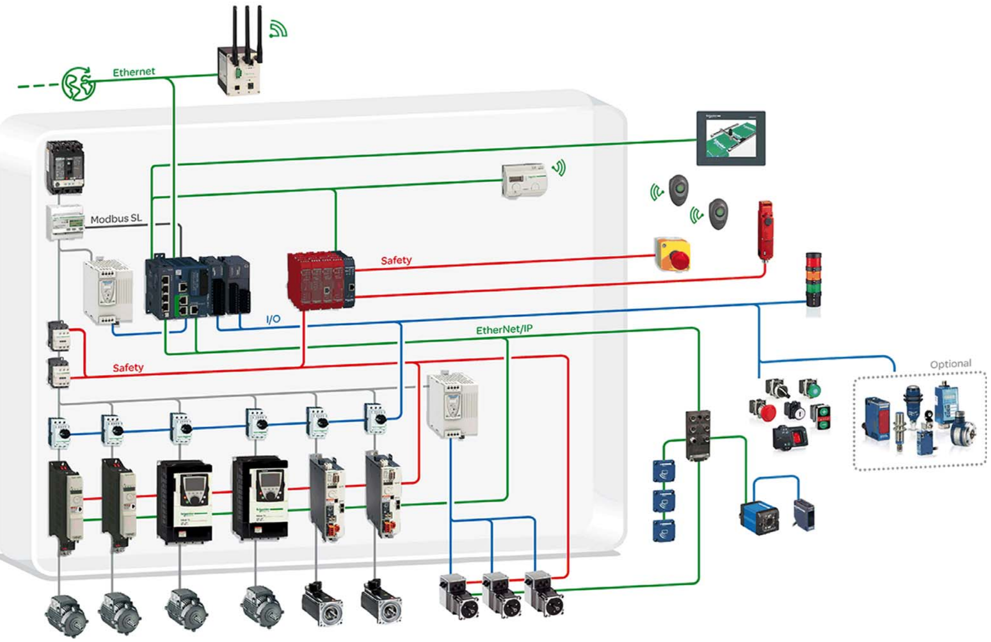
SoMachine Industrial Ethernet

User Guide

04/2017



www.schneider-electric.com

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

Table of Contents

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

⚡ The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

⚠ This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER
DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING
WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION
CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE
NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book

At a Glance

Document Scope

Use this document to:

- Plan and configure your Industrial Ethernet network.
- Install and set up your Industrial Ethernet network.
- Operate your Industrial Ethernet network.
- Perform diagnostics and maintain your Industrial Ethernet network.

NOTE: Read and understand this document and all related documents before installing, operating, or maintaining your controller.

Validity Note

This document has been updated for the release of SoMachine V4.3.

Related Documents

Title of Documentation	Reference Number
Modicon M241 Logic Controller - Programming Guide	*EIO0000001432 (ENG)* *EIO0000001433 (FRE)* *EIO0000001434 (GER)* *EIO0000001435 (SPA)* *EIO0000001436 (ITA)* *EIO0000001437 (CHS)*
Modicon M251 Logic Controller - Programming Guide	*EIO0000001462 (ENG)* *EIO0000001463 (FRE)* *EIO0000001464 (GER)* *EIO0000001465 (SPA)* *EIO0000001466 (ITA)* *EIO0000001467 (CHS)*
Modicon TM4 Expansion Modules - Programming Guide	*EIO0000001802 (ENG)* *EIO0000001803 (FRE)* *EIO0000001804 (GER)* *EIO0000001805 (SPA)* *EIO0000001806 (ITA)* *EIO0000001807 (CHS)*

Title of Documentation	Reference Number
SoMachine - Programming Guide	*EIO0000000067 (ENG)* *EIO0000000069 (FRE)* *EIO0000000068 (GER)* *EIO0000000071 (SPA)* *EIO0000000070 (ITA)* *EIO0000000072 (CHS)*
Motion Control Library Guide	*EIO0000002221 (ENG)* *EIO0000002222 (GER)* *EIO0000002223 (CHS)*
TcpUdpCommunication Library Guide	*EIO0000002204 (ENG)* *EIO0000002255 (FRE)* *EIO0000002205 (GER)* *EIO0000002257 (SPA)* *EIO0000002256 (ITA)* *EIO0000002258 (CHS)*
Distributed Modbus TCP Logic Controller M251 - System User Guide	*EIO0000001680 (ENG)*
Compact EtherNet/IP Logic Controller M251 - System User Guide	*EIO0000002183 (ENG)*

You can download these technical publications and other technical information from our website at http://www.schneider-electric.com/en/download

Product Related Information

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Chapter 1
Industrial Ethernet Overview

What Is in This Chapter?

This chapter contains the following topics:

Overview

Overview

Industrial Ethernet is the term used to represent the industrial protocols that use the standard Ethernet physical layer.

On an Industrial Ethernet network, you can connect:

- industrial devices (industrial protocols)
- non-industrial devices (other Ethernet protocols)

In this document, Industrial Ethernet covers:

- EtherNet/IP
- Modbus TCP
- TCP/UDP

This document focuses on Industrial Ethernet devices connected on the device network of a logic controller.

Industrial Ethernet Features

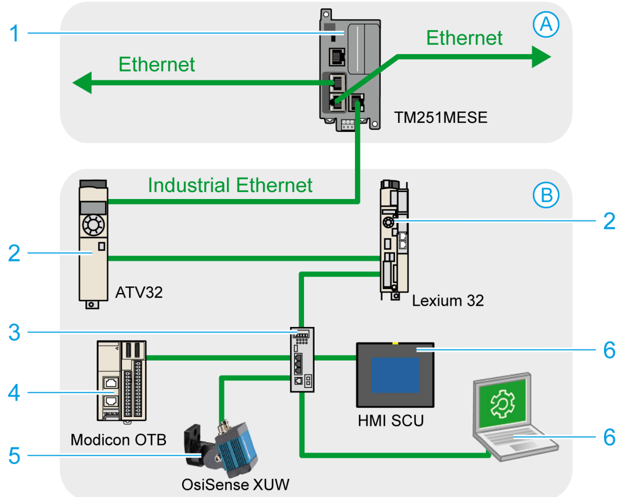
This table gives an overview of the Industrial Ethernet implementation features:

Feature	Industrial Ethernet capability
Number of devices and network flexibility	Star configuration. Infrastructure virtually unlimited. Superior total network distance. Advanced network management.
Data rate, wiring and distance	Capable of mixing fiber optic and copper cables within a system. 10/100 Mbit/s, up to 100 m (328 ft) for copper cable runs and up to 2000 m (6561 ft) for fiber optic cables. Noise immunity.
Protocols	Variety of protocols based on Ethernet.

Architecture

Industrial Ethernet Architecture

This figure presents a typical Industrial Ethernet architecture:



A Control network
B Device network
1 Logic controller *(see page 21)*
2 Daisy-chained devices
3 Ethernet switch
4 I/O island (Modbus TCP)
5 Vision sensor (EtherNet/IP)
6 PC and HMI (TCP/UDP)
2, 4, and 5 Industrial Ethernet devices (EtherNet/IP / Modbus TCP)

This architecture is configurable with SoMachine.

Principles

Overview

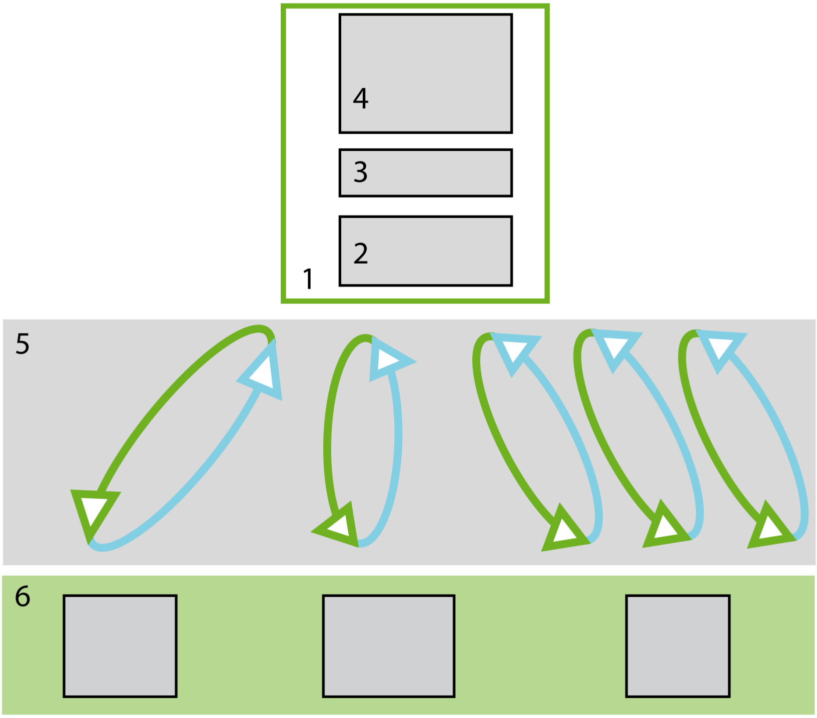
The logic controller controls Industrial Ethernet operating mode management. This management is performed using stable and cyclic data exchanges (scanner service).

Scanner services are available for the following protocols:

- EtherNet/IP *(see page 19)*
- Modbus TCP *(see page 19)*

Scanner Principle

Industrial Ethernet scanner principle:



1 Logic controller *(see page 21)*
2 I/O images
3 Application interface
4 Application
5 Data exchanges in Modbus channels or EtherNet/IP connections
6 Slave devices *(see page 23)*

Data Exchanges

Logic controller manages (for each supported protocol):

- Cyclic data exchanges
- Non-cyclic data exchanges

Cyclic data exchange (implicit messages in EtherNet/IP) is used when data must be exchanged at a constant rate such as:

- Scanning various I/O modules
- Updating a variable speed drive
- Reading input data on sensors

Non-cyclic data exchange (explicit messages in EtherNet/IP) is typically used to obtain on-demand information from the target devices, such as:

- Configuration
- Diagnostics
- Data collection

EtherNet/IP Overview

EtherNet/IP is the implementation of the CIP protocol over standard Ethernet.

The EtherNet/IP protocol uses an Originator/Target architecture for data exchange.

Originators are devices that initiate data exchanges with Target devices on the network. This applies to both I/O communications and service messaging. This is the equivalent of the role of a client in a Modbus network.

Targets are devices that respond to data requests generated by Originators. This applies to both I/O communications and service messaging. This is the equivalent of the role of a server in a Modbus network.

EtherNet/IP Adapter is an end-device in an EtherNet/IP network. I/O blocks and drives can be EtherNet/IP Adapter devices.

The communication between an EtherNet/IP Originator and Target is accomplished using an EtherNet/IP connection *(see page 57)*.

Modbus TCP Overview

The Modbus TCP protocol uses a Client/Server architecture for data exchange.

Modbus TCP explicit (non-cyclic) data exchanges are managed by the application.

Modbus TCP implicit (cyclic) data exchanges are managed by the Modbus TCP IOScanner. The Modbus TCP IOScanner is a service based on Ethernet that polls slave devices continuously to exchange data, status, and diagnostic information. This process monitors inputs and controls outputs of slave devices.

Clients are devices that initiate any data exchange with other devices on the network. This applies to both I/O communications and service messaging.

Servers are devices that address any data requests generated by a Client. This applies to both I/O communications and service messaging.

The communication between the Modbus TCP IOScanner and the slave device is accomplished using Modbus TCP channels *(see page 73)*.

Logic Controllers

Logic Controllers

This table presents the logic controllers that support Industrial Ethernet:

Parameter		TM251MESE, TM241CE24•, TM241CE40•, TM241CEC24•
Industrial Ethernet	Topology	Daisy chain and Star via switches
	Bandwidth	10/100 Mbit/s
EtherNet/IP Scanner	Performance	Up to 16 EtherNet/IP target devices managed by the logic controller, monitored within a timeslot of 10 ms.
	Number of connections	0…16
	Number of input words	0…1024
	Number of output words	0…1024
	I/O communications	EtherNet/IP Scanner service Function block for configuration and data transfer
		Originator/Target
Modbus TCP IO Scanner	Performance	Up to 64 Modbus TCP slave devices managed by the logic controller, monitored within a timeslot of 64 ms.
	Number of channels	0…64
	Number of input words	0…2048
	Number of output words	0…2048
	I/O communications	Modbus TCP IOScanner service Function block for data transfer
		Master/slave
Other services		FDT/DTM/EDS management
		FDR (Fast Device Replacement)
		DHCP server
		Security management (refer to Security Parameters and Firewall Configuration.
		Modbus TCP server
		Modbus TCP client
		EtherNet/IP Adapter (controller as a target on EtherNet/IP)
		EtherNet/IP Originator
		Modbus TCP server (controller as a slave on Modbus TCP)
		Web server
		FTP server
		SNMP
		IEC VAR ACCESS

Parameter	TM251MESE, TM241CE24•, TM241CE40•, TM241CEC24•
Additional features	Possible to mix up to 16 EtherNet/IP and Modbus TCP devices. Devices can be directly accessed for configuration, monitoring, and management purposes. Network transparency between control network and device network (logic controller can be used as a gateway). **NOTE:** Using the logic controller as a gateway can impact the performance of the logic controller.

NOTE: The input/output word limitations of the scanner have an impact on the number of devices on the device network. For example, with a TM251MESE, you can only connect up to 4 OsiSense XUW devices. Refer to Industrial Ethernet Manager Load Verification *(see page 78)*.

Industrial Ethernet Port

To configure the Industrial Ethernet port:

1. Double-click the following node in the **Devices tree**:
   - TM241CE24•/TM241CE40•: **MyController → Ethernet_1**
   - M251 Logic Controller: **MyController → Ethernet_2**
2. Configure the network settings (IP address, subnet mask, gateway address), and activate the DHCP server if using DHCP addressing.

Supported Devices

Supported Devices

This table presents the supported Industrial Ethernet devices:

Device name		Supported protocols			TVDA	Key features
		TCP/UDP	Modbus TCP	EtherNet/IP		
Predefined devices	Altivar 32	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Altivar 320	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Altivar 340	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Altivar 6••	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Altivar 71	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Altivar 9••	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Lexium 32 M	-	X	X	X	FDR, DTM, libraries, predefined connections, predefined data exchanges
	Lexium ILA	-	X	X	X	FDR, libraries, predefined connections, predefined data exchanges
	Lexium ILE	-	X	X	X	FDR, libraries, predefined connections, predefined data exchanges
	Lexium ILS	-	X	X	X	FDR, libraries, predefined connections, predefined data exchanges

(1) The device can be declared as a Modbus TCP slave device if you add it in SoMachine as a generic slave device.
(2) An EDS file provides, among other things, predefined connections to facilitate device network integration.
(3) A generic slave device is used in SoMachine to add devices such as speed drives, sensors, or other controllers that are Modbus TCP, EtherNet/IP, or TCP/UDP devices.

Device name		Supported protocols			TVDA	Key features
		TCP/UDP	Modbus TCP	EtherNet/IP		
	OsiSense XG	-	X	X	X	Predefined connections, predefined data exchanges
	OsiSense XUW	-	-	X	X	Predefined connections, predefined data exchanges
	OTB1EODM9LP	-	X	-	X	Libraries, predefined connections, predefined data exchanges
	XPSMCM	-	(1)	X	X	Predefined connections, predefined data exchanges
	Harmony XB4R/5R	-	X	-	-	DTM, libraries, predefined connections, predefined data exchanges
Other devices	Device provided with EDS file[2]	-	-	X	-	User parameters, predefined connections
	Generic slave device [3]	X	X	X	-	User parameters (only for EtherNet/IP), libraries

(1) The device can be declared as a Modbus TCP slave device if you add it in SoMachine as a generic slave device.
(2) An EDS file provides, among other things, predefined connections to facilitate device network integration.
(3) A generic slave device is used in SoMachine to add devices such as speed drives, sensors, or other controllers that are Modbus TCP, EtherNet/IP, or TCP/UDP devices.

Key Features

This table presents the key features:

Key features	Description
FDR	Fast Device Replacement: The device configuration is stored in the logic controller. When a device is replaced, the configuration is automatically loaded into the new device.
DTM	For devices supported by a DTM: FDT/DTM technology allows network devices to be configured in SoMachine. Refer to the Device Type Manager User Guide.
Libraries	Functions / function blocks (dedicated to the device) available for use by the application.
Predefined connections	Used to set up cyclic data exchanges. Select one of the proposed connections containing the relevant information. For more information, refer to Cyclic Data Exchanges *(see page 57)*.
Predefined data exchanges	The cyclic data exchanges are set automatically: one predefined connection is automatically selected when you add the device to the project.
User parameters	Parameters that are sent automatically to the device at power-up. These parameters are used when replacing devices that do not support FDR.

TVDA

The following TVDA (Tested Validated Documented Architecture) System User Guides are related to Industrial Ethernet:

- Distributed Modbus TCP Logic Controller M251
- Compact EtherNet/IP Logic Controller M251

Some supported Industrial Ethernet devices *(see page 23)* are provided with application code templates (referred to as Device Modules) that provide a way to integrate devices such as variable speed drives or servo drives in the SoMachine project. The Device Modules are realized on function templates, a mechanism within SoMachine to recall predefined application program contents.

Each Device Module embeds the SoMachine application content to control the field device, monitor its status, and handle detected errors. It includes a separate global variable definition that provides the interface to access the device functionalities across the SoMachine automation project.

For more details, refer to TVDA Device Module Library, Function Template Library Guide.

Setup Procedure Overview

Overview

This document is structured in accordance with the phases of a machine life cycle.

The chapters that follow contain information and procedures to follow to set up a use case system:

- Device network configuration *(see page 27)*
- Device network commissioning *(see page 83)*
- Device network operating *(see page 91)*
- Device network diagnostics *(see page 107)*
- Device network maintenance *(see page 119)*

Chapter 2
Device Network Configuration

Overview

This chapter contains the information and procedures necessary to configure the device network.

The device network configuration is prepared in SoMachine.

At the end of this phase, you can perform the device network commissioning *(see page 83)*.

What Is in This Chapter?

This chapter contains the following sections:

Section 2.1
Network Planning

Network Planning

Purpose

A planned network helps increase the installation efficiency and decrease the installation time and costs. The preliminarily interfacing of materials (switches, cables, ports) must be designed to plan the network.

Network Design

To design and plan the Industrial Ethernet network, refer to the corresponding documentation, such as the *Media Planning and Installation Manual*, by ODVA. You can download this manual from the *ODVA website*.

Switch Types

Depending on the specific needs of your network, use the appropriate switch type:

If you need…	then plan to use…
network diagnostics and operation information	manageable switches
communication availability in case of a physical connection loss	redundant switches
long range network (fiber optic)	switch with duplex SC connector

Hubs can reduce the available bandwidth. This can result in lost requests and devices no longer being managed.

NOTICE

LOSS OF DATA

Do not use a hub to set up an Industrial Ethernet network.

Failure to follow these instructions can result in equipment damage.

For more information about switches, refer to the *Essential Guide: Networks, connectivity and Web servers*.

Cable Types

These tables present cable references that can be used in the network.

In a typical installation, you can use these cables:

Reference	Description	Details	Length
490NTW000••	Ethernet shielded cable for DTE connections	Regular cable, equipped with RJ45 connectors at each end for DTE. CE compliant	2, 5, 12, 40, or 80 m (6.56, 16.4, 39.37, 131.23, or 262.47 ft)
490NTW000••U		Regular cable, equipped with RJ45 connectors at each end for DTE. UL compliant	2, 5, 12, 40, or 80 m (6.56, 16.4, 39.37, 131.23, or 262.47 ft)
TCSECE3M3M••S4		Cable for harsh environments, equipped with RJ45 connectors at each end. CE compliant	1, 2, 3, 5, or 10 m (3.28, 6.56, 9.84, 16.4,or 32.81 ft)
TCSECU3M3M••S4		Cable for harsh environments, equipped with RJ45 connectors at each end. UL compliant	1, 2, 3, 5, or 10 m (3.28, 6.56, 9.84, 16.4, or 32.81 ft)
TCSECL1M1M••S2••		Cable for harsh environments. 2 M12 connectors. CE compliant	1, 3, 10, 25, or 40 m (3.28, 9.84, 32.8, 82.02, or 131.23 ft)
TCSECL1M3M••S2••		Cable for harsh environments. 1 M12 connector 1 RJ45 connector CE compliant	1, 3, 10, 25, or 40 m (3.28, 9.84, 32.8, 82.02, or 131.23 ft)

In fiber optic networks, you can use these cables:

Reference	Description	Details	Length
490NOC00005	Glass fiber optic cable for DTE connections	1 SC connector 1 MT-RJ connector	5 m (16.4 ft)
490NOT00005		1 ST connector (BFOC) 1 MT-RJ connector	5 m (16.4 ft)
490NOR00003		2 MT-RJ connectors	3 m (9.8 ft)
490NOR00005		2 MT-RJ connectors	5 m (16.4 ft)

Section 2.2
IP Address Assignment Strategy

What Is in This Section?

This section contains the following topics:

IP Address Assignment Strategy

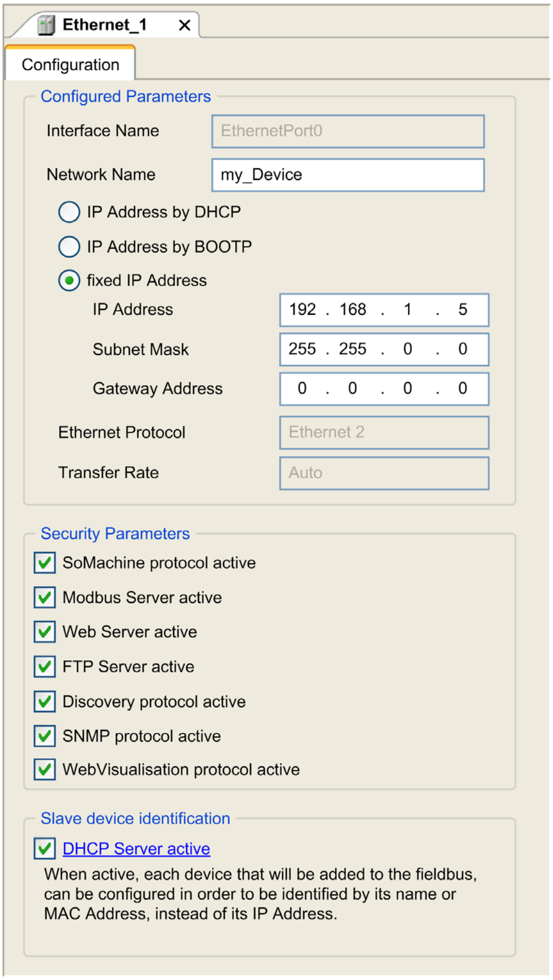
Overview

This section describes the steps to follow to implement the strategy for IP address assignment of the network devices:

- Configure the Industrial Ethernet port *(see page 22)* of the controller:
  - Network settings: IP address, subnet mask, and gateway address
  - Choose the IP addressing method *(see page 34)* to use.
- Configure the Industrial Ethernet manager

Industrial Ethernet Port Configuration

To configure the Industrial Ethernet port *(see page 22)*, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click the Industrial Ethernet port node. The configuration tab is displayed, for example:  Ethernet_1 Configuration Configured Parameters Interface Name: EthernetPort0 Network Name: my_Device ○ IP Address by DHCP ○ IP Address by BOOTP ● fixed IP Address IP Address: 192 . 168 . 1 . 5 Subnet Mask: 255 . 255 . 0 . 0 Gateway Address: 0 . 0 . 0 . 0 Ethernet Protocol: Ethernet 2 Transfer Rate: Auto Security Parameters ☑ SoMachine protocol active ☑ Modbus Server active ☑ Web Server active ☑ FTP Server active ☑ Discovery protocol active ☑ SNMP protocol active ☑ WebVisualisation protocol active Slave device identification ☑ DHCP Server active When active, each device that will be added to the fieldbus, can be configured in order to be identified by its name or MAC Address, instead of its IP Address.
2	Select **fixed IP Address**.
3	Set the **IP Address**. It must be different from `0.0.0.0`. This IP address is used in the Network Manager *(see page 40)*.
4	Set the **Subnet Mask**.

Step	Action
5	Verify that the **Gateway Address** is set by default to `0.0.0.0`. The gateway address allows a message to be routed to a device that is not on the local network. If there is no gateway, the gateway address is `0.0.0.0`.
6	Select the **Security Parameters** check boxes: ● **Web Server active**: used during configuration and maintenance phases ● **FTP Server active**: used by FDR service *(see page 51)*
7	Select the **DHCP Server active** check box if using a DHCP server to assign IP addresses. For more details, see IP Addressing Methods *(see page 34)*.

IP Addressing Methods

Presentation

This table presents the IP addressing methods:

Method	Description	Details
DHCP	The DHCP server uses the **DHCP device name** of the device to send it its IP address: The **DHCP device name** is also used by the FDR service.	New devices use the DHCP addressing method by default. By using DHCP, the FDR service is available. To replace a device: ● Install the new device ● Define the DHCP device name in the device ● Power up the device and launch the application. At power up, the new device is recognized and the logic controller loads the previously stored configuration into the new device.
BOOTP	The BOOTP server uses the **MAC Address** of the device to send its IP address:	To replace a device: ● Install the new device ● In SoMachine, enter the MAC address of the new device ● Build the application and load it in the logic controller. ● Configure the parameters in the device. ● Power up the device and launch the application.
Fixed	The IP address is fixed in the application.	To replace a device: ● Install the new device ● Configure in the device the network settings (IP address, subnet mask, and gateway address). ● Configure the parameters in the device directly or using SoMachine. ● Power up the device and launch the application.

Activating the DHCP Server

When using the DHCP addressing method, the DHCP server assigns IP addresses to devices upon request.

To activate the DHCP server, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click the Industrial Ethernet port *(see page 22)* node.
2	Select the **DHCP Server active** check box. When active, devices added to the fieldbus can be configured to be identified by DHCP device name instead of by MAC address or fixed IP address.

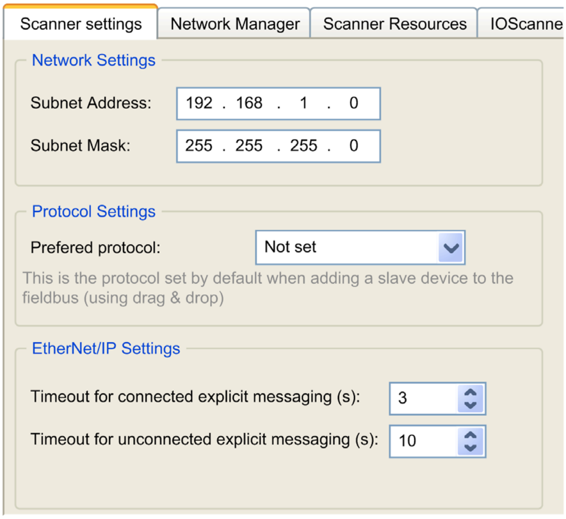
Industrial Ethernet Manager Configuration

Overview

The logic controller uses an Industrial Ethernet manager to manage the device network.

Industrial Ethernet Manager Settings

To configure the Industrial Ethernet manager, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **Industrial_Ethernet_Manager** above the Industrial Ethernet port *(see page 22)* node. **Result:**  Scanner settings Network Manager Scanner Resources IOScanne Network Settings Subnet Address: 192 . 168 . 1 . 0 Subnet Mask: 255 . 255 . 255 . 0 Protocol Settings Prefered protocol: Not set This is the protocol set by default when adding a slave device to the fieldbus (using drag & drop) EtherNet/IP Settings Timeout for connected explicit messaging (s): 3 Timeout for unconnected explicit messaging (s): 10 **NOTE:** The **Network Settings** are automatically generated in accordance with the Industrial Ethernet port network settings *(see page 31)*.
2	Select the **Preferred protocol**: ● **Not set** ● **EtherNet/IP** (by default) ● **Modbus TCP** Your selection becomes the device protocol set by default for each device declaration *(see page 36)*.
3	In the **EtherNet/IP Settings**, define the values of the explicit messaging timeout.

