download the following are supported by the procedure that allows offline download of modifications without interrupting process control, provided that the PROFIBUS network is redundant:

- Insert a new Nexto Series or Ponto Series PROFIBUS remote;
- Insert a new I/O module into a Nexto Series or Ponto Series PROFIBUS remote;
- Remove an I/O module from a Ponto Series PROFIBUS remote;
- Change parameters on Ponto Series PROFIBUS remotes.
- Modify parameters in I/O modules of Nexto Series or Ponto Series PROFIBUS remotes.

Some design modifications require interrupting process control. These modifications are described in the paragraph Modifications which Demand Offline Download and the Interruption of the Process Control. Other design changes related to the PROFIBUS network, although supported by the procedure that allows offline modifications download without interrupting process control, will result in output discontinuity of the PROFIBUS remote where the changes are being applied. These are:

- Remove module (including "NX9999 Slot Reserved") from Nexto Series PROFIBUS remote;
- Replace module (including "NX9999 Slot Reserved") from Nexto Series PROFIBUS remote;
- Change the parameter of Nexto Series PROFIBUS remotes.

For these situations, it is recommended to perform the procedure that will require the interruption of process control, described in section Offline Download of Modifications with Process Control Interruption.

8.2.3. Initial Condition

Before starting the procedure described below, the following context must be ensured:

- One PLC is in Active (ACT) status;
- The other PLC is in Standby (SBY) status;
- Project synchronization is enabled on both PLCs (otherwise, the project will not be transferred from Active to Standby);
- The corresponding project is open in Mastertool, allowing login without the need for download or online change;
- There are no forces applied to any PLC.

8.2.4. Change Procedure

This procedure applies to configuration changes only, but also allows for code-only changes, non-redundant data-only changes, or all of these changes simultaneously, but without any changes to redundant data. In the case of simultaneous changes to redundant data and device configuration, two steps are required: first change the configuration using this method, and then change the redundant data using the online download method.

The following procedure applies to:

- Changes to configuration only;
- Changes to code or non-redundant data only;
- Simultaneous changes to these categories (provided they do not involve redundant data).

ATTENTION

If you need to change redundant data and configuration at the same time, the process must be divided into two steps:

1. Change the configuration using the method below;
2. Then change the redundant data using the specific procedure for this case.

8.2.5. Steps

The procedure can be divided into four main steps: preparation, download, control exchange, and final validation.

8.2.5.1. Step 1 – Preparation for the Change

1. Edit and compile the project in Mastertool, confirming there are no errors.
2. Set the Standby PLC (SBY) to Inoperative (INA) status.
3. Select the Standby PLC as the active path in the gateway.

8.2.5.2. Step 2 – Download to the Standby CPU

1. Disable project synchronization in PLC Standby.
2. Log in to PLC Standby (Mastertool will only allow downloads).
3. Perform the download (PLC Standby will enter STOP mode).
4. Request PLC Standby to return to RUN mode (after initialization, PLC returns to Standby mode – SBY).

8.2.5.3. Step 3 – Control Handover

1. Perform the switchover between Active PLC (ACT) and Standby PLC (SBY).
2. Enable project synchronization on the new Active PLC (ACT).
3. Wait for the program to be automatically transferred to the Standby PLC.

8.2.5.4. Step 4 – Final Validation

1. Confirm that both PLCs have returned to Active/Standby (ACT/SBY) status, restoring the initial condition.

ATTENTION

This procedure requires that the redundant data in the new project version be identical to that in the running project. Otherwise, the PLC Standby will enter the inoperative state (INA) after downloading.
In addition, it is recommended to use this procedure in the following situations:

1. When it is no longer possible to make online changes to the Active PLC;
2. When you want to test a new version of the project while keeping the previous one available on the Standby PLC, allowing for immediate recovery in case of failures.