NOTE: When the Modbus TCP IOScanner is configured, the post configuration file for the Industrial Ethernet network is ignored.

Section 2.3
Network Device Declaration

Network Device Declaration

Overview

This section describes how to add a device on the **Industrial_Ethernet_Manager** node.

The available Schneider Electric devices, as well as devices supplied with EDS files, are listed in the **Hardware Catalog**. These devices are supplied with predefined connection configurations *(see page 23)*. For other devices not listed in the catalog, use **Generic slave device**.

Automatic Settings

During each device declaration, SoMachine automatically:

- Sets the network settings (IP address, subnet mask, gateway address) in accordance with the Industrial Ethernet scanner settings.
- Sets a unique DHCP device name, normally compatible with the internal rules of the device (each **DHCP device name** must be unique).
- Creates predefined data exchanges for predefined devices.

NOTE: If the proposed **DHCP device name** is not compatible with the device, you may edit it.

Add a Device

To add a device on the **Industrial_Ethernet_Manager** node, select the device in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the Industrial Ethernet port *(see page 22)* node.

NOTE: The **Industrial_Ethernet_manager** node is automatically added when a device is added on the Industrial Ethernet port node.

Once a device is added, it appears in the **Network Manager** tab of the **Industrial_Ethernet_Manager**. Refer to Adapting Network Planning and Device Identification *(see page 40)*.

When you use the Drag-and-Drop method, the devices are defined with the preferred protocol, if possible.

For more information on adding a device to your project, refer to:

- Using the Drag-and-drop Method *(see SoMachine, Programming Guide)*
- Using the Contextual Menu or Plus Button *(see SoMachine, Programming Guide)*

Add a Device with a Protocol Other Than the Preferred Protocol

When you use the Drag-and-Drop method:

- If the device cannot be defined with the preferred protocol, the default supported protocol of the device is used instead.
- If the preferred protocol is not set, a list appears to select the protocol to use.

To add a slave device with a protocol other than the preferred protocol, refer to Using the Contextual Menu or Plus Button *(see SoMachine, Programming Guide)*.

For example, when you add an OTB1EODM9LP device, it is configured with Modbus TCP even if the preferred protocol is set to EtherNet/IP.

Add Device from Template

For devices that do not have key features but support TVDA *(see page 23)*, it is possible to declare them using a template. This imports additional elements to facilitate program writing.

Use this method for OsiSense XGCS, XUW, and Preventa XPSMCM devices.

To add a device from a template to the **Industrial_Ethernet_Manager**, proceed as follows:

Step	Action
1	In the **Hardware Catalog**, select the **Device Template** check box.
2	Select the device in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the Industrial Ethernet port *(see page 22)*.

For more information on adding a device to your project, refer to:

- Using the Drag-and-drop Method *(see SoMachine, Programming Guide)*
- Using the Contextual Menu or Plus Button *(see SoMachine, Programming Guide)*

Add a TCP/UDP Device

To add a TCP/UDP device on the **Industrial_Ethernet_Manager** node, select **Generic TCP/UDP equipment** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the Industrial Ethernet port *(see page 22)* node.

Add a Device from an EDS File

Some third-party devices are delivered with an EDS file.

To add a device with an EDS file on the **Industrial_Ethernet_Manager**, proceed as follows:

Step	Action
1	In the SoMachine menu, select **Tools** → **Device Repository**.
2	Click **Install** to open the **Install Device Description** dialog box.
3	Select **EDS and DCF files** in the file type list.
4	Select the EDS file.

Step	Action
5	Click **OK** to close the dialog box.
6	Click **Close** to close the **Install Device Description** dialog box.
7	Select **Industrial_Ethernet_Manager** and click Plus button. Select the new added slave device and click **Add Device**. For more details, refer to Using the Contextual Menu or Plus Button *(see SoMachine, Programming Guide)*

Section 2.4
Adapting Network Planning and Device Identification

What Is in This Section?

This section contains the following topics:

Adapting Network Planning and Device Identification

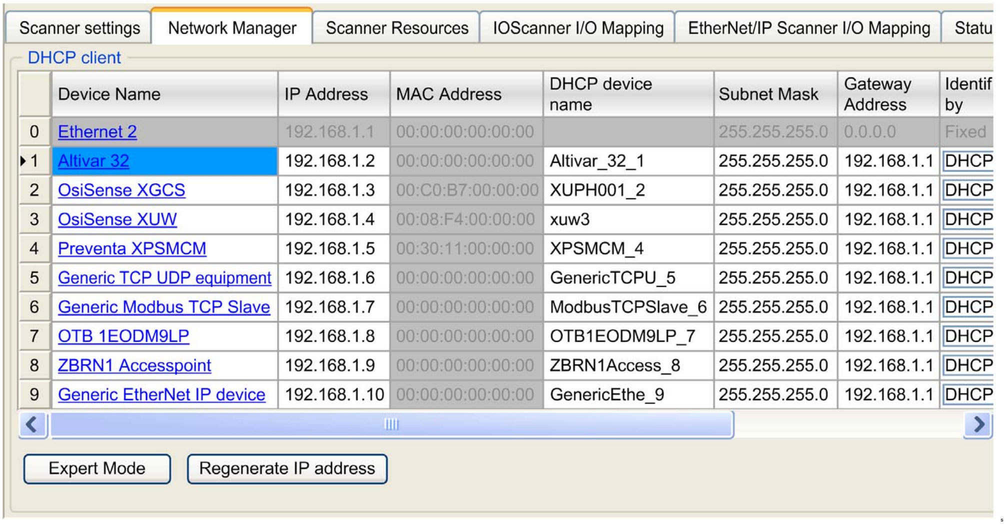
Overview

Once devices are added in the **Industrial_Ethernet_Manager**, use the **Network Manager** tab to edit the network planning.

Network Manager

In the **Devices tree**, double-click the **Industrial_Ethernet_Manager** node.

The **Network Manager** tab shows the devices defined on the device network:



	Device Name	IP Address	MAC Address	DHCP device name	Subnet Mask	Gateway Address	Identif by
0	Ethernet 2	192.168.1.1	00:00:00:00:00:00		255.255.255.0	0.0.0.0	Fixed
1	Altivar 32	192.168.1.2	00:00:00:00:00:00	Altivar_32_1	255.255.255.0	192.168.1.1	DHCP
2	OsiSense XGCS	192.168.1.3	00:C0:B7:00:00:00	XUPH001_2	255.255.255.0	192.168.1.1	DHCP
3	OsiSense XUW	192.168.1.4	00:08:F4:00:00:00	xuw3	255.255.255.0	192.168.1.1	DHCP
4	Preventa XPSMCM	192.168.1.5	00:30:11:00:00:00	XPSMCM_4	255.255.255.0	192.168.1.1	DHCP
5	Generic TCP UDP equipment	192.168.1.6	00:00:00:00:00:00	GenericTCPU_5	255.255.255.0	192.168.1.1	DHCP
6	Generic Modbus TCP Slave	192.168.1.7	00:00:00:00:00:00	ModbusTCPSlave_6	255.255.255.0	192.168.1.1	DHCP
7	OTB 1EODM9LP	192.168.1.8	00:00:00:00:00:00	OTB1EODM9LP_7	255.255.255.0	192.168.1.1	DHCP
8	ZBRN1 Accesspoint	192.168.1.9	00:00:00:00:00:00	ZBRN1Access_8	255.255.255.0	192.168.1.1	DHCP
9	Generic EtherNet IP device	192.168.1.10	00:00:00:00:00:00	GenericEthe_9	255.255.255.0	192.168.1.1	DHCP

Column	Use	Comment
Device Name	Click to open the device settings	Name of the device. A name is given by default. To rename your device, type a name in the **Name** box. Do not use spaces within the name. Do not use an underscore (“_”) at the end of the name. Give the device a meaningful name to facilitate the organization of your project.
IP Address	Modify the IP address	An IP address is shown as invalid if it has already been assigned to another device that uses the same protocol and DHCP address assignment. If the IP address is invalid, the icon ❗ is displayed.

Column	Use	Comment
MAC Address	Enter the MAC address	Used to retrieve an IP address through BOOTP. Each IP address must be unique for a particular protocol and for DHCP/BOOTP. For example, you can add the same device for both Modbus TCP and Ethernet/IP protocols, but if you use BOOTP or DHCP to obtain an IP address for one of the protocols, you must enter that same IP address for the other protocol as a **Fixed** IP address.
DHCP device name	Modify the DHCP device name	Used as device name to retrieve an IP address through DHCP, maximum 16 characters. The **DHCP device name** must be the same as that defined in the device. Each **DHCP device name** must be unique. The default **DHCP device name** is normally compatible with the internal rules of the device. For details on **DHCP device name** internal rules of the device, refer to the documentation of the device. **NOTE:** If the proposed **DHCP device name** is not compatible with the device, you may edit it.
Subnet Mask	Modify the subnet mask	Click **Expert Mode** to hide/show the column.
Gateway Address	Modify the gateway address	Click **Expert Mode** to hide/show the column. For operational details, refer to Out of Process Data Exchanges *(see page 98)*.
Identified by	Modify the IP addressing method: ● **DHCP** ● **BOOTP** ● **Fixed**	**DHCP**: The **DHCP device name** must be the same as defined in the device. This method is mandatory for the FDR service.
		BOOTP: The **MAC Address** of the device must be entered.
		Fixed: The **IP Address** must be the same as defined in the device.
Device Type	-	Device type
Protocol	-	Protocol used

The modifications made in this tab are applied in the corresponding device setting tab:

- EtherNet/IP Target settings *(see page 44)*
- Modbus TCP settings *(see page 46)*

IP Addressing Methods

By default, the added devices use DHCP.

This table presents the IP addressing methods:

Method	Description	Details
DHCP	The DHCP server uses the **DHCP device name** of the device to send it its IP address: The **DHCP device name** is also used by the FDR service.	By using DHCP, the FDR service is available. To replace a device, you have to: ● Install the new device ● Define the DHCP device name in the device ● Power up the device and launch the application. At power-up, the new device is recognized and the logic controller loads the previously stored configuration into the new device.
BOOTP	The BOOTP server uses the **MAC Address** of the device to send its IP address:	To replace a device, you have to: ● Install the new device ● In SoMachine, enter the MAC address of the new device ● Build the application and load it in the logic controller. ● Configure the parameters in the device. ● Power up the device and launch the application.
Fixed	The IP address is fixed in the application.	To replace a device, you have to: ● Install the new device ● Configure in the device the network settings (IP address, subnet mask, and gateway address). ● Configure the parameters in the device directly or using SoMachine. ● Power up the device and launch the application.

Reinitialize IP Address Plan

Click **Regenerate IP address** to reinitialize the IP address plan associated with the Industrial Ethernet port *(see page 22)* (for example, after a change of IP address on the Industrial Ethernet port).

SoMachine reads the IP address configured on the Industrial Ethernet port *(see page 22)* and assigns the next available IP addresses to the devices. For example, if the IP address configured on the Industrial Ethernet port *(see page 22)* is `192.168.0.11`, the IP addresses attributed to the devices are `192.168.0.12`, `192.168.0.13`, and so on.

Out of Process Data Exchanges

Out of process data exchanges are often data exchanges between the control network and the device network. For example, you may use a supervision software or a third-party configuration tool to communicate with a target on the device network.

For more operational details, refer to Out of Process Data Exchanges *(see page 98)*.

If you need an out of process data exchange, set the correct device gateway address parameter.

The gateway address parameter of the network devices must be set to the IP address of the Industrial Ethernet port *(see page 22)* of the logic controller.

A configuration tool must be able to communicate with the network devices in order to set their parameters.

If the configuration tool...	Then...
is connected on the control network	update the network device gateway parameter, see below.
is connected on the device network	the gateway parameter is not used.
uses a protocol other than TCP/IP	the gateway parameter is not used.

To configure the gateway parameter in the network device, refer to the documentation of the device.

NOTE: If the DHCP service is used to address the network devices, the gateway parameter is set in the logic controller network manager tab *(see page 40)*.

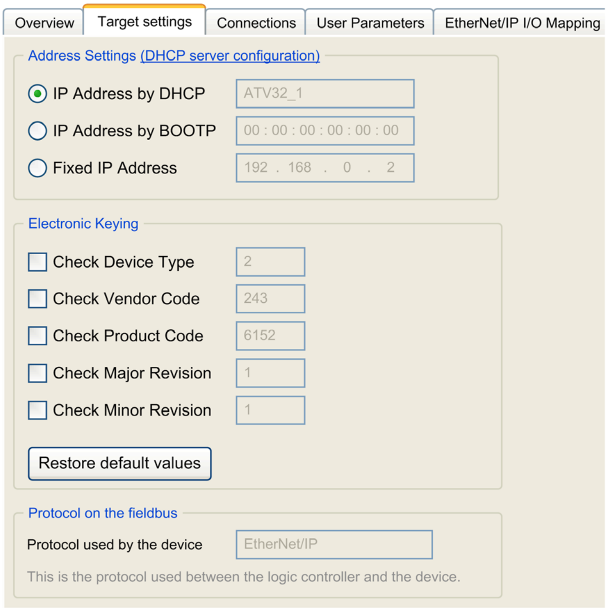
EtherNet/IP Target Settings

Overview

Once the devices are added in the **Industrial_Ethernet_Manager**, use its **Target Settings** tab to edit the network planning.

EtherNet/IP Target Settings

In the **Devices tree**, double-click an EtherNet/IP device node:



Address Settings (DHCP server configuration)

● IP Address by DHCP: ATV32_1
○ IP Address by BOOTP: 00 : 00 : 00 : 00 : 00 : 00
○ Fixed IP Address: 192 . 168 . 0 . 2

Electronic Keying

□ Check Device Type: 2
□ Check Vendor Code: 243
□ Check Product Code: 6152
□ Check Major Revision: 1
□ Check Minor Revision: 1

Protocol on the fieldbus

Protocol used by the device: EtherNet/IP

This is the protocol used between the logic controller and the device.

The **Address Settings** values are the same as those defined in the **Industrial_Ethernet_Manager**. Refer to Adapting Network Planning and Device Identification *(see page 40)*.

Electronic Keying

Electronic Keying signatures are used to identify the device.

Electronic Keying is information contained in the firmware of the device (Vendor Code, Product Code, …).

When the scanner starts, it compares each selected electronic keying value with the corresponding information in the device.

If the device values are not the same as the application values, the logic controller no longer communicates with the device.

For pre-configured devices, you cannot modify the **Electronic Keying** values.

For generic EtherNet/IP devices, you can modify the **Electronic Keying** values.

For **Electronic Keying** values, refer to the Identity Object (F1 hex) description in the documentation of the device.

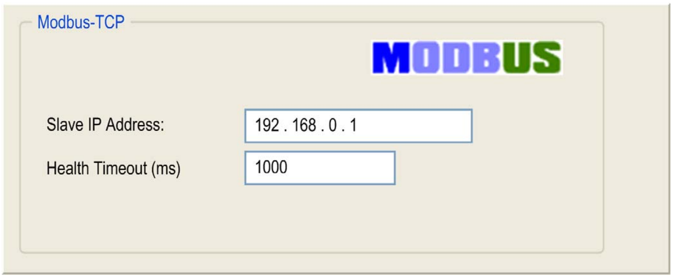
Modbus TCP Settings

Overview

Once the devices are added in the **Industrial_Ethernet_manager**, use its **Network manager** tab to edit the network planning.

Modbus TCP Settings

To configure pre-defined slave devices added on the Modbus TCP IOScanner, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click a Modbus TCP slave device node. **Result:** The configuration window is displayed:  Slave IP Address: 192 . 168 . 0 . 1 Health Timeout (ms) 1000
2	Enter a **Slave IP Address** value. The **Address settings** values are the same as those defined in the **Industrial_Ethernet_manager** *(see page 40)*.
3	Enter a **Health Timeout (ms)** value (by default 1000). This represents the delay (in ms) between a request of the Modbus TCP IOScanner and a response from the slave. When the health timeout expires, the associated health bit values change to 0. Health bit values can be visualized in the **IOScanner I/O Mapping** tab *(see page 76)* or through the Web server. The health timeout applies to the channels of the slave device.
4	For devices with advanced settings, some additional settings can be required: ● **Repetition rate (ms)**: Time value expressed in ms. This represents the delay between two sendings of a request. This value must be lower than the **Health Timeout (ms)**. ● **Unit ID**: unit ID of the Modbus TCP slave device (by default 255). Refer to the Device Type Manager User Guide.

Section 2.5
Network Device Configuration

Network Device Configuration

Overview

Once the network devices are defined on the device network, you can configure them with:

- User Parameters
- DTM
- Plugins
- Third-party tools

Description	Advantages
User Parameters	Available for EtherNet/IP devices. User Parameters are usable for device replacement. The User Parameters are written to the device each time the communication with the device starts.
DTM	Can manage complex configurations.
Plugins	Good transparency. Specifically designed for SoMachine.
Third-party tools	Tools specifically designed for the device.

User Parameters

Refer to User Parameters *(see page 52)*.

Devices with DTM

Some devices have a DTM. Refer to supported devices *(see page 23)*.

The DTM allows you to modify the parameters of the device.

To configure a device with its DTM, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click the device.
2	Select the **Configuration of the device** tab.
3	Click **OK**. **Result:** The content of the tab is updated by the DTM.

Step	Action
4	Modify the device configuration. For more information, refer to Device Type Manager User Guide.

NOTE: The use of a DTM may require a particular routing and IP forwarding *(see SoMachine, Device Type Manager (DTM), User Guide)* configuration on the logic controller.

Devices with Plugins

Depending on the plugin, the User Parameters may not be available. In that case, it is up to the plugin to manage the configuration of the device.

Example: Advantys OTB1EODM9LP

The Advantys OTB1EODM9LP is supported in SoMachine by a library. One function block is a configuration function block that allows you to send the configuration to the device. For more details, refer to the Distributed Modbus TCP Logic Controller M251 System User Guide.

To configure an OTB1EODM9LP, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click the OTB1EODM9LP node.
2	Configure the I/Os of the Advantys OTB device in the **OTB I/O Configuration** tab.
3	Add and configure any TM2 expansion modules attached to the OTB.
4	Call a `CONFIGURE_OTB` function block to update the Advantys OTB configuration with the data created on the previous steps.

NOTE: The expert functions of the Advantys OTB such as counters, fast counters, and pulse generators, cannot be directly used in the Industrial Ethernet scanner.

Third-party Tools

Some devices are configured outside of SoMachine (specific software, keypad, Web server, and so on).

For more details, refer to the documentation of the device.

Master IP Address Parameter

Some devices have a **Master IP address** parameter so that only one, declared Master, logic controller has access to the device.

If the device...	Then...
is configured to use the Industrial Ethernet manager	configure the **Master IP address** parameter inside the device, see below.
is not configured to use the Industrial Ethernet manager	use `0.0.0.0` for the **Master IP address** parameter in the device.

The **Master IP address** parameter of the device has to be set to the IP address of the logic controller supporting the Industrial Ethernet manager (configured on the Industrial Ethernet port *(see page 22)*).

To configure this parameter in the device, refer to the documentation of the device.

Section 2.6
Network Device Replacement

Overview

The device replacement strategy can be managed with:

- FDR service
- User Parameters

What Is in This Section?

This section contains the following topics:

Device Replacement with FDR

FDR Overview

Some devices support the Fast Device Replacement (FDR) service.

The FDR service stores network and operating parameters of devices on the network. If a device is replaced, the service automatically configures the replacement device with parameters identical to those of the removed device.

In order to configure this service in the device, refer to the documentation of the device.

The FDR server relies on two advanced services embedded in the logic controller:

- DHCP server for device address assignment
- FTP server for device parameter files. This optional service is used only by devices that contain parameters.

The DHCP server allows the configuration of the new device with the same addressing parameters.

Devices that contain parameters use the FTP server to save their parameter files.

The replacing device requests the FTP server to restore the parameter files.

Device Replacement with User Parameters

Overview

For EtherNet/IP devices that do not support the FDR service, you can configure **User Parameters** that are sent to the device to facilitate device replacement just before the scanner connection is started after:

- Application download
- Reset warm/cold start
- Manual start of a connection

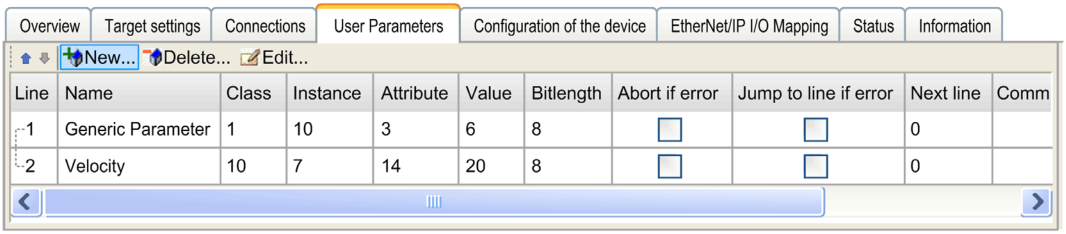
Some EtherNet/IP devices have predefined **User Parameters**.

The **User Parameters** tab allows you to add and manage other parameters.

For maintenance details, refer to Apply the Correct Device Configuration *(see page 89)*.

User Parameters

In the **Devices tree**, double-click an EtherNet/IP device and select the **User Parameters** tab:



Line	Name	Class	Instance	Attribute	Value	Bitlength	Abort if error	Jump to line if error	Next line	Comm
1	Generic Parameter	1	10	3	6	8	☐	☐	0	
2	Velocity	10	7	14	20	8	☐	☐	0	

Column	Description
Line	Line number. Indicates the order of the parameters loaded to the device.
Name	Name of the parameter.
Class	Class ID(1) of the class corresponding to the object.
Instance	Instance ID(1) of the instance corresponding to the object.
Attribute	Attribute ID(1) of the attribute corresponding to the object.
Value	Value of the parameter. Double-click the value to modify it. If applicable, a list opens containing possible values.
Bitlength	Number of bits of the parameter. Automatically changed depending of the parameter datatype selected.
Abort if error	If selected, when an error is detected, the transmission of the parameters is aborted.

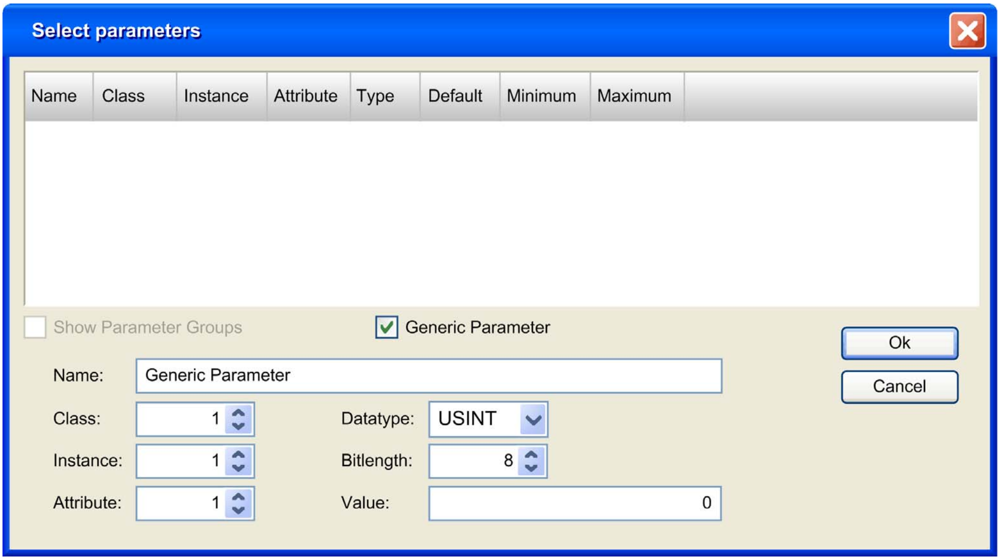
(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find User Parameter Information *(see page 54)*.

Column	Description
Jump to line if error	If selected, when an error is detected, the program resumes with the line specified in the **Next line** column. A block can thus be skipped during the initialization, or a return can be defined. **NOTE:** A return can lead to an endless loop if it is never possible to write a certain parameter.
Next line	Double-click to enter the line to jump to (if **Jump to line if error** is selected).
Comment	Double-click to enter a comment.
[1] The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find User Parameter Information *(see page 54)*.	

Icons	Description
Move up	Move up the selected parameter in the parameters list.
Move down	Move down the selected parameter in the parameters list.
New	Creates a new parameter.
Delete	Delete the selected parameter.
Edit	Edit the selected parameter.

Creating or Configuring User Parameters

Click **New**, or select a parameter and click **Edit**:



Select parameters

Name	Class	Instance	Attribute	Type	Default	Minimum	Maximum

□ Show Parameter Groups

☑ Generic Parameter

Name: Generic Parameter

Class: 1

Datatype: USINT

Instance: 1

Bitlength: 8

Attribute: 1

Value: 0

Fields	Description
Name	Name of the parameter.
Class	Class ID(1) of the class corresponding to the type of object.
Instance	Instance ID(1) of the instance corresponding to an implementation of a class.
Attribute	Attribute ID(1) of the attribute corresponding to a characteristic of an instance.
Datatype	List containing the possible data type.
Bitlength	Number of bits of the parameter. Automatically changed depending on the selected **Datatype**.
Value	Value of the parameter.
(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find User Parameter Information *(see page 54)*.	

How To Find User Parameter Information

Configurable user parameter information is provided in the device documentation. It is usually part of the description of application objects, explicit messaging, or objects belonging to EtherNet/IP category 3.

User parameter write access is usually specified for the class and/or instance to which the user parameter belongs. The write operation is typically performed using a service called `Set_Attribute_Single` or `Write one attribute`. Alternatively, a service identifier 0x10 (hexadecimal) or 16 (decimal) may be supported.

A user parameter always has the following numeric properties:
- Class, or Class ID, usually expressed as an hexadecimal value
- Instance, or Instance ID, usually expressed as an hexadecimal value
- Attribute, or Attribute ID, usually expressed as an hexadecimal value

A user parameter may also have an identifier, expressed in the form of a decimal triplet (xx/yy/zz) or hexadecimal triplet (16#xx/yy/zz)

Section 2.7
Cyclic Data Exchanges Configuration

What Is in This Section?

This section contains the following topics:

Cyclic Data Exchanges Overview

Overview

The Industrial Ethernet manager supports cyclic data exchanges (implicit messaging) between the logic controller and the slave devices.

The cyclic data exchange requests are supported by:

- A connection for EtherNet/IP.
- A channel for Modbus TCP.

Predefined devices have predefined data exchanges, for which the cyclic data exchanges are automatically defined. Devices with an EDS file have predefined connections—you must select the connection or channel to use with your application.

If necessary, you can configure these data exchanges using the dedicated DTM or the appropriate third-party tool. For details, refer to the documentation of the device.

You can add and configure new requests for these devices and generic slave devices.

For all data exchanges, you can map variables to be used by the program.

EtherNet/IP Cyclic Data Exchanges Configuration

Connection Overview

To access an EtherNet/IP device, it is necessary to start a connection (global name used by EtherNet/IP protocol level).

A connection allows to transfer data combined into assembly *(see page 57)*.

The connections processes (start/stop) are automatically managed by the logic controller.

For connections limitations, refer to the logic controller Programming Guide.

For more details, refer to Industrial Ethernet Manager Operating Modes *(see page 100)*.

Assembly

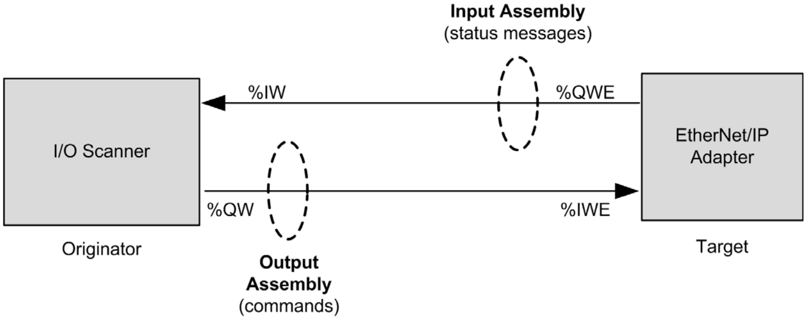
I/O data and configuration data can be combined into Assembly Objects.

Data (attributes) from different objects can be combined into a single object to allow data to be sent or received over a single connection.

Assembly Object instances are used to aggregate data for the input data and output data associated with I/O connections.

Assembly objects are structured into classes, instances, and attributes:
- A class is a set of objects that represent the same kind of system component.
- An object instance is the representation of a particular object within a class. Each instance has its own set of attribute values.
- Attributes are characteristics of an object and/or an object class. Typically, attributes provide status information or define the operation of an object.

The following graphic presents the directionality of Input Assembly and Output Assembly in EtherNet/IP communications:



The EtherNet/IP configuration parameters are defined as:

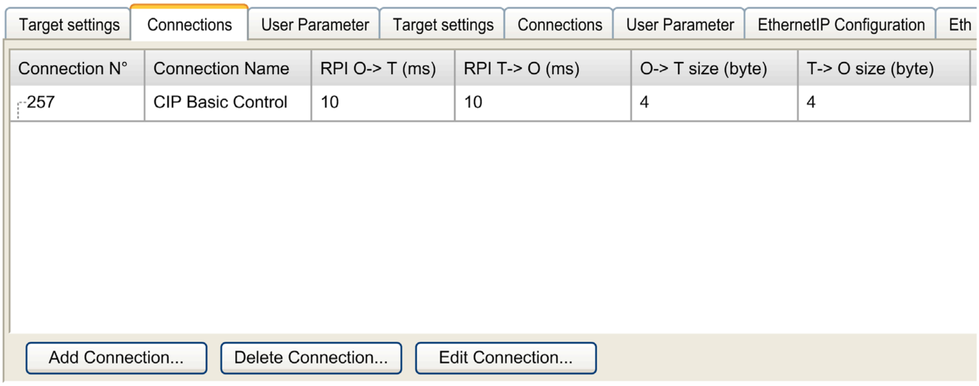
- **Instance:** Number referencing the assembly.
- **Size:** Number of channels of an assembly.
  The memory size of each channel is 2 bytes, which store the value of `%IWx` or `%QWx` objects, where x is the channel number.

For example, if the **Size** of the **Output Assembly** is 20, there are 20 input channels (IW0…IW19) addressing `%IWy…%IW(y+20-1)`, where y is the first available channel for the assembly.

EtherNet/IP Device Connection Tab

Each EtherNet/IP device has connections.

In the **Devices tree**, double-click an EtherNet/IP device and select the **Connections** tab.



Connection N°	Connection Name	RPI O-> T (ms)	RPI T-> O (ms)	O-> T size (byte)	T-> O size (byte)
257	CIP Basic Control	10	10	4	4

Column	Comment
Connection N°	The connection number is unique. It is automatically assigned by SoMachine.
Connection Name	The connection name is generated automatically by SoMachine.
RPI O --> T (ms)	Requested Packet Interval: The time period between cyclic data transmissions requested by the scanner.
RPI T --> O (ms)	
O->T size (byte)	Number of bytes to exchange between the Originator (O) and the Target (T).
T->O size (byte)	
Config#1 size (byte)	Number of bytes of configuration parameters to transfer. Displayed if the connection contains a configuration assembly *(see page 62)*.
Config#2 size (byte)	

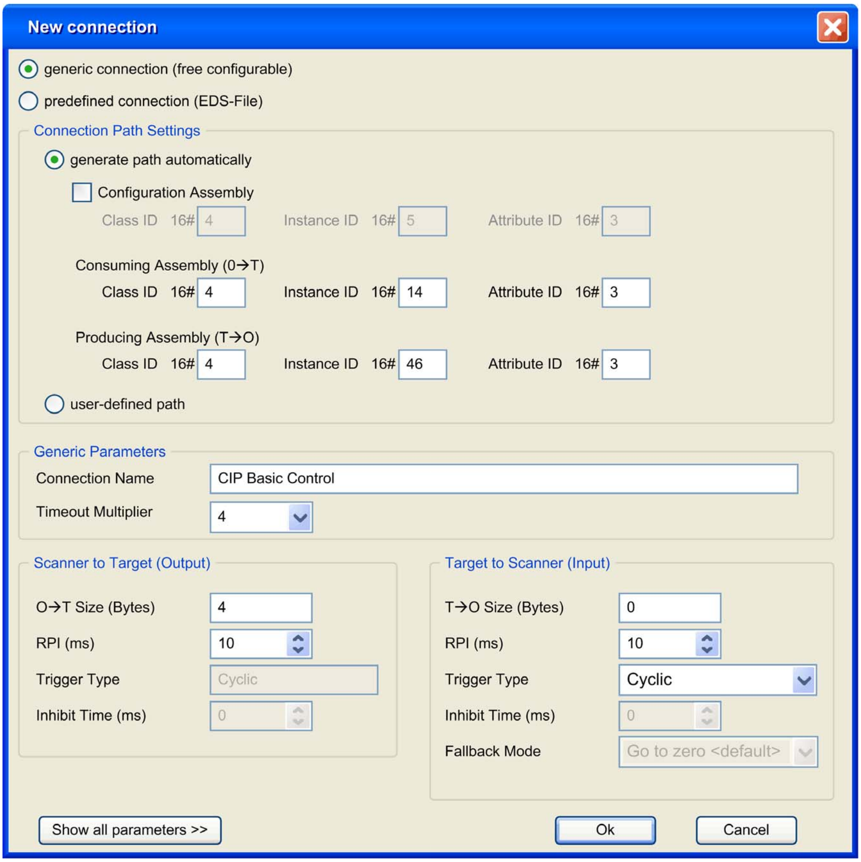
To create a connection, click **Add Connection**.

To modify a connection, select a connection and click **Edit Connection**, or double-click on it.

To remove a connection, select a connection and click **Delete Connection**.

Add an EtherNet/IP Connection

To configure an EtherNet/IP connection, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click an EtherNet/IP device.
2	Select the **Connections** tab.
3	Click **Add Connection**.
4	Select **generic connection (free configurable)**:  New connection ● generic connection (free configurable) ○ predefined connection (EDS-File) Connection Path Settings ● generate path automatically □ Configuration Assembly Class ID 16# 4 Instance ID 16# 5 Attribute ID 16# 3 Consuming Assembly (0→T) Class ID 16# 4 Instance ID 16# 14 Attribute ID 16# 3 Producing Assembly (T→O) Class ID 16# 4 Instance ID 16# 46 Attribute ID 16# 3 ○ user-defined path Generic Parameters Connection Name: CIP Basic Control Timeout Multiplier: 4 Scanner to Target (Output) O→T Size (Bytes): 4 RPI (ms): 10 Trigger Type: Cyclic Inhibit Time (ms): 0 Target to Scanner (Input) T→O Size (Bytes): 0 RPI (ms): 10 Trigger Type: Cyclic Inhibit Time (ms): 0 Fallback Mode: Go to zero <default>
5	Select **Configuration assembly** *(see page 62)*.
[1] The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information *(see page 69)*.	

Step	Action
6	Configure the **Consuming assembly (O --> T)**: ● **Class ID** (4 by default): Class identifier[1] ● **Instance ID**: Instance identifier[1] ● **Attribute ID** (3 by default): Attribute identifier[1]
7	Configure the **Producing assembly (T --> O)**: ● **Class ID** (4 by default): Class identifier[1] ● **Instance ID**: Instance identifier[1] ● **Attribute ID** (3 by default): Attribute identifier[1]
8	Select the **Timeout Multiplier**: 4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512
9	Configure the **Scanner to Target (Output)**: ● **O --> T Size (Bytes)**: Number of bytes to transfer: up to 505 ● **Trigger Type**: Cyclic ● **RPI (ms)** (10 ms by default): The time period between cyclic data transmissions requested by the scanner.
10	Configure the **Target to Scanner (Input)**: ● **T --> O Size (Bytes)**: Number of bytes to transfer (Number of channels of the assembly): up to 509) ● **Trigger Type**: Cyclic/Change of state. If **Change of state** is selected, then **Inhibit Time** is enabled and set to the default value of 2 ms ● **RPI (ms)** (10 ms by default): The time period between cyclic data transmissions requested by the scanner ● **Inhibit Time (ms)** (2 ms by default): Minimum period of time between 2 data exchanges. Accessible if **Trigger Type** is **Change of state**. The value must be a multiple of 2 ms. The maximum value is the target to scanner **RPI (ms)** value, up to a maximum possible value of 254 ms.
11	Click **OK**.
[1] The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information *(see page 69)*.	

For more details on supported assemblies, refer to the documentation of the device.

For more details on advanced parameters, refer to EtherNet/IP Connection Properties in Expert Mode *(see page 64)*.

NOTE: Due to the **O --> T Size (Bytes)** and **T --> O Size (Bytes)** limitations and the maximum input/output words of the scanner (1024), verify the scanner resources overload *(see page 78)*.

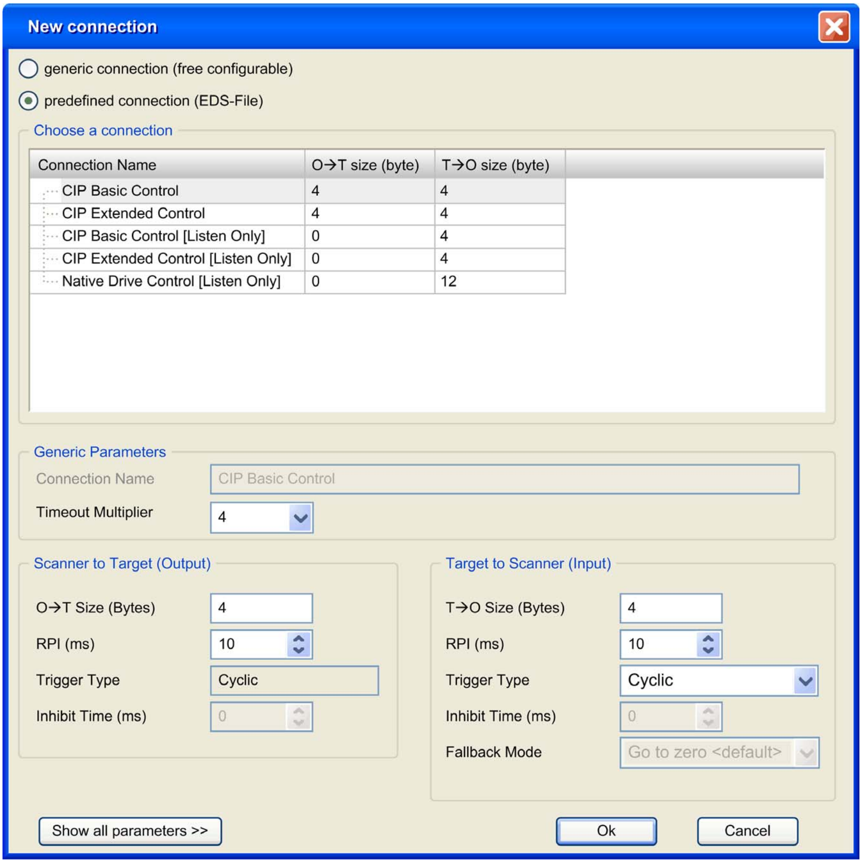
Add a Predefined Connection

Predefined connections are available for:

- Predefined devices *(see page 23)*.
- Devices that are supported by DTM.
- Devices that are delivered with an EDS file.

By definition, generic slave devices do not have predefined connections.

To add a predefined EtherNet/IP connection, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click an EtherNet/IP device.
2	Select the **Connections** tab.
3	Click **Add Connection**.
4	Select **predefined connection (EDS-File)**: 

New connection

○ generic connection (free configurable)

● predefined connection (EDS-File)

Choose a connection

Connection Name	O→T size (byte)	T→O size (byte)
CIP Basic Control	4	4
CIP Extended Control	4	4
CIP Basic Control [Listen Only]	0	4
CIP Extended Control [Listen Only]	0	4
Native Drive Control [Listen Only]	0	12

Generic Parameters

Connection Name: CIP Basic Control

Timeout Multiplier: 4

Scanner to Target (Output)

O→T Size (Bytes): 4

RPI (ms): 10

Trigger Type: Cyclic

Inhibit Time (ms): 0

Target to Scanner (Input)

T→O Size (Bytes): 4

RPI (ms): 10

Trigger Type: Cyclic

Inhibit Time (ms): 0

Fallback Mode: Go to zero <default>

Step	Action
4	Select one of the predefined connections.
5	Select the **Timeout Multiplier**: 4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512
6	Configure the **Scanner to Target (Output)**: ● **O --> T Size (Bytes)**: Number of bytes to transfer ● **Trigger Type**: Cyclic ● **RPI (ms)** (default value is defined in the EDS): The time period between cyclic data transmissions requested by the scanner.
7	Configure the **Target to Scanner (Input)**: ● **T --> O Size (Bytes)**: Number of bytes to transfer (Number of channels of the assembly) ● **Trigger Type**: Cyclic/Change of state. If **Change of state** is selected, then **Inhibit Time** is enabled and set to the default value of 2 ms ● **RPI (ms)** (default value is defined in the EDS): The period of time between cyclic data transmissions requested by the scanner ● **Inhibit Time (ms)** (2 ms by default): Minimum period of time between 2 data exchanges. Accessible if **Trigger Type** is **Change of state**. The value must be a multiple of 2 ms. The maximum value is the target to scanner **RPI (ms)** value, up to a maximum possible value of 254 ms.
8	Click **OK**.

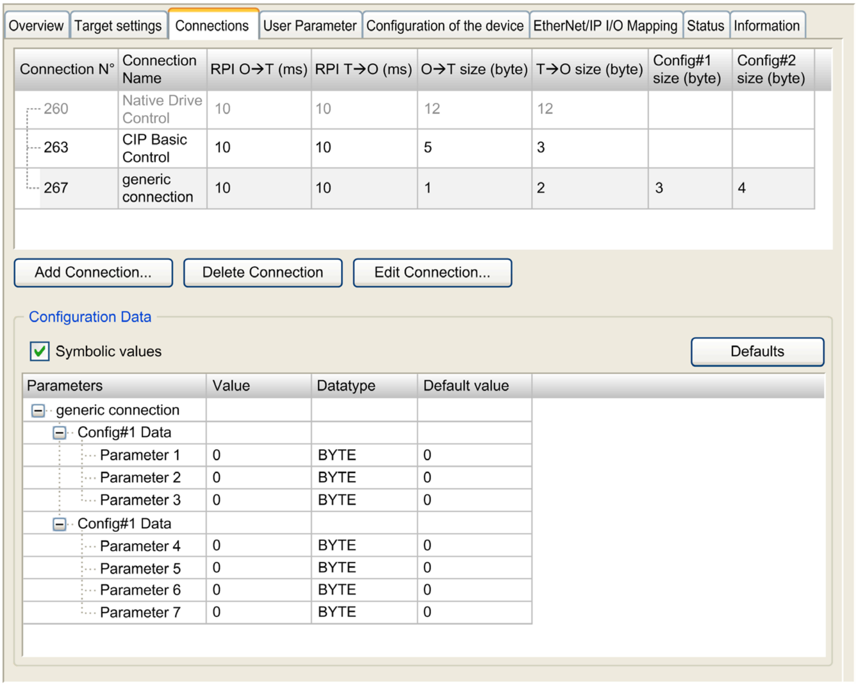
Configure a Configuration Assembly

Some devices support a configuration assembly.

A configuration assembly is a request, sent at the scanner start, that loads configuration parameters to the device in a single request.

To configure a configuration assembly, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click an EtherNet/IP device.
2	Select the **Connections** tab.
3	Select an existing connection and click **Edit Connection**.
4	Select **generic connection (free-configurable)**.
5	Select **Configuration Assembly**.
6	Configure the **Configuration Assembly**: ● **Class ID** (4 by default): Class identifier(1) ● **Instance ID**: Instance identifier(1) ● **Attribute ID** (3 by default): Attribute identifier(1)
7	Click **Show all parameters >>>**.
8	Configure the **Scanner to Target (Output)**: ● **Config#1 Size (Bytes)**: Number of the first set of configuration parameters. ● **Config#2 Size (Bytes)**: Number of the second set of configuration parameters.

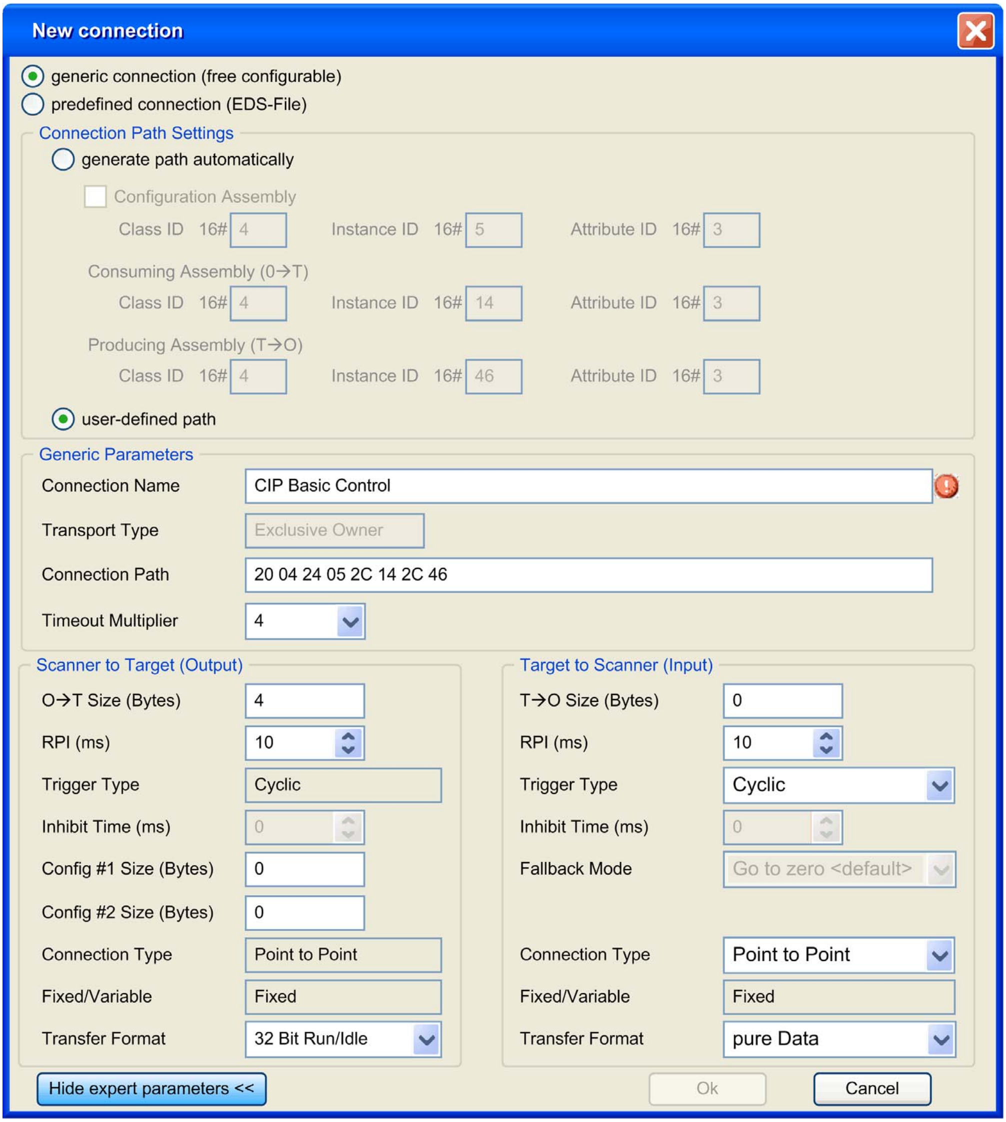
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>9</td><td>Click <b>OK</b>.
<b>Result:</b> The configuration parameters are displayed in the <b>Connections</b> tab:


<table>
<tr><th>Connection N°</th><th>Connection Name</th><th>RPI O→T (ms)</th><th>RPI T→O (ms)</th><th>O→T size (byte)</th><th>T→O size (byte)</th><th>Config#1 size (byte)</th><th>Config#2 size (byte)</th></tr>
<tr><td>260</td><td>Native Drive Control</td><td>10</td><td>10</td><td>12</td><td>12</td><td></td><td></td></tr>
<tr><td>263</td><td>CIP Basic Control</td><td>10</td><td>10</td><td>5</td><td>3</td><td></td><td></td></tr>
<tr><td>267</td><td>generic connection</td><td>10</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
</table>

☑ Symbolic values
<table>
<tr><th>Parameters</th><th>Value</th><th>Datatype</th><th>Default value</th></tr>
<tr><td>generic connection</td><td></td><td></td><td></td></tr>
<tr><td>Config#1 Data</td><td></td><td></td><td></td></tr>
<tr><td>Parameter 1</td><td>0</td><td>BYTE</td><td>0</td></tr>
<tr><td>Parameter 2</td><td>0</td><td>BYTE</td><td>0</td></tr>
<tr><td>Parameter 3</td><td>0</td><td>BYTE</td><td>0</td></tr>
<tr><td>Config#1 Data</td><td></td><td></td><td></td></tr>
<tr><td>Parameter 4</td><td>0</td><td>BYTE</td><td>0</td></tr>
<tr><td>Parameter 5</td><td>0</td><td>BYTE</td><td>0</td></tr>
<tr><td>Parameter 6</td><td>0</td><td>BYTE</td><td>0</td></tr>
<tr><td>Parameter 7</td><td>0</td><td>BYTE</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>10</td><td>Double-click in the <b>Value</b> column to set the configuration parameter values.</td></tr>
<tr><td colspan="2">[1] The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information <i>(see page 69)</i>.</td></tr>
</table>

EtherNet/IP Connection Properties

Edit connection with advanced parameters view:



New connection

● generic connection (free configurable)
○ predefined connection (EDS-File)

Connection Path Settings

○ generate path automatically

□ Configuration Assembly

Class ID 16# 4 Instance ID 16# 5 Attribute ID 16# 3

Consuming Assembly (0→T)

Class ID 16# 4 Instance ID 16# 14 Attribute ID 16# 3

Producing Assembly (T→O)

Class ID 16# 4 Instance ID 16# 46 Attribute ID 16# 3

● user-defined path

Generic Parameters

Connection Name	CIP Basic Control
Transport Type	Exclusive Owner
Connection Path	20 04 24 05 2C 14 2C 46
Timeout Multiplier	4

Scanner to Target (Output)

O→T Size (Bytes)	4
RPI (ms)	10
Trigger Type	Cyclic
Inhibit Time (ms)	0
Config #1 Size (Bytes)	0
Config #2 Size (Bytes)	0
Connection Type	Point to Point
Fixed/Variable	Fixed
Transfer Format	32 Bit Run/Idle

Target to Scanner (Input)

T→O Size (Bytes)	0
RPI (ms)	10
Trigger Type	Cyclic
Inhibit Time (ms)	0
Fallback Mode	Go to zero <default>
Connection Type	Point to Point
Fixed/Variable	Fixed
Transfer Format	pure Data

Connection settings:

Parameter			Values	Description
Generate path automatically			Yes/No	Enables you to configure the parameters of the assemblies.
	Configuration assembly		True/False	Enables you to configure a configuration assembly *(see page 62)*.
		Class ID	2 bytes (04h by default)	Class identifier[1]
		Instance ID	2 bytes (0 by default)	Instance identifier[1]
		Attribute ID	2 bytes (03h by default)	Attribute identifier[1]
	Consuming Assembly (O --> T)			
		Class ID	2 bytes (04h by default)	Class identifier[1]
		Instance ID	2 bytes (0 by default)	Instance identifier[1]
		Attribute ID	2 bytes (03h by default)	Attribute identifier[1]
	Producing Assembly (T --> O)			
		Class ID	2 bytes (04h by default)	Class identifier[1]
		Instance ID	2 bytes (0 by default)	Instance identifier[1]
		Attribute ID	2 bytes (03h by default)	Attribute identifier[1]
User-defined path			Yes/No	Disable the **Generate path automatically** area and enable the **Connection Path** field

[1] The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information *(see page 69)*.

Generic Parameters:

Parameter	Values	Description
Connection Path	Array of bytes	Coded transcription of the physical link object.
Transport Type	● Exclusive Owner (default) ● Listen Only ● Input Only	**Exclusive Owner**: This is a bidirectional connection to an Output connection point (typically an Assembly Object), where the data of this assembly can only be controlled by one Scanner. There may be a connection to an input assembly; this data is being sent to the scanner. If the input data length is zero, then this direction becomes a Heartbeat connection. **Listen only**: The scanner receives input data from the target device and produces a Heartbeat to the target device. There is no Output data. A Listen Only Connection can only be attached to an existing **Exclusive Owner** or Input Only Connection. If this underlying connection stops, then the **Listen Only** connection is also stopped or timed out. **Input Only**: The scanner receives input data from the target device and produces a Heartbeat to the target device. There is no Output data.
Timeout Multiplier	4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512	Scanner Timeout *(see page 35)* is managed connection by connection with RPI and timeout multiplier.

Scanner to Target (Output):

Parameter	Values	Description
O --> T Size (Bytes)	0 to XX => device specific	Size of channel for an assembly. The memory size of each channel is 2 bytes that stores the value of `%IW`x or `%QW`x object, where x is the channel number.

NOTE: If transfer format is set to **32 bit Run-idle**, the scanner status is sent in the request. Targets may not respond in the same manner when they receive the information that the scanner is in IDLE status. For example, some targets may not update their inputs while others do when the controller is `STOPPED` or `HALT`.

Parameter	Values	Description
RPI (ms)	In ms (10 ms by default)	Requested Packet Interval. The time period between cyclic data transmissions requested by the scanner. The device always provides a minimum RPI, whereas in the controller the goal is to have the highest RPI in order to not overload the system. Each time a device is added to the EtherNet/IP fieldbus, or each time a RPI value is modified, it is recommended to check the resources (refer to the scanner resource checker *(see page 78)*). The device RPI may be specified in the device documentation. Usually, however, this information is provided as part as the EDS file *(see page 37)* delivered with the device.
Trigger Type	Cyclic	**Cyclic:** Endpoints send their messages at pre-determined cyclic time intervals
Inhibit Time	0 ms	Minimum period time between 2 data exchanges.
Config#1 Size (Bytes)	0 to XX => device specific	Accessible if connection path contains a configuration assembly. Numbers of parameter (1 byte) to transfer. The configuration values are sent to the device at the scanner start.
Config#2 Size (Bytes)	0 to XX => device specific	
Connection Type	Point to Point	Connection type of the request
Fixed/Variable	Fixed	The request length is fixed.
Transfer format	● 32 bit Run-idle (by default) ● pure Data ● Heartbeat	Transfer format of the request. For more information, refer to *ODVA website*.

NOTE: If transfer format is set to **32 bit Run-idle**, the scanner status is sent in the request. Targets may not respond in the same manner when they receive the information that the scanner is in IDLE status. For example, some targets may not update their inputs while others do when the controller is `STOPPED` or `HALT`.

Target to Scanner (Input):

Parameter	Values	Description
T --> O Size (Bytes)	0 to XX => device specific	Size of channel of an assembly. The memory size of each channel is 2 bytes that stores the value of `%IW`x or `%QW`x object, where x is the channel number.
RPI (ms)	In ms (10 ms by default)	Requested Packet Interval. The time period between cyclic data transmissions requested by the scanner. The device is always providing a minimum RPI, whereas in the controller the goal is to have the highest RPI in order to not overload the system. Each time a device is added to the EtherNet/IP fieldbus, or each time a RPI value is modified, it is recommended to check the resources (refer to the scanner resource checker *(see page 78)*). The device RPI may be specified in the device documentation. Usually, however, this information is provided as part as the EDS file *(see page 37)* delivered with the device.
Trigger Type	● Cyclic (default) ● Change of state	**Cyclic:** Endpoints send their messages at pre-determined cyclic time intervals **Change of state:** Change of state endpoints send their messages when a change occurs. The data is also sent at a background cyclic interval (RPI) if no change occurs to keep the connection from timing out.
Inhibit Time (ms)	In multiples of 2 ms (2 ms by default)	Minimum period time between 2 data exchanges. Accessible if **Trigger Type** is **Change of state**. Inhibit Time maximum value is RPI and is limited to 254 ms.
Fallback Mode	Go to zero <default>	Reset the input on error/stop
Connection Type	● Multicast (default) ● Point to Point	Connection type of the request
Fixed/Variable	Fixed	The request length is fixed.
Transfer format	● pure Data (by default) ● Heartbeat	Transfer format of the request. For more information, refer to *ODVA website*.

How to Find Assembly Information

Assembly information is provided in the device documentation. It is usually part of the description of assembly objects.

To configure an assembly, identify the following items of information:

1. Class ID
   The "Assembly object" Class ID is equal to 4.
2. Instance ID
   Select the assembly instance, depending on the application and on the type of device. The selection of the assembly instance will induce a dedicated state machine in the device:
   - **Configuration assembly**: Supported by few devices; verify in the device documentation which assembly instance is supported.
   - **Consuming assembly**: sometimes referred to as “device output” in the device documentation (from the device point of view).
   - **Producing assembly**: sometimes referred to as “device input“ in the device documentation (from the device point of view).
3. Attribute ID
   Search for the attribute to read. This corresponds to the data buffer exchanged during the connection.
   The attribute property must have write access for the producing assembly and read access for the consuming assembly.
   The attribute ID is the same for the two assemblies and equal to 3. It matches an attribute whose access is `Get/Set`. The name is often "data", and the type of data “Array of byte”.

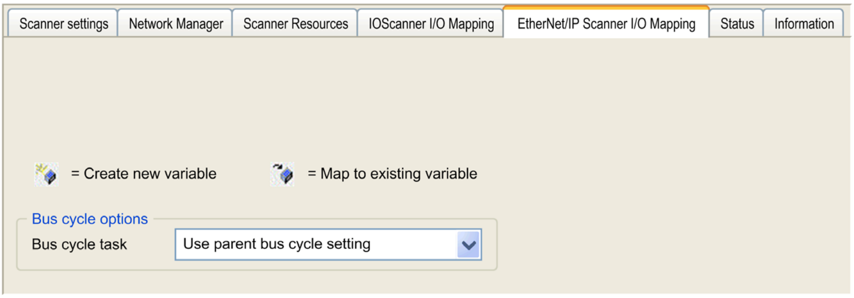
EtherNet/IP I/O Mapping

Overview

Once the data exchanges are configured, you can map variables to be used by the program.

Configure the EtherNet/IP Scanner I/O Mapping

To configure the EtherNet/IP Scanner I/O Mapping, proceed as follows:

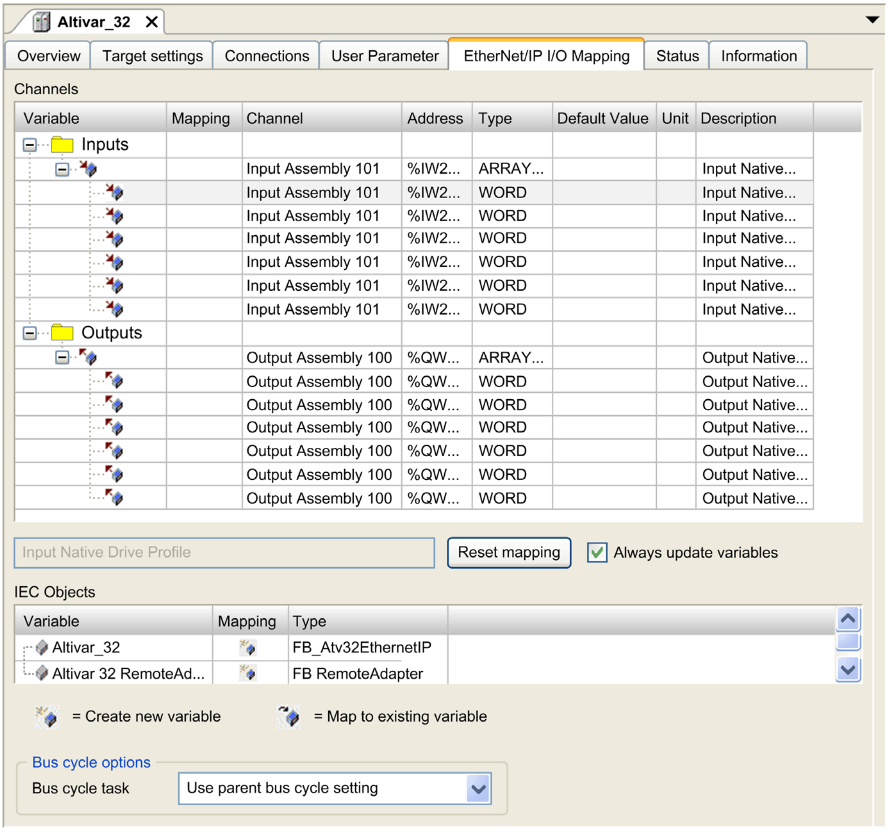
Step	Action
1	In the **Devices tree**, double-click **Industrial_Ethernet_manager**. **Result**: The configuration window is displayed.
2	Select the **EtherNet/IP Scanner I/O Mapping** tab.  Scanner settings Network Manager Scanner Resources IOScanner I/O Mapping EtherNet/IP Scanner I/O Mapping Status Information = Create new variable = Map to existing variable Bus cycle options Bus cycle task Use parent bus cycle setting
3	Select the **Bus cycle task** in the list: ● **Use parent bus cycle setting** (by default), ● **MAST**, ● An existing task of the project. **NOTE:** The **Bus cycle task** parameter inside the I/O mapping editor of the device that contains the Industrial Ethernet manager defines the task responsible for the refresh of the I/O images (%QW, %IW). These I/O images correspond to the EtherNet/IP request sent to the EtherNet/IP target devices and the health bits.

NOTE: When the Industrial Ethernet manager is configured, the post configuration file for the device network is ignored.

Configure an EtherNet/IP Target Device I/O Mapping

Once the data exchanges are configured in predefined or new connections, you can map variables to be used by the program.

To configure an EtherNet/IP target device I/O mapping, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click an EtherNet/IP target device. **Result**: Its configuration window is displayed.
2	Select the **EtherNet/IP I/O Mapping** tab. 

Altivar_32

Overview | Target settings | Connections | User Parameter | EtherNet/IP I/O Mapping | Status | Information

Channels

Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
Inputs							
		Input Assembly 101	%IW2...	ARRAY...			Input Native...
		Input Assembly 101	%IW2...	WORD			Input Native...
		Input Assembly 101	%IW2...	WORD			Input Native...
		Input Assembly 101	%IW2...	WORD			Input Native...
		Input Assembly 101	%IW2...	WORD			Input Native...
		Input Assembly 101	%IW2...	WORD			Input Native...
		Input Assembly 101	%IW2...	WORD			Input Native...
Outputs							
		Output Assembly 100	%QW...	ARRAY...			Output Native...
		Output Assembly 100	%QW...	WORD			Output Native...
		Output Assembly 100	%QW...	WORD			Output Native...
		Output Assembly 100	%QW...	WORD			Output Native...
		Output Assembly 100	%QW...	WORD			Output Native...
		Output Assembly 100	%QW...	WORD			Output Native...
		Output Assembly 100	%QW...	WORD			Output Native...

Input Native Drive Profile

Reset mapping ☑ Always update variables

IEC Objects

Variable	Mapping	Type
Altivar_32		FB_Atv32EthernetIP
Altivar 32 RemoteAd...		FB RemoteAdapter

= Create new variable = Map to existing variable

Bus cycle options

Bus cycle task: Use parent bus cycle setting

Step	Action
3	Select the **Bus cycle task** in the list: ● **Use parent bus cycle setting** (by default), ● **MAST**, ● An existing task of the project. **NOTE:** The **Bus cycle task** parameter inside the I/O mapping editor of the device that contains the Industrial Ethernet manager defines the task responsible for the refresh of the I/O images (%QW, %IW). These I/O images correspond to the EtherNet/IP request sent to the EtherNet/IP target devices and the health bits.
4	Double-click in a cell of the **Variable** column to open a text field. Enter the name of a variable or click the browse button **[...]** and chose a variable with the **Input Assistant**.

Modbus TCP Cyclic Data Exchanges Configuration

Overview

To configure the Modbus TCP cyclic data exchanges, you have to:

- configure for each Modbus TCP slave devices the data exchanges request (on channels) and the I/O Mapping.
- configure the I/O scanner for Modbus TCP slave devices.

Modbus TCP Channel

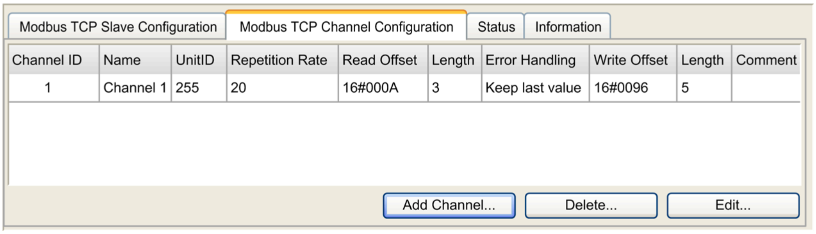
A Modbus channel carries a Modbus request between the master and a slave.

Advantys OTB and predefined slave devices use one channel per device. This channel is configured using SoMachine software.

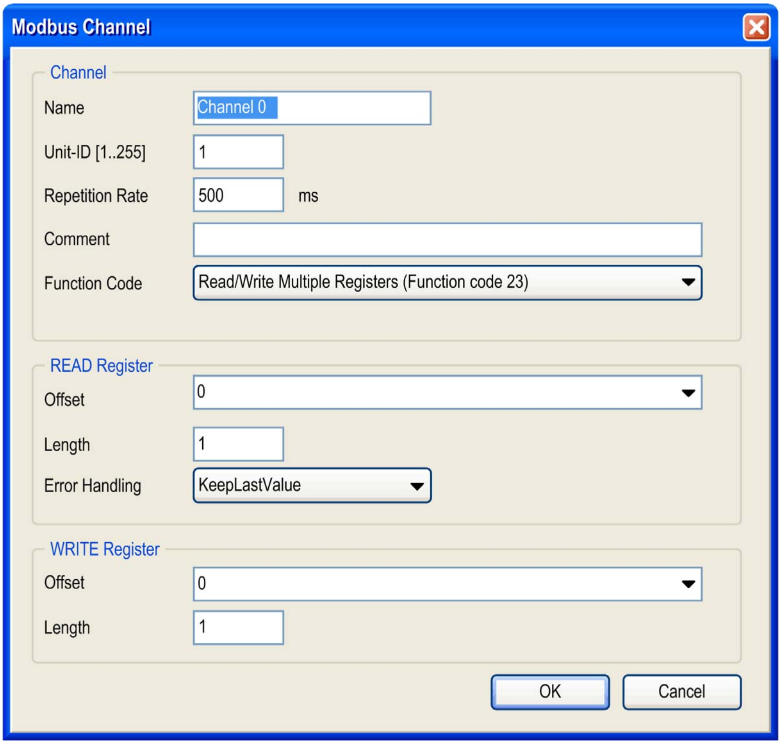
For a generic slave device, you can use multiple channels. To send several different requests to a device, create several channels.

Configure the Modbus TCP Slave Device Channels

To configure the data exchanges (on channels) of a Modbus TCP slave device, proceed as follows:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>In the <b>Devices tree</b>, double-click a Modbus TCP slave device.
<b>Result</b>: Its configuration window is displayed.</td></tr>
<tr><td>2</td><td>Click the <b>Modbus TCP Channel Configuration</b> tab:


Modbus TCP Slave Configuration
Modbus TCP Channel Configuration
Status
Information
Add Channel...
Delete...
Edit...
<table>
<tr><th>Channel ID</th><th>Name</th><th>UnitID</th><th>Repetition Rate</th><th>Read Offset</th><th>Length</th><th>Error Handling</th><th>Write Offset</th><th>Length</th><th>Comment</th></tr>
<tr><td>1</td><td>Channel 1</td><td>255</td><td>20</td><td>16#000A</td><td>3</td><td>Keep last value</td><td>16#0096</td><td>5</td><td></td></tr>
</table></td></tr>
<tr><td>3</td><td>To remove a non-predefined channel, select it and click <b>Delete</b>.</td></tr>
<tr><td>4</td><td>To change the parameters of a channel, select the channel and click <b>Edit</b>.
<b>NOTE:</b> For the devices that provide predefined channels, only the <b>Repetition Rate</b> value can be modified.</td></tr>
</table>

Step	Action
5	To add a channel, click **Add Channel**. This dialog box is displayed:  Modbus Channel Channel Name: Channel 0 Unit-ID [1..255]: 1 Repetition Rate: 500 ms Comment: Function Code: Read/Write Multiple Registers (Function code 23) READ Register Offset: 0 Length: 1 Error Handling: KeepLastValue WRITE Register Offset: 0 Length: 1

Step	Action
6	In the **Channel** area, you can define: ● **Name**: optional string for naming the channel. ● **Unit-ID [1..255]**: unit ID (1) of the Modbus TCP slave device (by default 255). ● **Repetition Rate**: polling interval of the Modbus request (by default 20 ms). ● **Comment**: optional field to describe the channel. ● **Function Code**: type of Modbus request: ❍ **Read/Write Multiple Registers (Function code 23)** (by default). ❍ **Read Holding Registers (Function code 03)**. ❍ **Write Multiple Registers (Function code 16)**. In the **READ register** area, you can define: ● **Offset**: starting register number to read from 0 to 65535. ● **Length**: number of the registers to be read (depending on the function code). ● **Error Handling**: define the fallback value in the case of a communication interruption: ❍ **Keep Last Value** (by default) holds the last valid value. ❍ **SetToZero** resets the values to 0. In the **WRITE register** area, you can define: ● **Offset**: starting register number to write from 0 to 65535. ● **Length**: number of the registers to be written (depending on the function code).
7	Click **OK** to validate the configuration of the channel.
8	Repeat steps 5 to 7 to create other channels that define the Modbus communication with the device. For each Modbus request, you must create a channel.

(1) Unit identifier is used with Modbus TCP devices which are composed of several Modbus devices, for example, on Modbus TCP to Modbus RTU gateways. In such case, the unit identifier allows reaching the slave address of the device behind the gateway. By default, Modbus/TCP-capable devices ignore the unit identifier parameter.

Read/Write Register Length

The read/write register length depends on the Modbus function code.

This table contains, for 1 channel, the maximum length of the read/write registers:

Modbus function code	Maximum length	
	Read register	Write register
Read/write multiple registers (function code 23)	125	121
Read registers (function code 03)	125	-
Write registers (function code 16)	-	123

NOTE: Due to these limitations and the maximum input/output words of the scanner (2048), verify the scanner resources overload *(see page 78)*.

Modbus TCP I/O Mapping

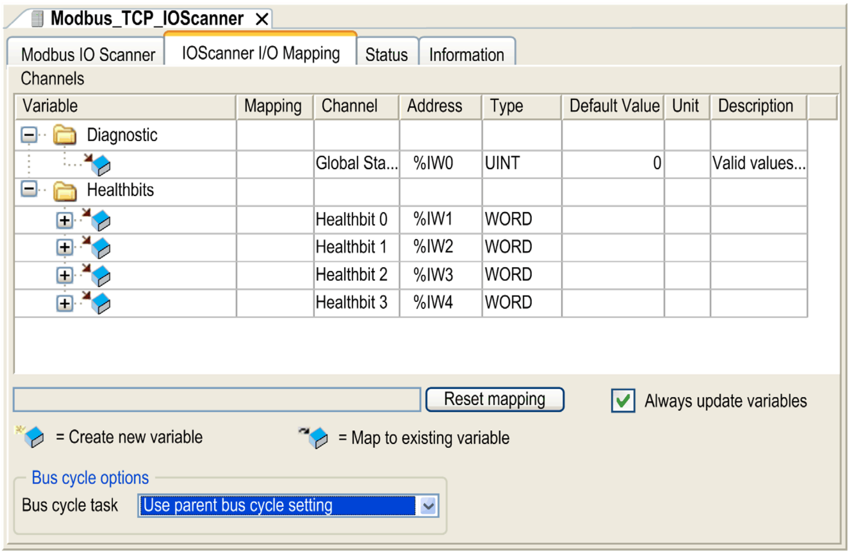
Prerequisites

A Modbus TCP channel must exist.

Configure the Modbus TCP IOScanner

To configure a Modbus TCP IOScanner, proceed as follows:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>In the <b>Devices tree</b>, double-click <b>Industrial_Ethernet_manager</b>.
<b>Result</b>: The configuration window is displayed.</td></tr>
<tr><td>2</td><td>Select the <b>IOScanner I/O Mapping</b> tab:



Channels

<table>
<tr><th>Variable</th><th>Mapping</th><th>Channel</th><th>Address</th><th>Type</th><th>Default Value</th><th>Unit</th><th>Description</th></tr>
<tr><td>Diagnostic</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>Global Sta...</td><td>%IW0</td><td>UINT</td><td>0</td><td></td><td>Valid values...</td></tr>
<tr><td>Healthbits</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>Healthbit 0</td><td>%IW1</td><td>WORD</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>Healthbit 1</td><td>%IW2</td><td>WORD</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>Healthbit 2</td><td>%IW3</td><td>WORD</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>Healthbit 3</td><td>%IW4</td><td>WORD</td><td></td><td></td><td></td></tr>
</table>

☑ Always update variables

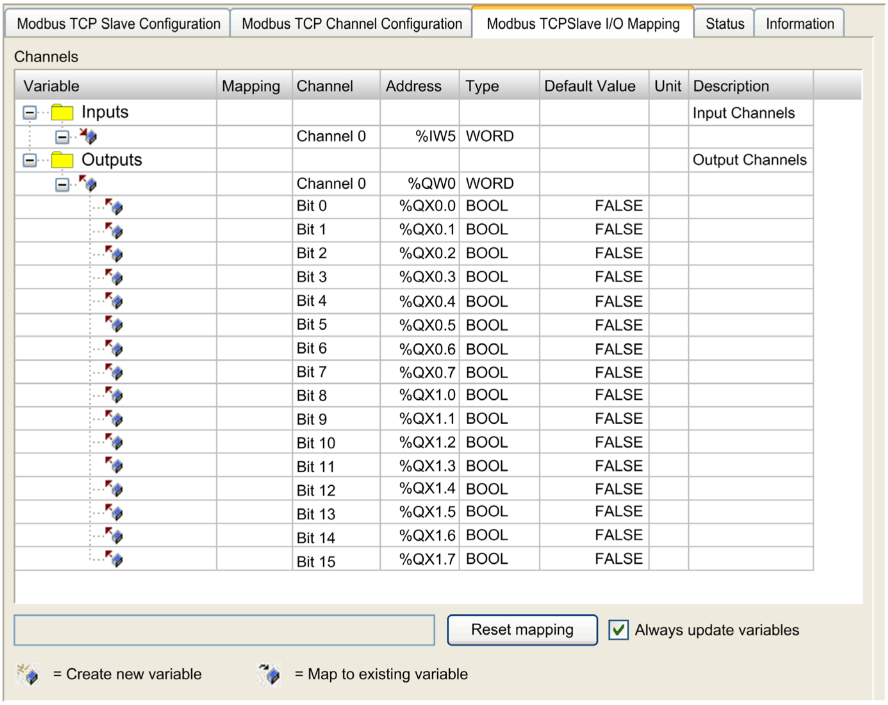
Bus cycle task: Use parent bus cycle setting</td></tr>
<tr><td>3</td><td>Select the <b>Bus cycle task</b> in the list:
● <b>Use parent bus cycle setting</b> (by default),
● <b>MAST</b>,
● An existing task of the project.

<b>NOTE:</b> The <b>Bus cycle task</b> parameter inside the I/O mapping editor of the device that contains the Modbus TCP IOScanner defines the task responsible for the refresh of the I/O images (%QW, %IW). These I/O images correspond to the Modbus request sent to the Modbus slaves and the health bits.</td></tr>
<tr><td>4</td><td>Double-click in a cell of the <b>Variable</b> column to open a text field.
Enter the name of a variable or click the browse button <b>[...]</b> and chose a variable with the <b>Input Assistant</b>.</td></tr>
</table>

Configure a Modbus TCP Slave Device I/O Mapping

To configure a Modbus TCP slave device I/O Mapping, proceed as follows:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>In the <b>Devices tree</b>, double-click a Modbus TCP slave device.
<b>Result</b>: Its configuration window is displayed.</td></tr>
<tr><td>2</td><td>Select the <b>ModbusTCPSlave I/O Mapping</b> tab.



Modbus TCP Slave Configuration | Modbus TCP Channel Configuration | Modbus TCPSlave I/O Mapping | Status | Information

Channels

<table>
<tr><th>Variable</th><th>Mapping</th><th>Channel</th><th>Address</th><th>Type</th><th>Default Value</th><th>Unit</th><th>Description</th></tr>
<tr><td>⊟ Inputs</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Input Channels</td></tr>
<tr><td>⊟</td><td></td><td>Channel 0</td><td>%IW5</td><td>WORD</td><td></td><td></td><td></td></tr>
<tr><td>⊟ Outputs</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Output Channels</td></tr>
<tr><td>⊟</td><td></td><td>Channel 0</td><td>%QW0</td><td>WORD</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 0</td><td>%QX0.0</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 1</td><td>%QX0.1</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 2</td><td>%QX0.2</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 3</td><td>%QX0.3</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 4</td><td>%QX0.4</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 5</td><td>%QX0.5</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 6</td><td>%QX0.6</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 7</td><td>%QX0.7</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 8</td><td>%QX1.0</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 9</td><td>%QX1.1</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 10</td><td>%QX1.2</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 11</td><td>%QX1.3</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 12</td><td>%QX1.4</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 13</td><td>%QX1.5</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 14</td><td>%QX1.6</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
<tr><td></td><td></td><td>Bit 15</td><td>%QX1.7</td><td>BOOL</td><td>FALSE</td><td></td><td></td></tr>
</table>

</td></tr>
<tr><td>3</td><td>Select the <b>Bus cycle task</b> in the list:
● <b>Use parent bus cycle setting</b> (by default),
● <b>MAST</b>,
● An existing task of the project.

<b>NOTE:</b> The <b>Bus cycle task</b> parameter inside the I/O mapping editor of the device that contains the Modbus TCP IOScanner defines the task responsible for the refresh of the I/O images (%QW, %IW). These I/O images correspond to the Modbus request sent to the Modbus slaves and the health bits.</td></tr>
<tr><td>4</td><td>Double-click in a cell of the <b>Variable</b> column to open a text field.
Enter the name of a variable or click the browse button <b>[...]</b> and chose a variable with the <b>Input Assistant</b>.</td></tr>
</table>

Industrial Ethernet Manager Load Verification

Purpose

If the load on the **Industrial_Ethernet_manager** exceeds 100%, cyclic data exchanges might not be processed at the configured rate.

The **Scanner Resources** tab allows you to estimate the load on the **Industrial_Ethernet_manager**.

Verify this load before operating the machine.

To manage the load, you can manipulate one or more of the following load factors:

- Number of slaves
- With EtherNet/IP:
  - Number of connections (on the EtherNet/IP Scanner)
  - The RPI of the connections
- With Modbus TCP:
  - Number of channels (on the Modbus TCP IOScanner)
  - The repetition rate of the channels

Load Estimation

This equation allows estimation of the load on the **Industrial_Ethernet_manager** if it manages exclusively Modbus TCP IOScanner devices:

$$\text{IOScanner load (\%)} = \sum_{\text{channel}=1}^{\text{Nb Channels}} \frac{50}{\text{Repetition Rate}_{\text{channel}}}$$

This equation allows estimation of the load on the **Industrial_Ethernet_manager** if it manages at least one Ethernet/IP device:

$$\text{Scanner load (\%)} = \sum_{\text{channel}=1}^{\text{Nb channels}} \frac{200}{\text{Repetition Rate}_{\text{channel}}} + \sum_{\text{connection}=1}^{\text{Nb connection}} \frac{\text{load}}{\text{RPI}_{\text{connection}}}$$

$$\text{if RPI}_{\text{connection}} < 5 \text{ then load} = 100\text{, else load} = 62.5$$

NOTE:
This load estimate does not take into account increases in load resulting from out of process data exchanges *(see page 98)* such as:

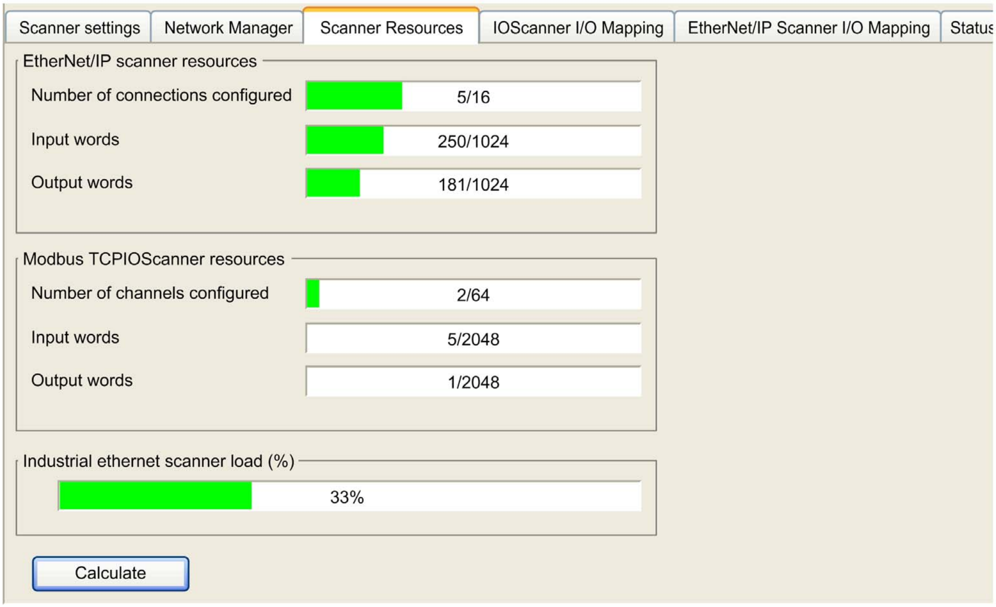
- DTM, Web server, and Modbus TCP requests.
- fieldbus communications (DTM, Web server communications when the PC is on the fieldbus)
- TCP UDP communications generated by the TcpUdpCommunications library.

In SoMachine, an automatic load calculation is available:

Step	Action
1	In the **Devices tree**, double-click the **Industrial_Ethernet_manager** node.
2	Select the **Scanner Resources** tab.
3	Click **Calculate**.

Description

This picture presents the **Scanner Resources** tab:



EtherNet/IP scanner resources

Number of connections configured	5/16
Input words	250/1024
Output words	181/1024

Modbus TCPIOScanner resources

Number of channels configured	2/64
Input words	5/2048
Output words	1/2048

Industrial ethernet scanner load (%)

33%

Section 2.8
Programming Over Industrial Ethernet

Programming Over Industrial Ethernet

Overview

When the **Industrial_Ethernet_manager** is added, the following libraries are automatically instantiated:

- Modbus TCP IOScanner
- EtherNet/IP Scanner

In addition, most Industrial Ethernet slave devices have a dedicated library containing function and function blocks.

Use these elements to facilitate the program writing.

SoMachine contains TVDA templates that can be used.

Manage the Operating Modes of the Devices

The Modbus TCP IOScanner library contains these functions:

- IOS_GETSTATE: Read the state of the Modbus TCP IOScanner
- IOS_START: Start the Modbus TCP IOScanner
- IOS_GETHEALTH: Read the Health Bit value
- IOS_STOP: Stop the Modbus TCP IOScanner
- CONFIGURE_OTB: Send the software configuration of the Advantys OTB

For more details, refer to Modbus TCP IOScanner Library *(see page 123)*.

The EtherNet/IP Scanner library contains these functions:

- EipControl: Start/Stop the EtherNet/IP scanner
- EipGetHealth: Read the Health Bit value

For more details, refer to EtherNetIP Scanner Library *(see page 169)*.

For operational details, refer to Mastering slave devices operating modes *(see page 92)* and Impact of the logic controller States on the Industrial Ethernet *(see page 100)*.

Send Commands and Read Status from Devices

Cyclic data exchanges are used with generic devices that require deterministic data exchanges. Cyclic data exchanges are managed by the Industrial Ethernet manager. To configure cyclic data exchanges, see EtherNet/IP Cyclic Data Exchanges Configuration *(see page 57)*. To use cyclic data in your program, see EtherNet/IP I/O Mapping *(see page 70)*.

In addition, you can send explicit messages.

For EtherNet/IP devices, with the EtherNet/IP Explicit Messaging library, you can use:

- `Get_Attribute_All`, refer to Get_Attribute_All,Get All Attributes of an Object *(see page 139)*
- `Set_Attribute_All`, refer to Set_Attribute_All, Set All Attributes of an Instance or Class *(see page 142)*
- `Get_Attribute_Single`, refer to Get_Attribute_Single, Get an Attribute of an Object *(see page 145)*
- `Set_Attribute_Single`, refer to Set_Attribute_SIngle, Set an Attribute of an Object *(see page 148)*

For EtherNet/IP devices, with the EtherNet/IP Scanner library, you can use `EipDataExch` for features not implemented in the EtherNet/IP Explicit messaging Library *(see page 174)*.

For Modbus TCP devices, you can use READ_VAR and WRITE_VAR.

For operational details, refer to Slave devices configuration on start *(see page 97)* and Data exchanges on demand *(see page 95)*.

Use TVDA Templates

Most Industrial Ethernet slave devices are parts of TVDA.

SoMachine proposes to add a device from a template *(see page 37)*.

By this, the device is added with several already parametrized blocks and/or function blocks.

Chapter 3
Device Network Commissioning

Overview

This chapter describes how to perform the commissioning of your Industrial Ethernet network.

This phase follows the device network configuration *(see page 27)*.

At the end of this phase, the application can be started *(see page 91)*.

What Is in This Chapter?

This chapter contains the following topics:

Commissioning

Overview

During the commissioning, you have to:

- perform the machine (logic controller and slave devices) first power-up.
- perform network tests.
- download the configuration to the network devices.
- adjust logic controller and network devices configuration (online or directly on the devices).
- complete the FDR on each available device.
- back up your application.

First Machine Power-up

To realize the first power-up, proceed as follow:

Step	Action
1	Transfer the application to the logic controller. Refer to Downloading an Application *(see SoMachine, Programming Guide)*.
2	Prepare each device to be recognized on the device network by referring to the network planning *(see page 40)*: BOOTP, DHCP, fixed IP, network name. For details, refer to Prepare the Device to Be Recognized *(see page 86)*.
3	Perform a machine power cycle. This may be necessary for some devices to acquire the correct network settings.
4	Perform network tests *(see page 108)*.

Download the Configuration to the Network Devices

Refer to Apply the Correct Device Configuration *(see page 89)*.

Adjust Logic Controller and Devices Application

Once the first machine power-up performed and the configuration downloaded to the devices, you can adjust the system with:

- User Parameters online modification
- Embedded DTMs online modification, such as:
  - parameters adjustment,
  - autotuning for performances and energy efficiency,
  - oscilloscope for fine dynamic tuning
  - …
- For devices not having DTM, manual adjustment directly done on the devices. Refer to the documentation of the device.

Complete the FDR Service

Once the system is configured, you have to complete the FDR service. This step consists in saving the device configuration in the logic controller FTP server.

Depending on the device, several tools can be used:

- SoMachine,
- Third-party tools (for example: SoMove),
- Device Web server,
- Directly on the device (with embedded HMI),
- …

For more details, refer to the documentation of the device.

Back Up Application

Once the machine commissioning is completed, and before operation phase, upload and save project for further use.

Depending of the logic controller, several methods are available:

- SoMachine: Perform a backup of the application program to the hard disk of the PC.
- Logic controller Web server
- Logic controller clone function (with SD card).
- …

For more details, refer to the documentation of the device.

Prepare the Device to Be Recognized

Overview

The aim of this step is to configure the IP address assignment method of the device to be in accordance with that configured in the network manager *(see page 40)*.

This can be done during:
- the commissioning phase *(see page 83)*.
- a device replacement *(see page 119)*.

Depending on the device, different tools can be used:
- Screwdriver: for devices with rotary switch, dip switch, …(example: OTB)
- Keypad (example: ATV)
- PC, for devices that have to be configured with:
  - SoMachine
  - Third-party software
  - Its Web server (example: OsiSense XGCS)

Depending on the IP address assignment, different actions can be done:
- DHCP: configure the DHCP device name in the device.
- BOOTP: refer to Device Configured in BOOTP *(see page 87)*.
- Fixed IP: configure the IP address in the device.

If using EtherNet/IP, and using Electronic Keying *(see page 87)*, verify whether it is correctly configured.

Main Device Configuration Method

Tool	IP address assignment method	Description
None	DHCP	The device is pre-configured in DHCP with the correct DHCP device name
Screwdriver	DHCP	Use a screwdriver on the device (rotary switch, dip switch, …) to configure the DHCP device name. Example: Advantys OTB.
	BOOTP	Use a screwdriver on the device (rotary switch, dip switch, …) in BOOTP. Example: XPSMCM.
	Fixed IP	Use a screwdriver on the device (rotary switch, dip switch, …) to configure the IP address.
You may have to perform a device power cycle for parameter modifications to take affect.		

Tool	IP address assignment method	Description
Keypad	DHCP	Use the keypad of the device to configure DHCP device name. Example: ATV32.
	BOOTP	Use the keypad of the device to configure the device in BOOTP.
	Fixed IP	Use the keypad of the device to configure IP address.
PC, tablet, …	DHCP BOOTP Fixed IP	Use PC or tablet to connect to the **Device Web server** and configure the network settings. Choose a connection method: ● Connect the PC to an Ethernet port of the device The current IP address of the device must be known. ● Connect a WIFER TCSEGWB13FA0 to an Ethernet port of the device. Connect the PC to the WIFER.
PC	DHCP BOOTP Fixed IP	Use SoMachine (via DTM) to configure the network settings. Connect the PC to a dedicated communication port of the device. Example: Modbus serial line port of ATV32. For details, refer to Using DTMs to Configure Devices on Modbus Serial Line *(see SoMachine, Device Type Manager (DTM), User Guide)*.
	DHCP BOOTP Fixed IP	Use **third-party software** to configure the network settings. Choose a connection method: ● Connect the PC to an Ethernet port of the device The current IP address of the device must be known. ● Connect the PC to a dedicated communication port of the device.
You may have to perform a device power cycle for parameter modifications to take affect.		

Device Configured in BOOTP

If the device IP address assignment is BOOTP, you must use SoMachine:

- Set the MAC address of the new device in the network manager *(see page 40)*,
- Load the new application in the logic controller.

Electronic Keying with EtherNet/IP

Electronic Keying signatures are used to identify the device.

Electronic Keying is information about the network device contained within the firmware of the device (Vendor Code, Product Code, etc.).

When the scanner starts, it compares the electronic keying values of the network device with those stored in the application.

If the device values are not the same as the application values, the logic controller no longer communicates with the device.

During first commissioning and at device replacement, if the EtherNet/IP scanner verifies the electronic keying, you may use SoMachine to:

- verify, and modify if necessary, the values of the Electronic Keying *(see page 44)*,
- load the new application in the logic controller.

Apply the Correct Device Configuration

Overview

Once the device is recognized on the device network, you may have to configure it.

This can be done during:
- the commissioning phase *(see page 83)*.
- a device replacement *(see page 119)*.

Description

It may be necessary to perform different actions, depending on the device, to apply the correct device configuration. In addition, a power cycle of the device may also be required before any configuration information is taken into account by the device.

Action	Description
No manual modification	The device is provided pre-configured. Everything is automated.
	User parameters are sent to the device when the application starts. For more details, refer to Device Replacement with User Parameters *(see page 52)*.
	For Advantys OTB, the configuration download can be performed by program only. For more details, refer to Services Configuration on Start *(see page 97)*.
SD card, USB memory key, keypad …	Often, the media used to store the configuration is already prepared for operation. However, inserting the media into the new device may require some manual actions.
Multiloader	Use the multiloader tool to load a previously saved configuration file in the device.
FDR (through keypad menus)	In some cases, you must explicitly ask the device to get its configuration from the FDR server, and then switch the FDR service back to IDLE. For more details, refer to the documentation of the device. For FDR details, refer to Device Replacement with FDR *(see page 51)*.
FDR (through Web server)	Use an external tool such as a PC, smart phone, tablet, etc. that supports the use of a web browser to affect the device replacement. In some cases, you must explicitly ask the device to get its configuration from the FDR server, then switch the FDR service back to IDLE.
Device Web server (parameter by parameter)	Use an external tool such as a PC, smart phone, tablet, etc. that supports the use of a web browser to affect the configuration.
You may have to perform a device power cycle for parameter modifications to take affect.	

Action	Description
SoMachine	Use SoMachine to download the configuration to the device. For devices that support DTM, refer to Using DTMs to Configure Devices on Modbus TCP or EtherNet/IP *(see SoMachine, Device Type Manager (DTM), User Guide)*.
Third-party software	Use a third-party software.
You may have to perform a device power cycle for parameter modifications to take affect.	

For more information concerning the device configuration, refer to the documentation of the device.

Chapter 4
Device Network Operation

Overview

This chapter describes the functionalities, data exchange process, and security for operating modes.

What Is in This Chapter?

This chapter contains the following topics:

Managing Slave Devices Operating Modes

Overview

The operating modes of slave devices are managed by the Industrial Ethernet manager with the following scanners and their dedicated libraries:

- Modbus TCP IOScanner: Modbus TCP IOScanner library *(see page 123)*
- EtherNet/IP Scanner:
  - EtherNet/IP Scanner library *(see page 169)*
  - EtherNet/IP Explicit Messaging library *(see page 137)*

These libraries contain function blocks that allow you to:

- control the Modbus TCP IOScanner,
- control the EtherNet/IP Scanner,
- manage cyclic data exchanges (implicit messages),
- manage the status variables,
- send non-cyclic data exchange requests (explicit messages).

Other libraries can be used depending on the devices.

Status Variables of the Modbus TCP IOScanner

There are two status variable types:

- **Health bits**: variables to indicate the communication state of the channels. There is one health bit per channel.
- **Global scanner status**: variable to indicate the Modbus TCP IOScanner state.

This table presents the health bit values:

Health bit value	Communication state of the channel
0	Health timeout expired without receiving a reply.
1	No errors detected. Request and reply are received.

Status Variables of the EtherNet/IP Scanner

There are no pre-configured status variables of the EtherNet/IP Scanner.

To visualize the health bit of the EtherNet/IP targets, you must use:

- `EipGetHealth` function block *(see page 173)*
- `EIPGetHealthBit` function block *(see page 159)*

I/O Image Variables

The scanners collect and write data from/to the devices. These variables constitutes the I/O image.

Variables Addresses

Each variable gets its own address:

Variable	Type	Amount
I/O image variables	%IW for inputs %QW for outputs	A table of words is created per channel/connection.
Health bit	%IW	Four consecutive words for Modbus TCP N/A for EtherNet/IP
Global scanner status	%IW	One word for Modbus TCP N/A for EtherNet/IP
For EtherNet/IP, refer to Status Variables of the EtherNet/IP Scanner *(see page 92)*		

Function Blocks to Control the Modbus TCP IOScanner

Modbus TCP IOScanner library contains function blocks used by the application to communicate with the logic controller and the Modbus TCP slave devices:

- `CONFIGURE_OTB`: Send the software configuration of the Advantys OTB
- `IOS_GETSTATE`: Read the state of the Modbus TCP IOScanner
- `IOS_START`: Launch the Modbus TCP IOScanner
- `IOS_GETHEALTH`: Read the health bit value
- `IOS_STOP`: Stop the Modbus TCP IOScanner

For more details, refer to Modbus TCP IOScanner *(see page 123)*.

Function Blocks to Control the EtherNet/IP Scanner

EtherNet/IP Scanner library contains function blocks used by the application to communicate with the logic controller and the EtherNet/IP target devices:

- `EipDataExch`: Send an explicit message to a device
- `EipControl`: Start/Stop the connections of the EtherNet/IP Scanner
- `EipGetHealth`: Read the health bit value

For more details, refer to EtherNet/IP Scanner *(see page 169)*.

Function Blocks for EtherNet/IP Explicit Messaging

EtherNet/IP Explicit Messaging library contains function blocks used by the application to send EtherNet/IP Explicit Messages:

- `Get_Attribute_All`: Get All Attributes of an Object
- `Set_Attribute_All`: Set All Attributes of an Instance or Class
- `Get_Attribute_Single`: Get an Attribute of an Object
- `Set_Attribute_Single`: Set a Class Attribute
- `EIPStartConnection`: Start a Connection
- `EIPStartAllConnection`: Start All Connections
- `EIPStopConnection`: Stop a Connection

- `EIPStopAllConnections`: Stop All Connections
- `EipGetHealth`: Read the health bit value

For more details, refer to EtherNet/IP Explicit Messaging library *(see page 137)*.

Function Blocks to Control ATV and Lexium Devices

Use the PLC Open and other function blocks dedicated to drives to control ATV and Lexium devices. These function blocks can be accessed in the GMC Independent PLCopen MC library, GMC Independent Altivar library, and GMC Independent Lexium library. For more information, refer to the Motion Control Library Guide.

Bus Cycle Task

The Industrial Ethernet scanners and the slave devices exchange data at each cycle of an application task.

The **Bus Cycle Task** parameter allows you to select the application task that manages the scanner:

- **Use parent bus cycle setting**: associate the scanner with the application task that manages the controller.
- **MAST**: associate the scanner with the MAST task.
- Another existing task: you can select an existing task and associate it to the scanner.

For more information about the application tasks, refer to the SoMachine Programming Guide *(see SoMachine, Programming Guide)*.

Data Exchanges on Demand

Description

The Cyclic (implicit) data exchanges are managed by the chosen Industrial Ethernet scanner.

To perform data exchanges on demand, you must use explicit messages.

Explicit messages are initiated by the application with function blocks:

- For EtherNet/IP devices, you can use the function blocks of the EtherNet/IP Explicit Messaging library *(see page 137)*.
- For EtherNet/IP devices, you can also use the generic `EipDataExch` function block *(see page 174)* of the EtherNet/IP Scanner library.
- For Modbus TCP devices, you can use `READ_VAR` and `WRITE_VAR` function blocks.
- For TCP/UDP devices, you can use function blocks *(see page 185)*.

Custom Cyclic Data Exchanges

Description

When predefined devices are added in the project, cyclic data exchanges are created automatically.

Furthermore, you can create additional cyclic data exchanges on each slave device *(see page 55)*.

Slave Devices Configuration on Start

Description

To simplify device maintenance, you can send configuration data to slave devices.

At application start, you can send device configuration automatically by:

- User parameters *(see page 52)* when the application starts the connections.
- Configuration assembly *(see page 62)* (for devices that can support this function)

In addition, configuration of Advantys OTB devices can be sent on demand by the application using the CONFIGURE_OTB function block *(see page 129)*.

Out of Process Data Exchanges

Overview

Out of process data exchanges are often data exchanges between control network and device network. For example, you may use a supervision software or a third-party configuration tool to communicate with a target on the device network.

The Industrial Ethernet network permits out of process data exchanges.

To enable out of process data exchanges:

- Configure the gateway address in the devices *(see page 43)*.
- Ensure that the IP forwarding service is enabled.
- Check the PC routing (see below).

NOTE:
Out of process data exchanges originating from any of the following sources may impact the performance of the logic controller:

- DTM, Web server, and Modbus TCP requests.
- Network communications (DTM, Web server communications when the PC is on the network).
- TCP UDP communications generated by the TcpUdpCommunications library.

When connecting a DTM to a device using the network, the DTM communicates in parallel with the running application. The overall performance of the system is impacted and may overload the network, and therefore have consequences for the coherency of data across devices under control.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Do not connect DTMs that communicate across the device network on a running application if the DTM causes deleterious effect on performance.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

PC Routing

The PC supporting the supervision software or configuration tool must be configured to communicate with the slave devices. The PC must be in the same subnet as one of the Ethernet ports of the controller.

If the slave device is configured...	Then...
As a predefined slave through FDT/DTM	No specific PC parameterization is needed. **NOTE:** The PC configuration is not altered.
Using another tool	If the PC is not in the same subnet as the slave devices, you must update the routing table of the PC (see below).

To update the routing table of the PC, stop every connection from the PC to the controller and/or other devices. Then, in a Windows command prompt, execute the command:

```
route ADD destination MASK subnet_mask gateway
```

Where:

Parameter	Value
destination	IP address of the Industrial Ethernet network
subnet_mask	Subnet mask of the Industrial Ethernet network
gateway	IP address of the controller port connected to the control network

For example, for a TM251MESE, if:

- IP address of the PC: 192.168.0.2
- Subnet mask of the PC: 255.255.0.0
- IP address of the Industrial Ethernet network: 10.10.0.0
- Subnet mask of the Industrial Ethernet network: 255.255.252.0
- IP address of the control network port “Ethernet_1”: 192.168.0.5
- Subnet mask of the control network port “Ethernet_1”: 255.255.0.0

The corresponding command would be:

```
route ADD 10.10.0.0 MASK 255.255.252.0 192.168.0.5
```

To verify the parameters, execute the command:

```
route PRINT
```

To remove the route from the PC, execute the command:

```
route DELETE destination
```

Where *destination* is the IP address of the Industrial Ethernet network entered previously.

Industrial Ethernet Manager Operating Modes

Industrial Ethernet Manager States

To manage the operating modes of the devices, Industrial Ethernet manager is composed by:

- Modbus TCP IOScanner
- EtherNet/IP Scanner

The Industrial Ethernet manager state defines the behavior of the different devices in the device network. For each state, monitoring information (health bit, communication states, and so on) is specific.

The scanners states depend on the logic controller state:

Logic controller state	Modbus TCP IOScanner state	EtherNet/IP Scanner state
EMPTY	IDLE	IDLE
CONFIGURED	STOPPED	STOPPED
STOPPED	STOPPED	OPERATIONAL
HALT	STOPPED	OPERATIONAL with a specific behavior
RUNNING	OPERATIONAL	OPERATIONAL
RUNNING with breakpoint	OPERATIONAL with a specific behavior	OPERATIONAL with a specific behavior

Logic Controller EMPTY State

TCP/IP connections are closed.

Device states are managed according to their individual mode of operation.

The Modbus TCP IOScanner and the EtherNet/IP Scanner are not created (IDLE state). Therefore, health bits and I/O images are not available.

Logic Controller CONFIGURED State

TCP/IP connections are closed.

Logic controller enters in CONFIGURED state after:

- an application load.
- a reset (cold/warm) command sent by SoMachine.

The Modbus TCP IOScanner is in STOPPED state, all channels with the Modbus TCP slave devices are closed in half-sided mode.

The EtherNet/IP Scanner is in STOPPED state, all connections with the targets are closed.

Logic Controller STOPPED State

The Modbus TCP IOScanner is in STOPPED state. All channels with the Modbus TCP slave devices are closed in half-sided mode.

Slave devices are managed according to their individual mode of operation.

This table presents the SoMachine variables for Modbus TCP IOScanner:

Variable	Value	Comments
Health bit value	0	-
Input image	0 or the last read value	Input values depend on the **Error Handling** parameter. Input values are those when the logic controller entered in the STOPPED state and therefore may not reflect the actual state of the input thereafter.
Output image	0 or the last written value	Output values depend on the **Behavior for outputs in Stop** parameter. Output values may not reflect the actual state of the output thereafter.

The EtherNet/IP Scanner remains in OPERATIONAL state. All Originator/Target connections remain active. The data exchange between the targets and the scanner continues.

This table presents the SoMachine variables for EtherNet/IP Scanner:

Variable	Value	Comments
Input image	Read value	Values are refreshed synchronously with the task which drives the EtherNet/IP Scanner.
Output image	Last written value or default value	Outputs are set to their default value or maintained in their current value (depends on the **Behavior for outputs in Stop** parameter). Output values may not reflect the actual state of the output thereafter. Refer to transfer format of the connection *(see page 64)*.

⚠ WARNING

OUTPUT VALUES IN MEMORY MAY BE DIFFERENT THAN THEIR PHYSICAL STATE

Do not rely on the memory values for the state of the physical outputs when the controller is not in the RUNNING state.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Logic Controller HALT State

For Modbus TCP IOScanner, same behavior as the logic controller STOPPED state.

This table presents the SoMachine variables for EtherNet/IP Scanner if the task in HALT is the EtherNet/IP bus cycle task (by default the MAST):

Variable	Value	Comments
Input image	Last read value	Input values are those when the logic controller entered the HALT state and therefore may not reflect the actual state of the input thereafter.
Output image	Last written value or default value	Outputs are set to their default value or maintained in their current value (depends on the **Behavior for outputs in Stop** parameter). Output values may not reflect the actual state of the output thereafter.

This table presents the SoMachine variables for EtherNet/IP Scanner if the task in HALT is another task:

Variable	Value	Comments
Input image	Last read value	Values are refreshed synchronously with the task which drives the EtherNet/IP Scanner.
Output image	Last written value or default value	Outputs are set to their default value or maintained in their current value (depends on the **Behavior for outputs in Stop** parameter). Output values are overwritten each cycle. Output values may not reflect the actual state of the output thereafter. Refer to the transfer format of the connection *(see page 64)*. Online modifications on output are not available.

⚠ **WARNING**

OUTPUT VALUES IN MEMORY MAY BE DIFFERENT THAN THEIR PHYSICAL STATE

Do not rely on the memory values for the state of the physical outputs when the controller is not in the RUNNING state.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Logic Controller RUNNING State

TCP/IP connections are open.

Slave devices are managed by the logic controller.

This table presents the SoMachine variables:

Variable	Value	Comments
Health bit value	0...1	0: No reply from the device before the timeout expired. 1: Requests are sent and replied before the timeout expires.
Input image	Last read value	Values are refreshed synchronously with the task which drives the scanners.
Output image	Last written value	Values are managed by the application.

Logic Controller RUNNING with Breakpoint State

TCP/IP connections are open.

Slave devices are managed by the logic controller.

This table presents the SoMachine variables for EtherNet/IP Scanner if the task in RUNNING with breakpoint is the EtherNet/IP bus cycle task (by default the MAST):

Variable	Value	Comments
Input image	Last read value	Input values are those when the logic controller entered the RUNNING with breakpoint state and therefore may not reflect the actual state of the input thereafter.
Output image	Last written value or default value	Outputs are maintained in their current value. Output values may not reflect the actual state of the output thereafter.

This table presents the SoMachine variables for EtherNet/IP Scanner if the task in RUNNING with breakpoint is another task:

Variable	Value	Comments
Input image	Last read value	Input values are those when the logic controller entered the RUNNING with breakpoint state and therefore may not reflect the actual state of the input thereafter.
Output image	Last written value or default value	Outputs are maintained in their current value. Output values may not reflect the actual state of the output thereafter. Refer to the transfer format of the connection *(see page 64)*.

⚠ WARNING

OUTPUT VALUES IN MEMORY MAY BE DIFFERENT THAN THEIR PHYSICAL STATE

Do not rely on the memory values for the state of the physical outputs when the controller is not in the RUNNING state.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Security

Overview

Some specific features can increase the system security level for device replacement:

- Master IP address
- Electronic keying

Master IP Address Description

Some devices have a **Master IP address** parameter so that only one, declared Master, logic controller has access to the devices.

For more details, refer to Master IP Address Parameter *(see page 49)*.

Electronic Keying Description

Electronic Keying signatures are used to identify the device.

Electronic Keying is information about the network device contained within the firmware of the device (Vendor Code, Product Code, etc.).

When the scanner starts, it compares the electronic keying values of the device with those stored in the application.

If the device values are not the same as the application values, the logic controller no longer communicates with the device.

For more details, refer to Electronic Keying with EtherNet/IP *(see page 44)*.

Chapter 5
Device Network Diagnostics

Overview

This chapter contains troubleshooting information.

What Is in This Chapter?

This chapter contains the following topics:

Network Test

Purpose

Before operating the Industrial Ethernet manager, test the network.

Verify the following:

- The address configuration of each device conforms to the network planning.
- Each device is correctly wired.

Some standard testing methods are presented below.

Status LED

Depending on your devices, verify that the status LEDs display a correct wiring.

Verification Using a PC

With a PC, verify that each network device is connected and addressed:

Step	Action
1	Connect the PC in the Industrial Ethernet network.
2	Access the command prompt.
3	Use a `ping xxx.xxx.xxx.xxx` command to reach each network device, where.`xxx.xxx.xxx.xxx` is the IP address of the device to test. **NOTE:** The command `ping -h` displays the help for the `ping` command.

Verification Using a Web Server

With the logic controller Web server, verify that the logic controller can communicate with each network device:

Step	Action
1	Access the logic controller Web server.
2	Open the **Ethernet Diagnostic** page.
3	Use the **Remote ping** service on each device.

Diagnostics: Web Server

Overview

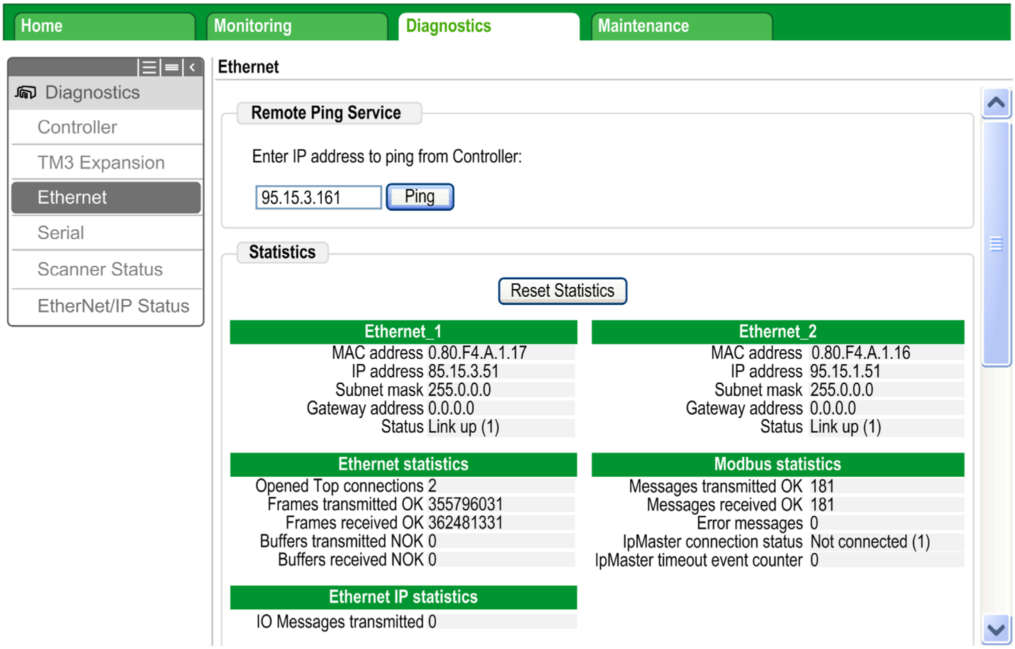
The Web server of the logic controller has a diagnostic tab.

In this tab, you can access to Industrial Ethernet diagnostic pages:

- **Ethernet** diagnostic page *(see page 109)*
- **Modbus TCP** diagnostic page *(see page 110)*
- **EtherNet/IP** diagnostic page *(see page 111)*

Ethernet Page

Click **Ethernet** to display Ethernet information of the logic controller and to allow you to test communication with a specific IP address:



Enter IP address to ping from Controller: 95.15.3.161

Ethernet_1	
MAC address	0.80.F4.A.1.17
IP address	85.15.3.51
Subnet mask	255.0.0.0
Gateway address	0.0.0.0
Status	Link up (1)

Ethernet_2	
MAC address	0.80.F4.A.1.16
IP address	95.15.1.51
Subnet mask	255.0.0.0
Gateway address	0.0.0.0
Status	Link up (1)

Ethernet statistics	
Opened Tcp connections	2
Frames transmitted OK	355796031
Frames received OK	362481331
Buffers transmitted NOK	0
Buffers received NOK	0

Modbus statistics	
Messages transmitted OK	181
Messages received OK	181
Error messages	0
IpMaster connection status	Not connected (1)
IpMaster timeout event counter	0

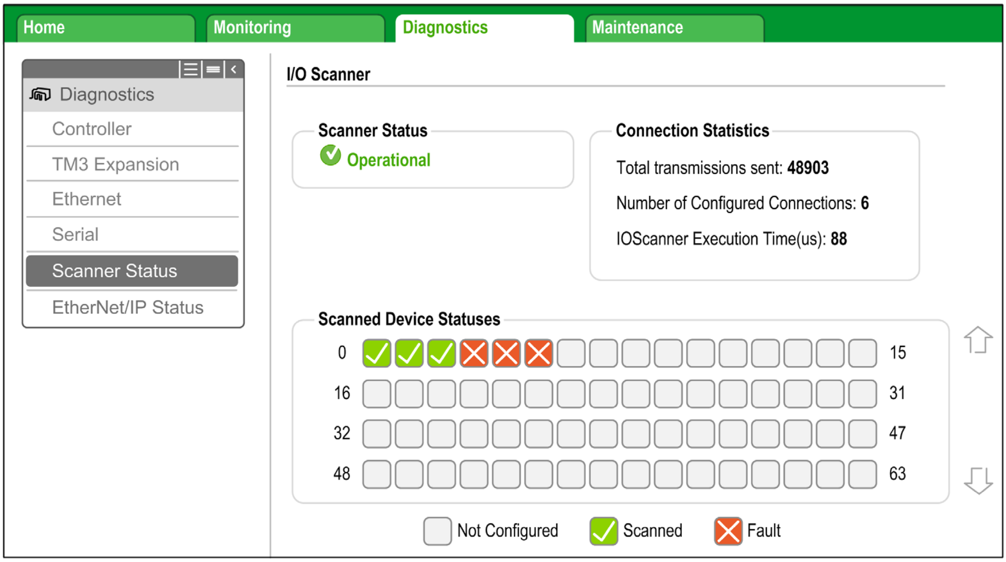
Ethernet IP statistics	
IO Messages transmitted	0

This table presents the ping test result on the **Ethernet** page:

Icon	Meaning
✓	The communication test is successful.
✗	The logic controller is unable to communicate with the defined IP address.

Modbus TCP Status Page

Click **Scanner Status** to display the Modbus TCP IOScanner status (IDLE, STOPPED, OPERATIONAL) and the health bit of up to 64 Modbus TCP slave devices:



I/O Scanner

Scanner Status

✓ Operational

Connection Statistics

Total transmissions sent: **48903**

Number of Configured Connections: **6**

IOScanner Execution Time(us): **88**

Scanned Device Statuses

0	✓	✓	✓	✗	✗	✗	□	□	□	□	□	□	□	□	□	□	15
16	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	31
32	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	47
48	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	63

□ Not Configured ✓ Scanned ✗ Fault

0…63 corresponds to the channel ID.

This table presents the status of each channel presented on the **Scanner Status** page:

Icon	Health bit value	Meaning	Scanner status
✓	1	Request and reply are ongoing on time.	OPERATIONAL
✗	0	An error is detected, the communications are closed.	OPERATIONAL

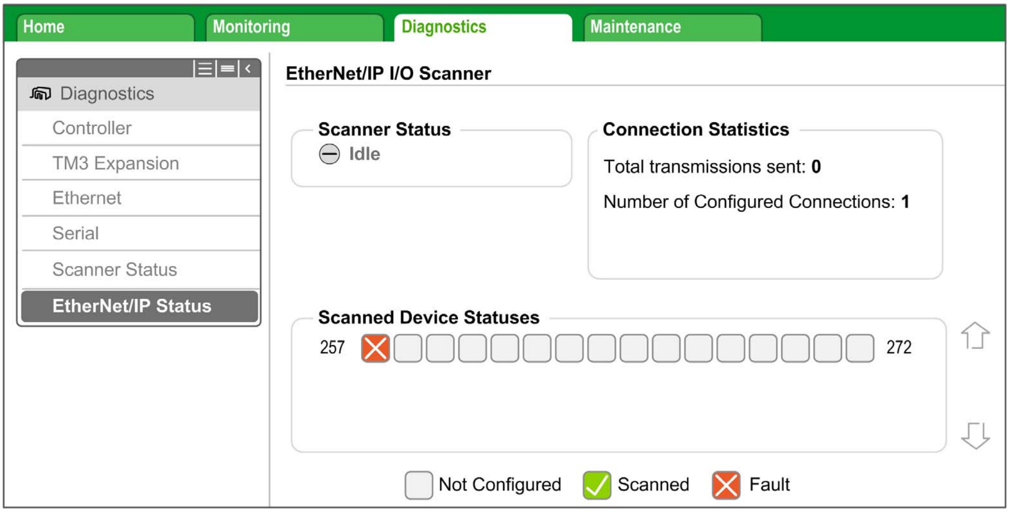
Icon	Health bit value	Meaning	Scanner status
□	–	This ID does not correspond to a configured channel.	OPERATIONAL
🚫	0	The communications are closed.	STOPPED

NOTE: Click any icon to open the device Web server (if existing). To access this Web server, the computer must be able to communicate with the device. For more information, refer to PC routing *(see page 98)*.

If the Modbus TCP IOScanner status is IDLE, no icon is displayed; **No scanned device reported** is displayed.

EtherNet/IP Status Page

Click **EtherNet/IP Status** to display the EtherNet/IP Scanner status (IDLE, STOPPED, OPERATIONAL) and the health bit of up to 16 EtherNet/IP target devices:



⊖ Idle

Total transmissions sent: **0**

Number of Configured Connections: **1**

257 ☒ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ 272

257…272 corresponds to the connection ID.

This table presents the status of each connection presented on the **EtherNet/IP Status** page:

Icon	Health bit value	Meaning	Scanner status
☑	1	Communications are ongoing on time.	STOPPED or OPERATIONAL.
☒	0	An error is detected, the communications are closed.	STOPPED or OPERATIONAL.
□	–	This ID does not correspond to a configured connection.	STOPPED or OPERATIONAL.

NOTE: Click any icon to open the network device Web server (if existing). To access this Web server, the computer must be able to communicate with the device. For more information, refer to PC routing *(see page 98)*.
If the EtherNet/IP Scanner status is IDLE, no icon is displayed; **No scanned device reported** is displayed.

Diagnostics: SoMachine Online Mode

Overview

In online mode, you can monitor the Industrial Ethernet manager in SoMachine using the following methods:

- Icons in the **Devices tree**
- Status tab of the Industrial Ethernet manager and the devices
- **IOScanner I/O Mapping** tab of the Industrial Ethernet manager for Modbus TCP IOScanner
- Visualization for health bit variables of the EtherNet/IP targets
- I/O mapping tab of the devices
- Industrial Ethernet manager resources tab

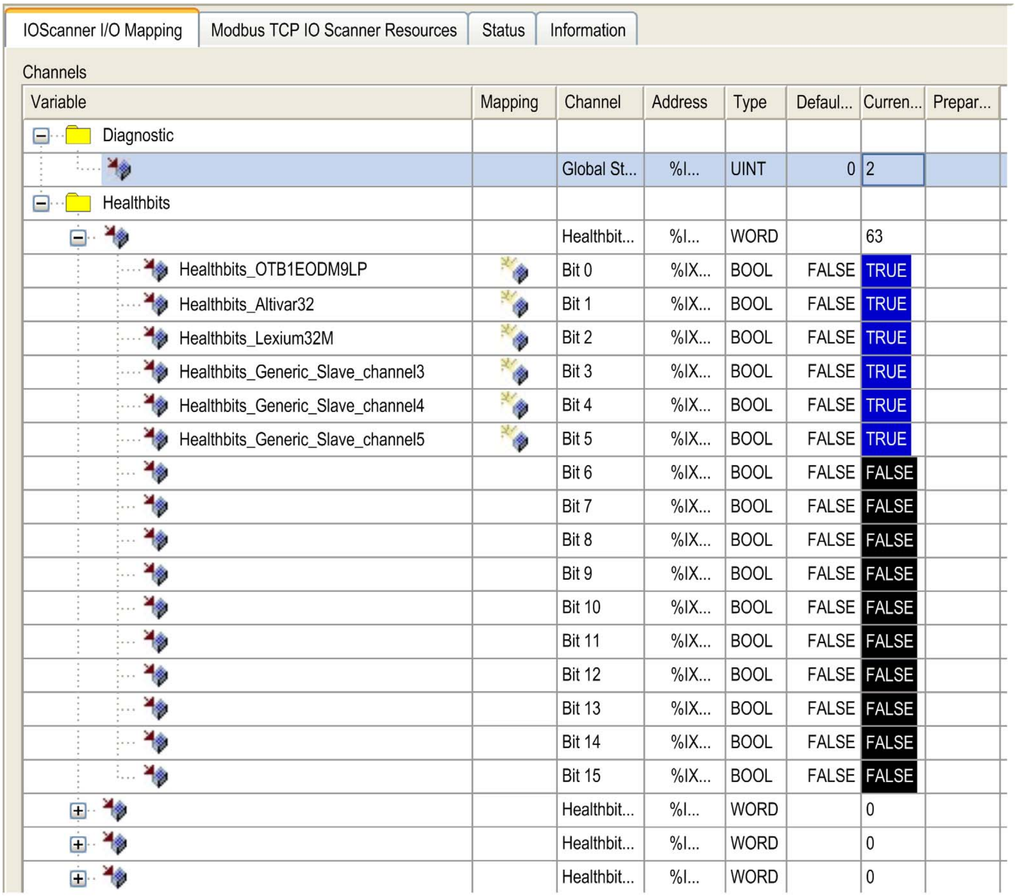
Devices Tree

The communication status of the Industrial Ethernet manager and the devices is presented with icons in the **Devices Tree**:

Icon	Meaning
🔄	The communication with the device is normal. **NOTE:** Industrial Ethernet manager is always presented with this icon.
⚠	The logic controller is unable to communicate with the device. **NOTE:** When the Industrial Ethernet manager is STOPPED, all devices show this icon.

Industrial Ethernet Manager I/O Mapping

The **IOScanner I/O Mapping** tab of the Industrial Ethernet manager allows you to monitor the Modbus TCP IOScanner status and the health bit of the Modbus TCP slave devices:



Variable	Mapping	Channel	Address	Type	Defaul...	Curren...	Prepar...
Diagnostic							
		Global St...	%I...	UINT	0	2	
Healthbits							
		Healthbit...	%I...	WORD		63	
Healthbits_OTB1EODM9LP		Bit 0	%IX...	BOOL	FALSE	TRUE	
Healthbits_Altivar32		Bit 1	%IX...	BOOL	FALSE	TRUE	
Healthbits_Lexium32M		Bit 2	%IX...	BOOL	FALSE	TRUE	
Healthbits_Generic_Slave_channel3		Bit 3	%IX...	BOOL	FALSE	TRUE	
Healthbits_Generic_Slave_channel4		Bit 4	%IX...	BOOL	FALSE	TRUE	
Healthbits_Generic_Slave_channel5		Bit 5	%IX...	BOOL	FALSE	TRUE	
		Bit 6	%IX...	BOOL	FALSE	FALSE	
		Bit 7	%IX...	BOOL	FALSE	FALSE	
		Bit 8	%IX...	BOOL	FALSE	FALSE	
		Bit 9	%IX...	BOOL	FALSE	FALSE	
		Bit 10	%IX...	BOOL	FALSE	FALSE	
		Bit 11	%IX...	BOOL	FALSE	FALSE	
		Bit 12	%IX...	BOOL	FALSE	FALSE	
		Bit 13	%IX...	BOOL	FALSE	FALSE	
		Bit 14	%IX...	BOOL	FALSE	FALSE	
		Bit 15	%IX...	BOOL	FALSE	FALSE	
		Healthbit...	%I...	WORD		0	
		Healthbit...	%I...	WORD		0	
		Healthbit...	%I...	WORD		0	

Column		Use	Comment
Variable	**Diagnostic**	Assign a name to the global scanner status variable.	-
	Healthbits	Assign a name to each health bit. For example, name a health bit with the associated device name.	Health bits are grouped in 4 subfolders of 16 bits.
Address		Retrieve the address of each variable.	Addresses may be modified when the configuration is changed.

Column	Use	Comment
Current value	Monitor the Modbus TCP devices	For boolean values (health bit): ● **TRUE** = 1 ● **FALSE** = 0

Health Bits of EtherNet/IP Target

To monitor the health bit of the EtherNet/IP targets, you must:

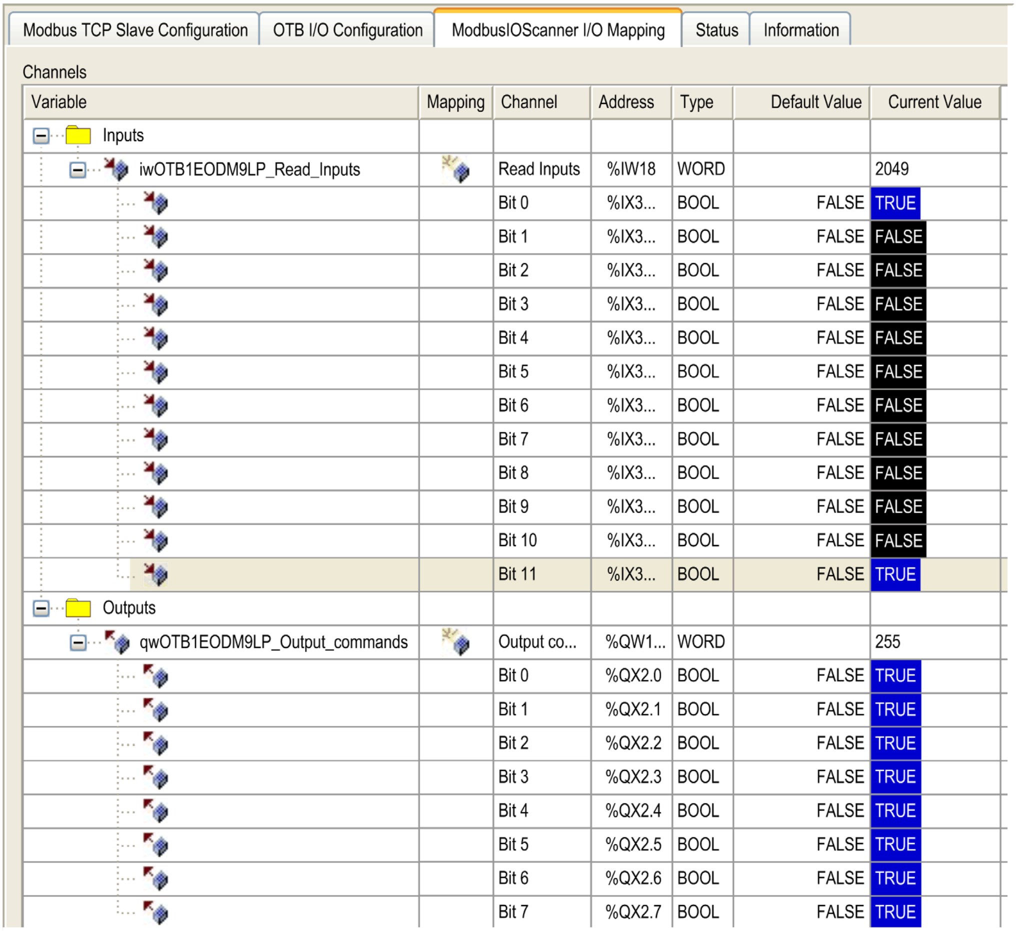
- Create a visualization in the application.
- Add in the visualization the health bits variables of:
  - `EipGetHealth` function block *(see page 173)*
  - `EIPGetHealthBit` function block *(see page 159)*

Slave Device Mapping

Industrial Ethernet devices have an I/O mapping tab containing their I/Os.

NOTE: Generic TCP/UDP does not have an I/O mapping tab;

This figure presents an example of an I/O mapping tab for an Advantys OTB slave device:



Modbus TCP Slave Configuration | OTB I/O Configuration | ModbusIOScanner I/O Mapping | Status | Information

Channels

Variable	Mapping	Channel	Address	Type	Default Value	Current Value
Inputs						
iwOTB1EODM9LP_Read_Inputs		Read Inputs	%IW18	WORD		2049
		Bit 0	%IX3...	BOOL	FALSE	TRUE
		Bit 1	%IX3...	BOOL	FALSE	FALSE
		Bit 2	%IX3...	BOOL	FALSE	FALSE
		Bit 3	%IX3...	BOOL	FALSE	FALSE
		Bit 4	%IX3...	BOOL	FALSE	FALSE
		Bit 5	%IX3...	BOOL	FALSE	FALSE
		Bit 6	%IX3...	BOOL	FALSE	FALSE
		Bit 7	%IX3...	BOOL	FALSE	FALSE
		Bit 8	%IX3...	BOOL	FALSE	FALSE
		Bit 9	%IX3...	BOOL	FALSE	FALSE
		Bit 10	%IX3...	BOOL	FALSE	FALSE
		Bit 11	%IX3...	BOOL	FALSE	TRUE
Outputs						
qwOTB1EODM9LP_Output_commands		Output co...	%QW1...	WORD		255
		Bit 0	%QX2.0	BOOL	FALSE	TRUE
		Bit 1	%QX2.1	BOOL	FALSE	TRUE
		Bit 2	%QX2.2	BOOL	FALSE	TRUE
		Bit 3	%QX2.3	BOOL	FALSE	TRUE
		Bit 4	%QX2.4	BOOL	FALSE	TRUE
		Bit 5	%QX2.5	BOOL	FALSE	TRUE
		Bit 6	%QX2.6	BOOL	FALSE	TRUE
		Bit 7	%QX2.7	BOOL	FALSE	TRUE

Column		Use	Comment
Variable	Inputs	Assign a name to each input of the device.	Each bit can also be mapped.
	Outputs	Assign a name to each output of the device.	
Address		Retrieve the address of each variable.	Addresses may be modified when the configuration is changed.
Current value		Follow the device inputs real-time value. The output values can be changes dynamically.	For boolean values (each bit): ● TRUE = 1 ● FALSE = 0

Troubleshooting

Main Issues

Symptom	Possible cause	Resolution
Industrial Ethernet manager is presented with a red triangle in the **Devices tree**.	The configuration is not compliant with the logic controller version.	● **Build** → **Clean all** ● **Build** → **Rebuild all** ● Ensure that the logic controller has the latest firmware version.
A device is presented with a red triangle in the **Devices tree**.	The logic controller is unable to communicate with the device.	● Verify device wiring and powering. ● Verify device IP address (by using the Remote ping service on the IP address of the device.). ● Verify whether the device supports the read/write request. ● Verify whether the accessed registers are relevant for this device. ● Verify whether the accessed registers are not write-protected. ● Verify that the FDR (Fast Device Replacement) service is properly configured inside the device. ● Verify that the **Master IP address** parameter is properly configured inside the device. ● Verify that the **Electronic Keying** parameters are properly configured for the device.
A device/channel is temporarily presented in red.	The wiring is unstable.	Verify the wiring.
	Configuration requires adjustment.	● Increase the health timeout value. ● Increase the repetition rate value.
	The load is too important for the Industrial Ethernet manager.	Verify the **Scanner Resources** tab *(see page 78)*.
Some states of the device are not presented in the application.	For Modbus TCP slave device: The repetition rate is too slow (the value is too high).	Decrease the repetition rate value for the channels associated to this device.
	For EtherNet/IP target device: The RPI values are too slow (the values are too high).	Decrease the RPI values for the connections associated with this device.
	The bus cycle task is not fast enough.	● Associate the scanner to a different task (Modbus TCP IOScanner or EtherNet/IP Scanner). ● Decrease the cycle value of the associated task.

Chapter 6
Maintenance

Maintenance Overview

Main Steps

In case of device replacement, the main steps are:

- Power off the machine or the part of the machine affected
- Unmount the device
- Mount the new device
- Power on the new device
- Prepare the device to be recognized by the system *(see page 86)*
- Apply the correct device configuration *(see page 89)*
- Acknowledge the device replacement (depends on your application)

Appendices

What Is in This Appendix?

The appendix contains the following chapters:

Appendix A
Modbus TCP IOScanner Library

Overview

This chapter describes the `ModbusTCPIOScanner` library.

What Is in This Chapter?

This chapter contains the following sections:

Section A.1
Modbus TCP IOScanner Functions

Overview

This section describes the functions included in the `ModbusTCPIOScanner` library.

What Is in This Section?

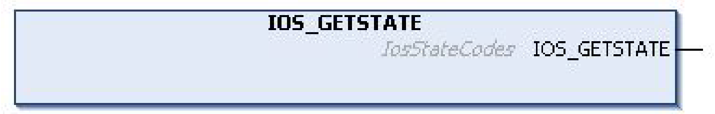
This section contains the following topics:

IOS_GETSTATE: Read the State of the Modbus TCP IOScanner

Function Description

This function returns the value corresponding to the state of the Modbus TCP IOScanner.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
`IOS_GETSTATE`	IosStateCodes *(see page 133)*	Return values: `IosStateCodes enum`

Example

This is an example of a call of this function:

```
mystate := IOS_GETSTATE() ; (* 0=NOT CONFIGURED 2=OPERATIONAL or
3=STOPPED. *)
```

IOS_START: Launch the Modbus TCP IOScanner

Function Description

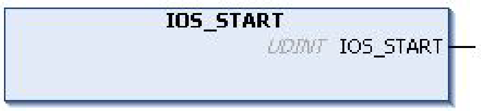
This function starts the Modbus TCP IOScanner.

It allows runtime control of the Modbus TCP IOScanner execution. By default, the Modbus TCP IOScanner starts automatically when the application starts.

This function call waits for the Modbus TCP IOScanner to be physically started, so it can last up to 5 ms.

Starting a Modbus TCP IOScanner already started has no effect.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*.

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
`IOS_START`	UDINT	● 0 = successful start ● Other value = start unsuccessful

Example

This is an example of a call of this function:

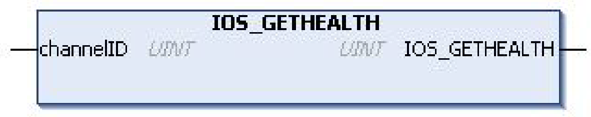
```
rc := IOS_START() ;
IF rc <> 0 THEN (* Abnormal situation to be processed at application level
*)
```

IOS_GETHEALTH: Read the Health Bit Value

Function Description

This function returns the health bit value of a specific channel.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*.

I/O Variable Description

This table describes the input variable:

Input	Type	Comment
`channelID`	UINT	Channel ID of the channel to monitor.

This table describes the output variable:

Output	Type	Comment
`IOS_GETHEALTH`	UINT	● 0: Channel I/O values are not updated ● 1: Channel I/O values are updated

Example

This is an example of a call of this function:

```
chID:=1 ;

channelHealth := IOS_GETHEALTH(chID)(* Get the health value (1=OK, 0=Not
OK) of the channel number chID. The channel ID is displayed in the
configuration editor of the device *)
```

IOS_STOP: Stop the Modbus TCP IOScanner

Function Description

This function stops the Modbus TCP IOScanner.

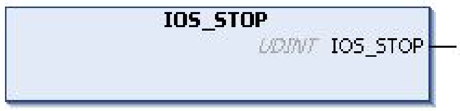
It allows runtime control of the Modbus TCP IOScanner execution. By default, the Modbus TCP IOScanner stops when the controller is `STOPPED`.

The Modbus TCP IOScanner has to be stopped, from the first cycle, until all network devices are operational.

This function call may take as long as 5 ms as it waits for the Modbus TCP IOScanner to physically stop.

Stopping an already stopped Modbus TCP IOScanner has no effect.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*.

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
`IOS_STOP`	UDINT	● 0 = successful stop ● Other value = stop unsuccessful

Example

This is an example of a call of this function:

```
rc := IOS_STOP() ;
```

```
IF rc <> 0 THEN (* Abnormal situation to be processed at application level
*)
```

CONFIGURE_OTB: Send the Software Configuration of the Advantys OTB

Function Block Description

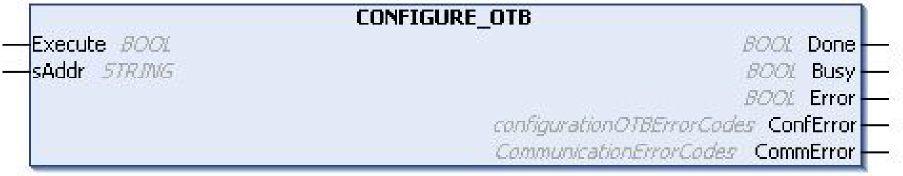
This function block sends the SoMachine configuration data of an Advantys OTB to the physical device through Modbus TCP.

It allows the update of the configuration parameters of an I/O island without third-party software.

The Modbus TCP IOScanner must be stopped before calling this function.

The execution of this function block is asynchronous. In order to check the configuration completion, the `Done`, `Busy`, and `Error` output flags must be tested at each application cycle.

Graphical Representation


CONFIGURE_OTB
Execute BOOL
sAddr STRING
BOOL Done
BOOL Busy
BOOL Error
configurationOTBErrorCodes ConfError
CommunicationErrorCodes CommError

IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*.

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
`Execute`	BOOL	Activation entry. Start the configuration on rising edge.
`sAddr`	STRING	OTB IP address. The format of the string must be `3{xx.xx.xx.xx}`

This table describes the output variables:

Output	Type	Comment
`Done`	BOOL	Set to TRUE when the configuration completion succeeded.
`Busy`	BOOL	Set to TRUE when the configuration is in progress.

Output	Type	Comment
`Error`	BOOL	Set to TRUE when the configuration ended with an error detected.
`ConfError`	configurationOTBErrorCodes *(see page 135)*	Return values: `configurationOTBErrorCodes`
`CommError`	CommunicationErrorCodes *(see page 134)*	Return values: `CommunicationErrorCodes`

Example

This is an example of a call of this function:

```
VAR
```

(*Function Block to configure OTB , need to stop the IOscanner before the execution of the FB*)

```
configure_OTB1: CONFIGURE_OTB;
```

(*init value different than 16#00000000 , IO_start_done=0 when we have a successful start*)

```
IO_start_done: UDINT := 1000;
```

(*init value different than 16#FFFFFFFF , IO_start_done=16#FFFFFFFF when we have a successful stop*)

```
IO_stop_done: UDINT := 1000;
```

(*Configure_OTB_done= true when we configure with success the OTB, then we can start the IO scanner*)

```
Configure_OTB_done: BOOL;
myBusy: BOOL;
myError: BOOL;
myConfError: configurationOTBErrorCodes;
myCommError: UINT;
myExecute: BOOL;
END_VAR
```

(* First, stop the IOScanner, before configuring OTB *)

```
IF NOT myExecute THEN
IO_stop_done:=IOS_STOP();
END_IF
```

```
(* Send the configuration data to OTB, at IP address 95.15.3.1, when myExecute is TRUE *)
configure_OTB1(
Execute:= myExecute,
sAddr:='3{95.15.3.1}' ,
Done=> Configure_OTB_done,
Busy=> myBusy,
Error=&gt; myError,
ConfError=&gt; myConfError,
CommError=&gt; myCommError);
(* After OTB is successfully configured, start the IOScanner *)
IF Configure_OTB_done THEN
IO_start_done:=IOS_START();
END_IF
```

Section A.2
Modbus TCP IOScanner Data Types

Overview

This section describes the data types of the `ModbusTCPIOScanner` library.

What Is in This Section?

This section contains the following topics:

IosStateCodes: Modbus TCP IOScanner Status Values

Enumeration Type Description

The `IosStateCodes` enumeration data type contains these values:

Enumerator	Value	Comment
`IosErr`	0	Modbus TCP IOScanner is in an error state.
`IosIdle`	1	Modbus TCP IOScanner is in IDLE state. The configuration is empty or not compliant.
`IosOperationnal`	2	Modbus TCP IOScanner is in OPERATIONAL state.
`IosStopped`	3	Modbus TCP IOScanner is in STOPPED state.

CommunicationErrorCodes: Error Detected Codes

Enumeration Type Description

The `CommunicationErrorCodes` enumeration data type contains these values:

Enumerator	Value	Comment
`CommunicationOK`	hex 00	Exchange is correct.
`TimedOut`	hex 01	Exchange stopped because of timeout.
`Canceled`	hex 02	Exchange stopped on user request.
`BadAddress`	hex 03	Address format is incorrect.
`BadRemoteAddr`	hex 04	Remote address is incorrect.
`BadMgtTable`	hex 05	Management table format is incorrect.
`BadParameters`	hex 06	Specific parameters are incorrect.
`ProblemSendingRq`	hex 07	Error detected on sending request to destination.
`RecvBufferTooSmall`	hex 09	Size of reception buffer is too small.
`SendBufferTooSmall`	hex 0A	Size of transmission buffer is too small.
`SystemResourceMissing`	hex 0B	System resource is missing.
`BadTransactionNb`	hex 0C	Transaction number is incorrect.
`BadLength`	hex 0E	Length is incorrect.
`ProtocolSpecificError`	hex FE	The detected operation error contains protocol-specific code.
`Refused`	hex FF	Transaction is refused.

configurationOTBErrorCodes: Error Detected Codes in the OTB Configuration

Enumeration Type Description

The `configurationOTBErrorCodes` enumeration data type contains these values:

Enumerator	Value	Comment
`ConfigurationOK`	hex 00	OTB configuration is done successful.
`IPAddrErr`	hex 01	`sAddr` input parameter is incorrect.
`ChannelNbErr`	hex 02	There is no OTB channel initialization value for this IP address.
`ChannelInitValueErr`	hex 03	Cannot get the OTB channel initialization value.
`CommunicationErr`	hex 04	OTB configuration stopped because of an error detected.
`IosStateErr`	hex 05	The Modbus TCP IOScanner is running. The Modbus TCP IOScanner must be stopped before executing the `CONFIGURE_OTB` function block.

Appendix B
EtherNet/IP Explicit Messaging Library

What Is in This Chapter?

This chapter contains the following sections:

Section B.1
EtherNet/IP Explicit Messaging Functions

What Is in This Section?

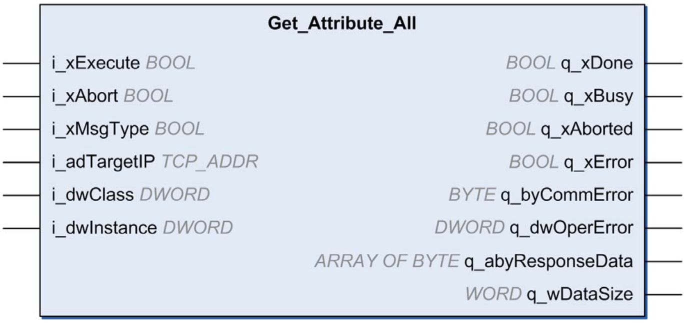
This section contains the following topics:

Get_Attribute_All: Get All Attributes of an Object

Function Block Description

This function block returns the content of all attributes of an object.

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
i_xExecute	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input Execute starts the function block. The function block continues execution and the output Busy is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input Execute is ignored while the function blocks are being executed. ● FALSE: If Enable is set to FALSE, the outputs Done, Error, or CommandAborted are set to TRUE for one cycle. ● TRUE: If Enable is set to FALSE, the outputs Done, Error, or CommandAborted remain set to TRUE.

Input	Data type	Comment
`i_xAbort`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by another function block.
`i_xMsgType`	BOOL	● FALSE: UCCM ● TRUE: Connected (Class 3) message
`i_adTargetIP`	TCP_ADDR	IP address of target.
`i_dwClass`	DWORD	Target class. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It must be 0xFFFFFFFF if the class is not part of the request.
`i_dwInstance`	DWORD	Target instance. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It can be 0 if the target is a class instance. It must be 0xFFFFFFFF if the instance is not part of the request.

Outputs

This table describes the output variable:

Output	Data type	Comment
`q_xDone`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
`q_xBusy`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
`q_xAborted`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by `Abort` input.
[1] The `Get_Attribute_All` function returns a formatted buffer according to the ODVA specification. Refer to CIP `Get_Attribute_All` response.		

Output	Data type	Comment
`q_xError`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.
`q_byCommError`	BYTE	Gives information about the detected error.
`q_dwOperError`	DWORD	Gives information about the detected error.
`q_abyResponseData`	ARRAY OF BYTE 0…MAX_EIP_REQUEST_ DATA_SIZE	Response data in case of a success.[1]
`q_wDataSize`	WORD	The size of the response data in bytes.

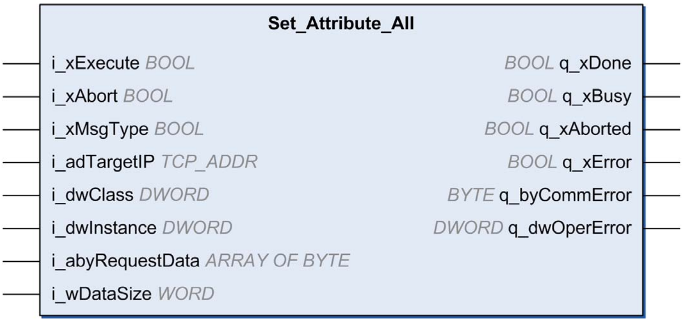
[1] The `Get_Attribute_All` function returns a formatted buffer according to the ODVA specification. Refer to CIP `Get_Attribute_All` response.

Set_Attribute_All: Set All Attributes of an Instance or Class

Function Block Description

This function block sets all attributes of an instance or classes.

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
i_xExecute	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input Execute starts the function block. The function block continues execution and the output Busy is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input Execute is ignored while the function blocks are being executed. ● FALSE: If Enable is set to FALSE, the outputs Done, Error, or CommandAborted are set to TRUE for one cycle. ● TRUE: If Enable is set to FALSE, the outputs Done, Error, or CommandAborted remain set to TRUE.
1 The input data buffer must be formatted as well. Refer to Set_Attribute_All request data in the ODVA EtherNet/IP specification volume 1.		

Input	Data type	Comment
`i_xAbort`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by another function block.
`i_xMsgType`	BOOL	● FALSE: UCCM ● TRUE: Connected (Class 3) message
`i_adTargetIP`	TCP_ADDR	IP address of target.
`i_dwClass`	DWORD	Target class. Refer to How To Find Object Information in Device Documentation. *(see page 161)* It must be 0xFFFFFFFF if the class is not part of the request.
`i_dwInstance`	DWORD	Target instance. Refer to How To Find Object Information in Device Documentation. *(see page 161)* It can be 0 if the target is class instance. It must be 0xFFFFFFFF if the instance is not part of the request.
`i_abyRequestData`	ARRAY OF BYTE 0…MAX_EIP_REQUEST_ DATA_SIZE	Data must be sent to the target. If not used, `wDataSize` must be 0 [1].
`q_wDataSize`	WORD	The actual size of the `abyRequestData` [1].

[1] The input data buffer must be formatted as well. Refer to `Set_Attribute_All` request data in the ODVA EtherNet/IP specification volume 1.

Outputs

This table describes the output variable:

Output	Data type	Comment
`q_xDone`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
`q_xBusy`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.

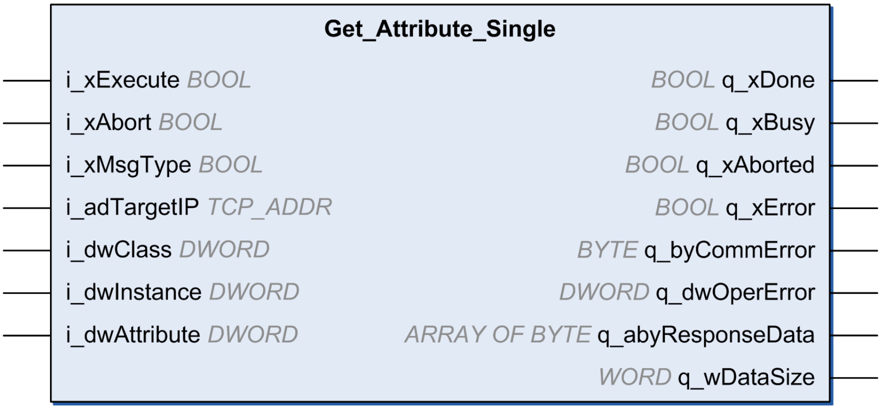
Output	Data type	Comment
q_xAborted	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by Abort input.
q_xError	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.
q_byCommError	BYTE	Gives information about the detected error.
q_dwOperError	DWORD	Gives information about the detected error.

Get_Attribute_Single: Get an Attribute of an Object

Function Block Description

This function block returns the content of a specific attribute of an object instance.

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
`i_xExecute`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.

Input	Data type	Comment
i_xAbort	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by another function block.
i_xMsgType	BOOL	● FALSE: UCCM ● TRUE: Connected (Class 3) message
i_adTargetIP	TCP_ADDR	IP address of target.
i_dwClass	DWORD	Target class. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It must be 0xFFFFFFFF if the class is not part of the request.
i_dwInstance	DWORD	Target instance. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It can be 0 if the target is class instance. It must be 0xFFFFFFFF if the instance is not part of the request.
i_dwAttribute	DWORD	Target attribute. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It must be 0xFFFFFFFF if the attribute is not part of the request.

Outputs

This table describes the output variable:

Output	Data type	Comment
q_xDone	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
q_xBusy	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.

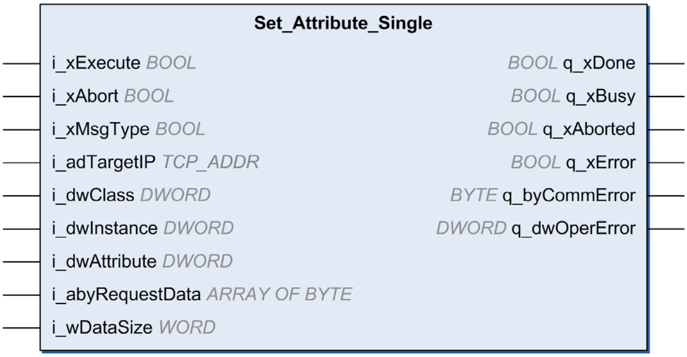
Output	Data type	Comment
q_xAborted	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by Abort input.
q_xError	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.
q_byCommError	BYTE	Gives information about the detected error.
q_dwOperError	DWORD	Gives information about the detected error.
q_abyResponseData	ARRAY OF BYTE 0…MAX_EIP_REQUEST_ DATA_SIZE	Response data in case of a success.
q_wDataSize	WORD	The size of the response data in bytes.

Set_Attribute_Single: Set an Attribute of an Object

Function Block Description

This function block set the content of a specific attribute of an object instance

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
`i_xExecute`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.

1 The input data buffer must be formatted as well. Refer to `Set_Attribute_Single` request data in the ODVA EtherNet/IP specification volume 1.

Input	Data type	Comment
`i_xAbort`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by another function block.
`i_xMsgType`	BOOL	● FALSE: UCCM ● TRUE: Connected (Class 3) message
`i_adTargetIP`	TCP_ADDR	IP address of target.
`i_dwClass`	DWORD	Target class. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It must be 0xFFFFFFFF if the class is not part of the request.
`i_dwInstance`	DWORD	Target instance. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It can be 0 if the target is class instance. It must be 0xFFFFFFFF if the instance is not part of the request.
`i_dwAttribute`	DWORD	Target attribute. Refer to How To Find Object Information in Device Documentation *(see page 161)*. It must be 0xFFFFFFFF if the attribute is not part of the request.
`i_abyRequestData`	ARRAY OF BYTE 0…MAX_EIP_REQUEST_ DATA_SIZE	Data must be sent to the target. If not used, `wDataSize` must be 0 [1].
`q_wDataSize`	WORD	The actual size of the `abyRequestData` [1].

[1] The input data buffer must be formatted as well. Refer to `Set_Attribute_Single` request data in the ODVA EtherNet/IP specification volume 1.

Outputs

This table describes the output variable:

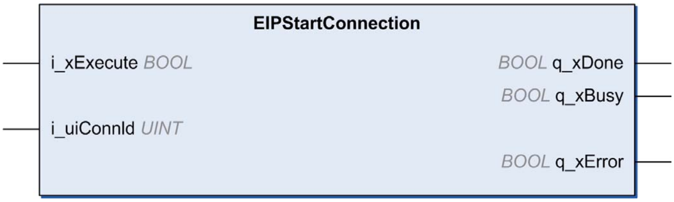
Output	Data type	Comment
`q_xDone`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
`q_xBusy`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
`q_xAborted`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by `Abort` input.
`q_xError`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.
`q_byCommError`	BYTE	Gives information about the detected error.
`q_dwOperError`	DWORD	Gives information about the detected error.

EIPStartConnection: Start a Connection

Function Block Description

This function block starts the specified connection by accessing the corresponding control bits and then returns done when the connection is started.

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
i_xExecute	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input Execute starts the function block. The function block continues execution and the output Busy is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input Execute is ignored while the function blocks are being executed. ● FALSE: If Enable is set to FALSE, the outputs Done, Error, or CommandAborted are set to TRUE for one cycle. ● TRUE: If Enable is set to FALSE, the outputs Done, Error, or CommandAborted remain set to TRUE.
i_uiConnId	UINT	Connection id.

Outputs

This table describes the output variable:

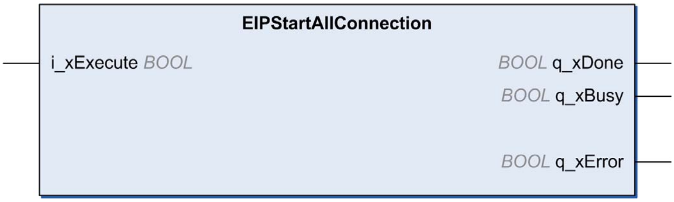
Output	Data type	Comment
`q_xDone`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
`q_xBusy`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
`q_xError`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.

EIPStartAllConnection: Start All Connections

Function Block Description

This function block starts all connections by accessing the corresponding control bits and then returns done when the connections are started.

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
`i_xExecute`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.

Outputs

This table describes the output variable:

Output	Data type	Comment
q_xDone	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
q_xBusy	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
q_xError	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.

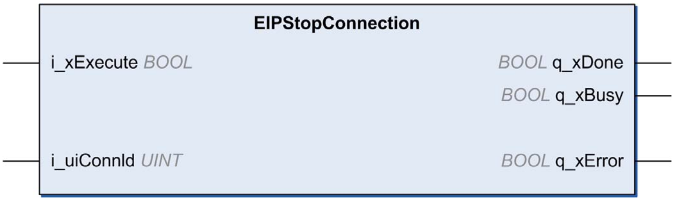
EIPStopConnection: Stop a Connection

Function Block Description

This function block stops the specified connection by accessing the corresponding control bits, then returns done when the connection is stopped.

NOTE: Even if the connection can be stopped, the system will try to reopen the connection. To stop a communication, you need to disable the associated remote adapter: `<DeviceName>.DisableRemoteAdapter (TRUE);`

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
`i_xExecute`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.
`i_uiConnId`	UINT	Connection id.

Outputs

This table describes the output variable:

Output	Data type	Comment
q_xDone	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
q_xBusy	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
q_xError	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.

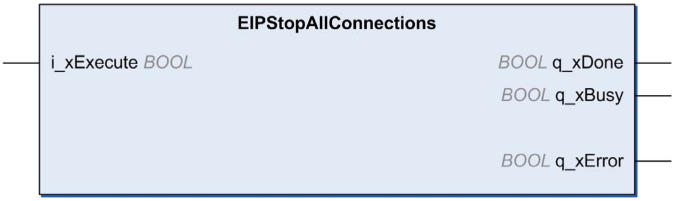
EIPStopAllConnections: Stop All Connections

Function Block Description

This function block stops all the connections by accessing the corresponding control bits, then returns done when the connections are stopped.

NOTE: Even if the connection can be stopped, the system will try to reopen the connection. To stop a communication, you need to disable the associated remote adapter: `<DeviceName>.DisableRemoteAdapter (TRUE);`

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
`i_xExecute`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.

Outputs

This table describes the output variable:

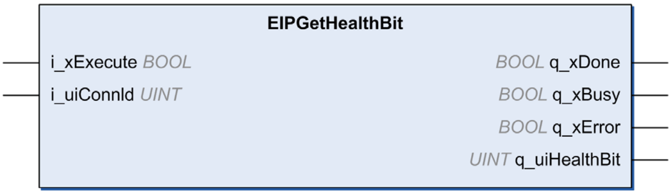
Output	Data type	Comment
q_xDone	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
q_xBusy	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
q_xError	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.

EIPGetHealthBit: Get the Health Bit Value

Function Block Description

This function block returns the value of a specified health bit.

Graphical Representation



Inputs

This table describes the input variable:

Input	Data type	Comment
`i_xExecute`	BOOL	Value range: FALSE, TRUE. Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.
`i_uiConnId`	UINT	Connection id.

Outputs

This table describes the output variable:

Output	Data type	Comment
q_xDone	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
q_xBusy	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.
q_xError	BOOL	Value range: FALSE, TRUE. Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.
q_byCommError	BYTE	Gives information about the detected error.
q_byOperError	BYTE	Gives information about the detected error.
q_HealthValue	UINT	Returns the health value.

How To Find Object Information in Device Documentation

Presentation

In the device documentation, you will find descriptions of objects corresponding to the data you want to access. Usually, they are referred to as *application objects*, accessible though explicit messaging, or are described as belonging to category 3.

An object is similar to a dictionary in software programming. There are several types of dictionaries, for example, automatically ordered, or with different search mechanisms. For example, `SortedDictionary` is one class, and `UnsortedDictionary` is another class. If an object is built using one of these classes, the identifiers of the classes are `SortedDictionary` and `UnsortedDictionary`, respectively.

Instantiating such an object with a variable name `myDictionary` means that a reserved area in memory is allocated to this dictionary called, for example, `instance`. Its identifier is `myDictionary`.

Within a dictionary, values are stored in a structure (key, value). The dictionary provides a method to get the list of the keys called, for example, `attribute`. Its identifier is `GetKeys`. This dictionary has also a method to get the list of values. This method is another `attribute`, whose identifier is `Values`. As the two attribute identifiers are common to both classes, they are called "attributes" of the "class". Indeed, there is a dedicated attribute for the `SortedDictionary` whose identifier is `GetSortedKeys`. In this case, it is called an “instance attribute”.

Attributes can also support several services. For the attribute `GetKeys`, it supports the service **Get_Attribute_Single** (read access), whereas the attribute `Values` supports the services **Get_Attribute_Single** or **Set_Attribute_Single** (read and write access); the corresponding identifiers of the supported services are `Get_Attribute_Single`, or `Set_Attribute_Single`.

Depending on the function block to use, the corresponding information is:

- `i_byService`: the identifier of the service to use to access the data; can be found by searching for example "Supported class attribute services", or "Supported instance attribute services".
- `i_dwClass`: the identifier of the class describing the object to access, "Class ID" is a numeric property, most of the time expressed as an hexadecimal value.
- `i_dwInstance`: the identifier of the instance describing the object to access, "Instance ID" is a numeric property, most of the time expressed as an hexadecimal value.
- `i_dwAttribute`: the identifier of the attribute to access, this is the data of interest; it can be a class attribute common to all instances of the same class, or just an instance attribute, "Attribute ID" is a numeric property, most of the time expressed as an hexadecimal value.
- `i_dwMember`: identifies the object as being a member of a group, but is rarely used.

Section B.2
EIP Explicit Messaging Data Types

What Is in This Section?

This section contains the following topics:

CommunicationErrorCodes: Communication Error Codes

Enumerated Type Description

The `CommunicationErrorCodes` enumerated type contains information about communication diagnostics, such as interruptions and detected errors. It contains these values:

Enumerator	Value (hex)	Description
`CommunicationOK`	`00`	The exchange is valid.
`TimedOut`	`01`	The exchange stopped when the timeout expired.
`Canceled`	`02`	The exchange was stopped by a user request (the `Abort` command).
`BadAddress`	`03`	The address format is incorrect.
`BadRemoteAddr`	`04`	The remote address is incorrect.
`BadMgtTable`	`05`	The management table format is incorrect.
`BadParameters`	`06`	Specific parameters are incorrect.
`ProblemSendingRq`	`07`	There was a problem while sending the request to the destination.
`RecvBufferTooSmall`	`09`	The reception buffer size is insufficient.
`SendBufferTooSmall`	`0A`	The transmission buffer size is insufficient.
`SystemRessourceMissing`	`0B`	A system resource is unavailable.
`BadTransactionNb`	`0C`	The transaction number is incorrect.
`BadLength`	`0E`	The length is incorrect.
`ProtocolSpecificError`	`FE`	The operation error code contains a protocol-specific code.
`Refused`	`FF`	The message was refused.

OperationErrorCodes: Operation Error Codes

Enumerated Type Description

The OperationErrorCodes enumerated type contains codes that correspond to detected errors.

00

When the CommunicationErrorCodes is 00 hex (correct transaction), the OperationErrorCodes enumerated type can return these values:

Enumerator	Value (hex)	Description
OperationOK	00	The exchange is valid.
NotProcessed_or_ TargetResourceMissing	01	The request has not been processed.
BadResponse	02	The received response is incorrect.

FF

When the CommunicationErrorCodes is FF hex (message refused), the OperationErrorCodes enumerated type can return these values:

Enumerator	Value (hex)	Description
NotProcessed_or_ TargetResourceMissing	01	The target system resource is incommunicative.
BadLength	05	The length is incorrect.
CommChannelErr	06	The communication channel is associated with a detected error.
BadAddr	07	The address is incorrect.
SystemResourceMissing	0B	A system resource is unavailable.
TargetCommInactive	0C	A target communication function is not active.
TargetMissing	0D	The target is incommunicative.
ChannelNotConfigured	0F	The channel is not configured.

FE

When the CommunicationErrorCodes is FE hex, the `OperationErrorCodes` enumerated type can return these values:

Status name	Value (hex)	Description
Success	0x00	Service was successfully performed by the object specified.
Connection failure	0x01	A connection-related service is unsuccessful along the connection path.
Resource unavailable	0x02	Resources needed for the object to perform the requested service are unavailable
Invalid parameter value	0x03	See Status code 0x20, which is the preferred value to use for this condition.
Path segment error	0x04	The path segment identifier or the segment syntax was not understood by the processing node. Path processing shall stop when a path segment error is encountered.
Path destination unknown	0x05	The path is referencing an object class, instance, or structure element that is incorrect or is not contained in the processing node. Path processing stops when a this error is encountered.
Partial transfer	0x06	Only part of the expected data was transferred.
Connection lost	0x07	The messaging connection was lost.
Service not supported	0x08	The requested service is not implemented or is not defined for this object Class/Instance.
Invalid attribute value	0x09	Invalid attribute data.
Attribute list error	0x0A	An attribute in the Get_Attribute_List or Set_Attribute_List response has a non-zero status.
Already in requested mode/state	0x0B	The object is already in the mode/state being requested by the service.
Object state conflict	0x0C	The object cannot perform the requested service in its present mode/state.
Object already exists	0x0D	The requested instance of object to be created already exists.
Attribute not settable	0x0E	A request to modify a non-modifiable attribute was received.
Privilege violation	0x0F	A permission/privilege check is unsuccessful.
Device state conflict	0x10	The device mode/state does not allow the execution of the requested service.
Reply data too large	0x11	The data to be transmitted in the response buffer is larger than the allocated response buffer.
Fragmentation of a primitive value	0x12	The service specified an operation that is going to fragment a primitive data value, that is, half a REAL data type.
Not enough data	0x13	The service did not supply enough data to perform the specified operation.

Status name	Value (hex)	Description
Attribute not supported	0x14	The attribute specified in the request is not supported.
Too much data	0x15	The service supplied more data than was expected.
Object does not exist	0x16	The object specified does not exist in the device.
Service fragmentation sequence not in progress	0x17	The fragmentation sequence for this service is not active for this data.
No stored attribute data	0x18	The attribute data of this object was not saved prior to the requested service.
Store operation failure	0x19	The attribute data of this object was not saved.
Routing failure, request packet too large	0x1A	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service.
Routing failure, response packet too large	0x1B	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to end the service.
Missing attribute list entry data	0x1C	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behavior.
Invalid attribute value list	0x1D	The service is returning the list of attributes supplied with status information for those attributes that were invalid.
Embedded service error	0x1E	An embedded service resulted in an error.
Vendor specific error	0x1F	A vendor specific error has been detected. The additional code field of the error response defines the particular error encountered. Use of this general error code should only be performed when none of the error codes presented in this table or within an object class definition accurately reflect the error.
Invalid parameter	0x20	A parameter associated with the request was invalid. This code is used when a parameter does not meet the requirements of this specification and/or the requirements defined in an application object specification.
Write-once value or medium already written	0x21	An attempt was made to write to a write-once medium (for example, WORM drive, PROM) that has already been written, or to modify a value that cannot be changed once established.
Invalid reply received	0x22	An invalid reply is received (for example, reply service code does not match the request service code, or reply message is shorter than the minimum expected reply size). This status code can serve for other causes of invalid replies.
Buffer overflow	0x23	The message received is larger than the receiving buffer can handle. The entire message is discarded.

Status name	Value (hex)	Description
Message format error	0x24	The format of the received message is not supported by the server.
Key failure in path	0x25	The Key segment that is included as the first segment in the path does not match the destination module. The object specific status indicates which part of the key check is unsuccessful.
Path size invalid	0x26	The size of the path which is sent with the Service request is either not sufficient to allow the request to be routed to an object, or too much routing data is included.
Unexpected attribute in list	0x27	An attempt was made to set an attribute that is not able to be set at this time.
Invalid member ID	0x28	The member ID specified in the request does not exist in the specified Class/Instance/Attribute.
Member not settable	0x29	A request to modify a non-modifiable member was received.
Group 2 only server general failure	0x2A	This error code may only be reported by Group 2 only servers with 4K or less code space and only in place of service not supported, attribute not supported and attribute not settable.
Unknown Modbus Error	0x2B	A CIP to Modbus translator received an invalid Modbus exception code.
Attribute not gettable	0x2C	A request to read a non-readable attribute is received.
Instance not deletable	0x2D	The requested object instance cannot be deleted.
Service not supported for specified path 1	0x2E	The object supports the service, but not for the designated application path (for example, attribute). **NOTE:** Not to be used for any set service (use General status code 0x0E or 0x29 instead).
Timeout	0xFF	No response from the target.

Appendix C
EtherNet/IP Scanner Library

Overview

This chapter describes the EtherNet/IP Scanner library.

What Is in This Chapter?

This chapter contains the following sections:

Section C.1
EtherNet/IP Scanner Functions

Overview

This section describes the functions included in the EtherNet/IP Scanner library.

What Is in This Section?

This section contains the following topics:

EipControl: Control the EtherNet/IP Scanner

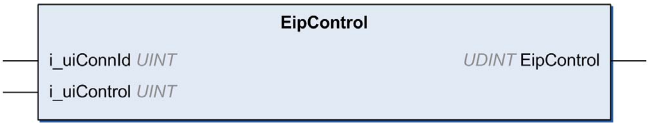
Function Description

This function starts or stops one or more EtherNet/IP connection.

The application doesn't manipulate the control bits directly. EipControl function must be used.

The connection ID can be found for each EtherNet/IP target device in its **Connections** tab *(see page 57)*.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*.

I/O Variable Description

This table describes the input variable:

Input	Type	Comment
`i_uiConnId`	UINT	Connection ID *(see page 57)* of the connection monitored.
`i_uiControl`	UINT	● 0 = Start specified connection ● 1 = Stop specified connection ● 2 = Start all connections ● 3 = Stop all connections

This table describes the output variable:

Output	Type	Comment
`EipControl`	UDINT	● 0 = successful start or stop ● -1 = incorrect connection ID

Example

This is an example of a call of this function:

```
rc := EipControl(0,257) ;(* opens the connection No 116 *)

IF rc <> 0 THEN (* Abnormal situation to be processed at application level
*)
```

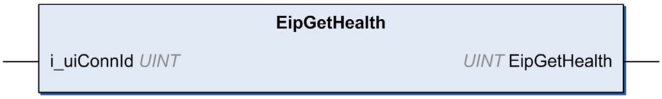
EipGetHealth: Read the Health Bit Value

Function Description

This function returns the health bit value of a specific EtherNet/IP connection.

The connection ID can be found for each EtherNet/IP target device in its **Connections** tab *(see page 57)*.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*.

I/O Variable Description

This table describes the input variable:

Input	Type	Comment
`i_uiconnId`	UINT	Connection ID *(see page 57)* of the connection monitored.

This table describes the output variable:

Output	Type	Comment
`EipGetHealth`	UINT	● 0: Connection not established ● 1: Connection established

Example

This is an example of a call of this function:

```
conID:=257 ;
```

```
channelHealth := EipGetHealth(conID)(* Get the health value (1=OK, 0=Not
OK) of the connection number conID. The connection ID is displayed in the
configuration editor of the device *)
```

EipDataExch: Send an Explicit Message

Function Block Description

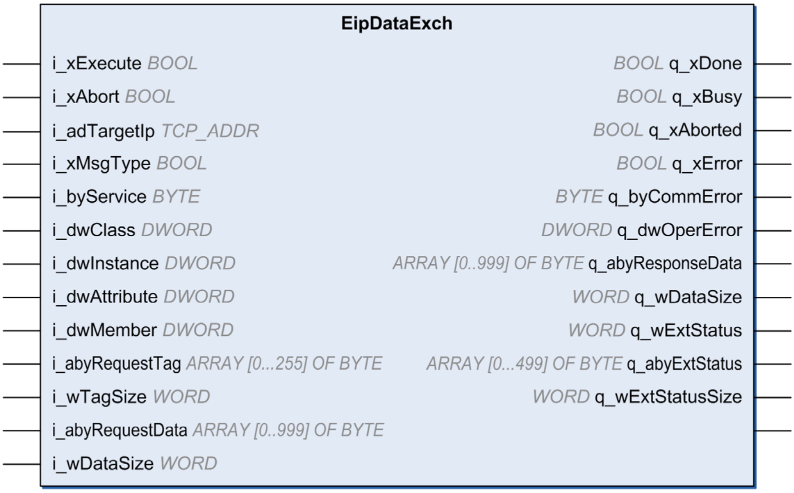
This function block sends an explicit message.

The time to perform the operation is configurable from the Industrial Ethernet manager *(see page 35)*.

There is a timeout value for connected messages and a timeout value for unconnected messages.

This generic function block may be used for features not implemented in the EtherNet/IP Explicit messaging Library.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 187)*

I/O Variable Description

This table describes the input variable:

Input	Type	Inherited from	Comment
`i_xExecute`	BOOL	BASE	Default value: FALSE. A rising edge of the input `Execute` starts the function block. The function block continues execution and the output `Busy` is set to TRUE. Function blocks which trigger a movement can be restarted while they are being executed. The target values are overwritten by the new values at the point in time the rising edge occurs. A rising edge at the input `Execute` is ignored while the function blocks are being executed. ● FALSE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` are set to TRUE for one cycle. ● TRUE: If `Enable` is set to FALSE, the outputs `Done`, `Error`, or `CommandAborted` remain set to TRUE.
`i_xAbort`	BOOL	BASE	Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by another function block.
`i_xMsgType`	BOOL	-	● FALSE: UCCM ● TRUE: Connected (Class 3) message
`i_adTargetIP`	TCP_ADDRES *(see page 182)*	-	IP address of target
`i_byService`	BYTE	-	Service to be performed (service code see above)
`i_dwClass`	DWORD	-	Target class. Refer to How To Find Object Information in Device Documentation *(see page 161)*. Must be 0xFFFFFFFF if the class should not be a part of request

Input	Type	Inherited from	Comment
i_dwInstance	DWORD	-	Target instance. Refer to How To Find Object Information in Device Documentation *(see page 161)*. Can be 0 if the target is class instance. Must be 0xFFFFFFFF if the instance should not be a part of request
i_dwAttribute	DWORD	-	Target attribute. Refer to How To Find Object Information in Device Documentation *(see page 161)*. Must be 0xFFFFFFFF if the attribute should not be a part of request
i_dwMember	DWORD	-	Target member. Refer to How To Find Object Information in Device Documentation *(see page 161)*. Must be 0xFFFFFFFF if the member should not be a part of request
i_abyRequestTag	ARRAY OF [0…250] BYTE	-	Target extended symbol segment. If not used i_wTagSize must be 0
i_wTagSize	WORD	-	The actual size of the i_abyRequestTag
i_abyRequestData	ARRAY OF [0…999] BYTE	-	Data that should be sent to the target. If not used i_wDataSize must be 0
i_wDataSize	WORD	-	The actual size of the i_abyRequestData

This table describes the output variable:

Output	Type	Inherited from	Comment
q_xDone	BOOL	BASE	Default value: FALSE. ● FALSE: Execution has not been started, or an error has been detected. ● TRUE: Execution terminated without an error detected.
q_xBusy	BOOL	BASE	Default value: FALSE. ● FALSE: Execution of the function block has not been started or not been terminated. ● TRUE: Function block is being executed.

Output	Type	Inherited from	Comment
q_xAborted	BOOL	BASE	Default value: FALSE. ● FALSE: Execution has not been aborted. ● TRUE: Execution has been aborted by Abort input.
q_xError	BOOL	BASE	Default value: FALSE. ● FALSE: Execution of the function block is running, no error has been detected. ● TRUE: An error has been detected in the execution of the function block.
q_byCommError	CommunicationError Codes *(see page 180)*	BASE	Communication error code
q_dwOperError	OperationErrorCodes *(see page 181)*	BASE	Operation error code
q_abyResponseData	ARRAY OF [0…999] BYTE	-	Response Data in case of a success
q_wDataSize	WORD	-	The size of the response Data in bytes
q_abyExtStatus	ARRAY OF [0…499] BYTE	-	Extended Status Data in case of an error response
q_wExtStatusSize	WORD	-	The size of the Extended Status Data in 16-bit words
q_wExtStatus	WORD	-	Extended status word

Example

This is an example of a call of this function:

```
MyEipDataExch(
    i_xExecute:= Execute,
        i_xAbort:= Abort,
        q_xDone=> Done,
        q_xBusy=> Busy,
        q_xAborted=> Aborted,
        q_xError=> Err,
        q_byCommError=> CommError,
        q_dwOperError=> OperError,
        i_adTargetIp:= IpAddr,
        i_xMsgType:= MsgType,
        i_byService:= Service,
        i_dwClass:= Class,
        i_dwInstance:= Instance,
        i_dwAttribute:= Attribute,
        i_dwMember:= Member,
        i_abyRequestTag:= RequestTag,
        i_wTagSize:= TagSize,
        i_abyRequestData:= RequestData,
        i_wDataSize:= ReqDataSize,
        q_abyResponseData=> ResponseData,
        q_wDataSize=> ResDataSize,
        q_abyExtStatus=> ExtStatusArray,
        q_wExtStatusSize=> ExtStatusSize,
    q_wExtStatus => ExtStatus);
```

Section C.2
EtherNet/IP Scanner Data Types

Overview

This section describes the data types of the EtherNet/IP Scanner library.

What Is in This Section?

This section contains the following topics:

CommunicationErrorCodes: Communication Error Codes

Enumerated Type Description

The `CommunicationErrorCodes` enumerated type contains information about communication diagnostics, such as interruptions and detected errors. It contains these values:

Enumerator	Value (hex)	Description
`CommunicationOK`	`00`	The exchange is valid.
`TimedOut`	`01`	The exchange stopped when the timeout expired.
`Canceled`	`02`	The exchange was stopped by a user request (the `Abort` command).
`BadAddress`	`03`	The address format is incorrect.
`BadRemoteAddr`	`04`	The remote address is incorrect.
`BadMgtTable`	`05`	The management table format is incorrect.
`BadParameters`	`06`	Specific parameters are incorrect.
`ProblemSendingRq`	`07`	There was a problem while sending the request to the destination.
`RecvBufferTooSmall`	`09`	The reception buffer size is insufficient.
`SendBufferTooSmall`	`0A`	The transmission buffer size is insufficient.
`SystemRessourceMissing`	`0B`	A system resource is unavailable.
`BadTransactionNb`	`0C`	The transaction number is incorrect.
`BadLength`	`0E`	The length is incorrect.
`ProtocolSpecificError`	`FE`	The operation error code contains a protocol-specific code.
`Refused`	`FF`	The message was refused.

OperationErrorCodes: Operation Error Codes

Enumerated Type Description

The `OperationErrorCodes` enumerated type contains codes that correspond to detected errors.

00

When the CommunicationErrorCodes is 00 hex (correct transaction), the `OperationErrorCodes` enumerated type can return these values:

Enumerator	Value (hex)	Description
`OperationOK`	`00`	The exchange is valid.
`NotProcessed_or_TargetResourceMissing`	`01`	The request has not been processed.
`BadResponse`	`02`	The received response is incorrect.

FF

When the CommunicationErrorCodes is FF hex (message refused), the `OperationErrorCodes` enumerated type can return these values:

Enumerator	Value (hex)	Description
`NotProcessed_or_TargetResourceMissing`	`01`	The target system resource is incommunicative.
`BadLength`	`05`	The length is incorrect.
`CommChannelErr`	`06`	The communication channel is associated with a detected error.
`BadAddr`	`07`	The address is incorrect.
`SystemResourceMissing`	`0B`	A system resource is unavailable.
`TargetCommInactive`	`0C`	A target communication function is not active.
`TargetMissing`	`0D`	The target is incommunicative.
`ChannelNotConfigured`	`0F`	The channel is not configured.

FE

When the communication error code is FE hex, the `OperationErrorCodes` enumerated type contains the protocol-specific error detection code. (Refer to your specific protocol's error detection codes.)

TCP_ADDR: Address for TCP Devices

Structure Description

The TCP_ADDR structure data type contains the address for TCP devices. It contains these variables:

Variable	Type	Description
A	BYTE	First value in IP address A.B.C.D
B	BYTE	Second value in IP address A.B.C.D
C	BYTE	Third value in IP address A.B.C.D
D	BYTE	Last value in IP address A.B.C.D
port	WORD	TCP port number (Modbus default: 502)

Appendix D
Motion Control Library

Motion Control Library

Overview

This document describes function blocks that are used to control ATV32, ATV320, ATV340 drives, ATV6••, ATV71, ATV9••, LXM32M, ILA, ILE and ILS drives in fieldbus under the SoMachine software environment.

For more details, refer to Motion Control Library Guide.

Appendix E
Generic TCP UDP Library

Generic TCP UDP Library

Overview

The TcpUdpCommunication library provides implementing TCP and UDP using IPv4.

The library provides the core functionality for implementing socket-based network communication protocols using TCP (client and server) or UDP (including broadcast and multicast if supported by the platform). Only IPv4-based communication is supported.

The application protocol used by the remote side (which can be hardware such as barcode scanners, vision cameras, industrial robots, or computer systems running software like database servers) has to be implemented using this library. While this requires extensive knowledge of socket-based communication and the protocol used, the TcpUdpCommunication library allows you to concentrate on the application layers.

For more details, refer to TcpUdpCommunication Library Guide.

Appendix F
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

- IL: Instruction List
- ST: Structured Text
- LD: Ladder Diagram
- FBD: Function Block Diagram
- CFC: Continuous Function Chart

This chapter provides functions and function blocks representation examples and explains how to use them for IL and ST languages.

What Is in This Chapter?

This chapter contains the following topics:

Differences Between a Function and a Function Block

Function

A function:

- is a POU (Program Organization Unit) that returns one immediate result.
- is directly called with its name (not through an instance).
- has no persistent state from one call to the other.
- can be used as an operand in other expressions.

Examples: boolean operators (`AND`), calculations, conversion (`BYTE_TO_INT`)

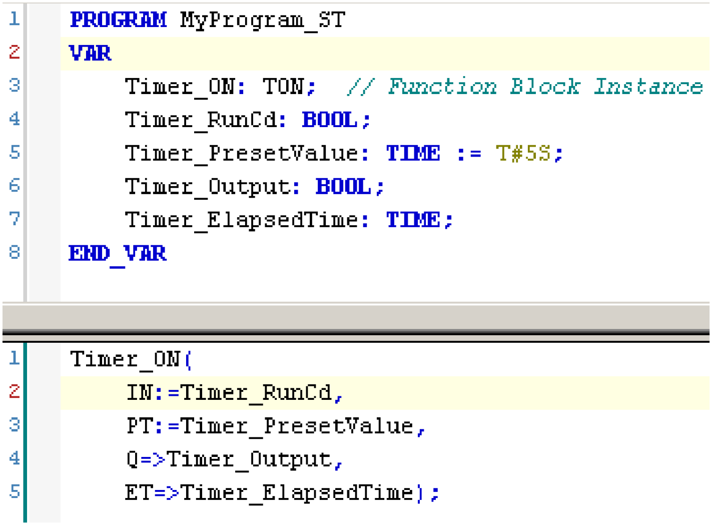
Function Block

A function block:

- is a POU (Program Organization Unit) that returns one or more outputs.
- needs to be called by an instance (function block copy with dedicated name and variables).
- each instance has a persistent state (outputs and internal variables) from one call to the other from a function block or a program.

Examples: timers, counters

In the example, `Timer_ON` is an instance of the function block `TON`:



```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;   // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

How to Use a Function or a Function Block in IL Language

General Information

This part explains how to implement a function and a function block in IL language.

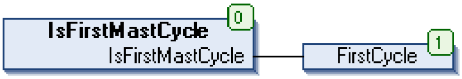
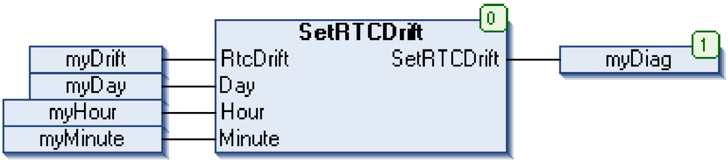
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

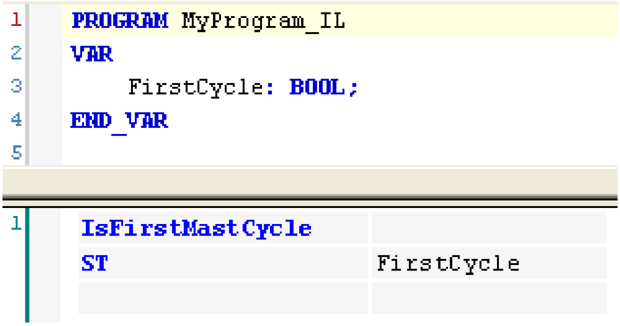
This procedure describes how to insert a function in IL language:

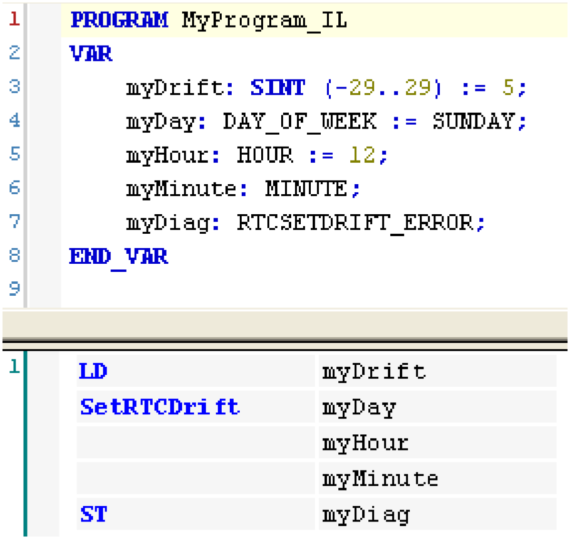
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	If the function has 1 or more inputs, start loading the first input using LD instruction.
4	Insert a new line below and: ● type the name of the function in the operator column (left field), or ● use the **Input Assistant** to select the function (select **Insert Box** in the context menu).
5	If the function has more than 1 input and when Input Assistant is used, the necessary number of lines is automatically created with `???` in the fields on the right. Replace the `???` with the appropriate value or variable that corresponds to the order of inputs.
6	Insert a new line to store the result of the function into the appropriate variable: type ST instruction in the operator column (left field) and the variable name in the field on the right.

To illustrate the procedure, consider the Functions `IsFirstMastCycle` (without input parameter) and `SetRTCDrift` (with input parameters) graphically presented below:

Function	Graphical Representation
without input parameter: `IsFirstMastCycle`	 IsFirstMastCycle 0 IsFirstMastCycle FirstCycle 1
with input parameters: `SetRTCDrift`	 SetRTCDrift 0 myDrift myDay myHour myMinute RtcDrift Day Hour Minute SetRTCDrift myDiag 1

In IL language, the function name is used directly in the operator column:

Function	Representation in POU IL Editor
IL example of a function without input parameter: `IsFirstMastCycle`	 PROGRAM MyProgram_IL VAR FirstCycle: BOOL; END_VAR IsFirstMastCycle ST FirstCycle

IL example of a function with input parameters:
`SetRTCDrift`
	
	PROGRAM MyProgram_IL
	VAR
	    myDrift: SINT (-29..29) := 5;
	    myDay: DAY_OF_WEEK := SUNDAY;
	    myHour: HOUR := 12;
	    myMinute: MINUTE;
	    myDiag: RTCSETDRIFT_ERROR;
	END_VAR
	
	LD myDrift
	SetRTCDrift myDay
	    myHour
	    myMinute
	ST myDiag

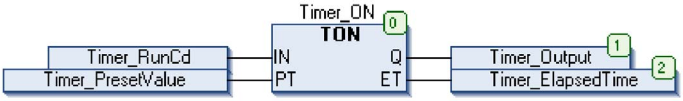
Using a Function Block in IL Language

This procedure describes how to insert a function block in IL language:

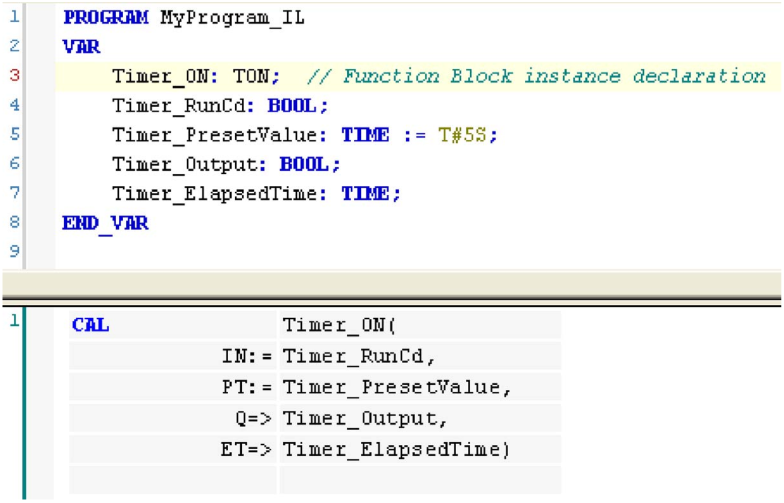
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function block requires, including the instance name.

Step	Action
3	Function Blocks are called using a CAL instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the CAL instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by ":=". ● Values to outputs are set by "=>".
4	In the CAL right-side field, replace ??? with the instance name.
5	Replace other ??? with an appropriate variable or immediate value.

To illustrate the procedure, consider this example with the TON Function Block graphically presented below:

Function Block	Graphical Representation
TON	

In IL language, the function block name is used directly in the operator column:

Function Block	Representation in POU IL Editor
TON	

```
PROGRAM MyProgram_IL
VAR
    Timer_ON: TON;  // Function Block instance declaration
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR

CAL             Timer_ON(
           IN:= Timer_RunCd,
           PT:= Timer_PresetValue,
            Q=> Timer_Output,
           ET=> Timer_ElapsedTime)
```

How to Use a Function or a Function Block in ST Language

General Information

This part explains how to implement a Function and a Function Block in ST language.

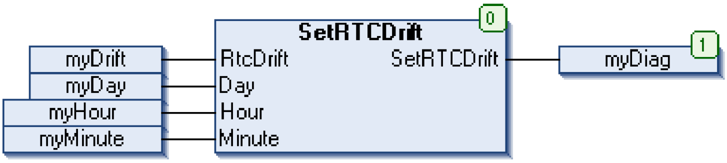
Function `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in ST Language

This procedure describes how to insert a function in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function requires.
3	Use the general syntax in the **POU ST Editor** for the ST language of a function. The general syntax is: `FunctionResult:= FunctionName(VarInput1, VarInput2,.. VarInputx);`

To illustrate the procedure, consider the function `SetRTCDrift` graphically presented below:

Function	Graphical Representation
`SetRTCDrift`	 0 SetRTCDrift myDrift myDay myHour myMinute RtcDrift Day Hour Minute SetRTCDrift myDiag 1

The ST language of this function is the following:

Function	Representation in POU ST Editor

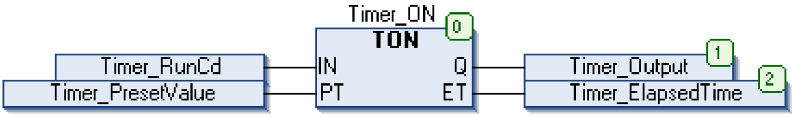
`SetRTCDrift`	`PROGRAM MyProgram_ST`
	`VAR myDrift: SINT(-29..29) := 5;`
	`myDay: DAY_OF_WEEK := SUNDAY;`
	`myHour: HOUR := 12;`
	`myMinute: MINUTE;`
	`myRTCAdjust: RTCDRIFT_ERROR;`
	`END_VAR`
	`myRTCAdjust:= SetRTCDrift(myDrift, myDay, myHour, myMinute);`

Using a Function Block in ST Language

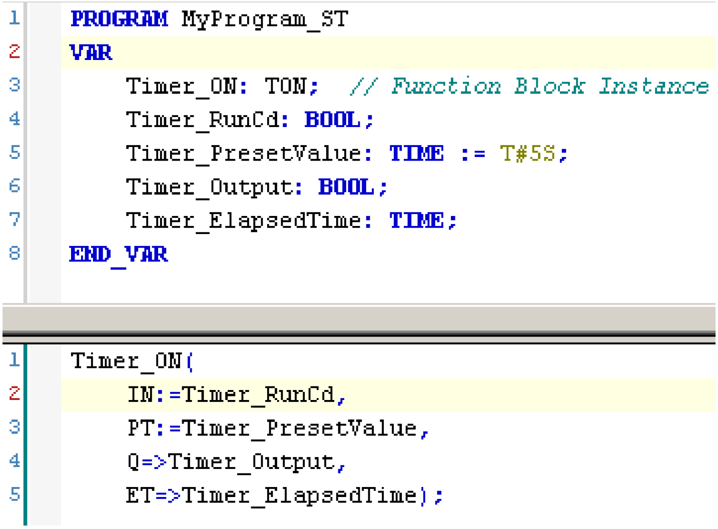
This procedure describes how to insert a function block in ST language:

Step	Action
1	Open or create a new POU in Structured Text language. **NOTE:** The procedure to create a POU is not detailed here. For more information on adding, declaring and calling POUs, refer to the related documentation *(see SoMachine, Programming Guide)*.
2	Create the input and output variables and the instance required for the function block: ● Input variables are the input parameters required by the function block ● Output variables receive the value returned by the function block
3	Use the general syntax in the **POU ST Editor** for the ST language of a Function Block. The general syntax is: `FunctionBlock_InstanceName(Input1:=VarInput1, Input2:=VarInput2,... Ouput1=>VarOutput1, Ouput2=>VarOutput2,...);`

To illustrate the procedure, consider this example with the `TON` function block graphically presented below:

Function Block	Graphical Representation
`TON`	 Timer_ON 0 TON Timer_RunCd Timer_PresetValue IN PT Q ET Timer_Output 1 Timer_ElapsedTime 2

This table shows examples of a function block call in ST language:

Function Block	Representation in POU ST Editor
TON	

```
PROGRAM MyProgram_ST
VAR
    Timer_ON: TON;  // Function Block Instance
    Timer_RunCd: BOOL;
    Timer_PresetValue: TIME := T#5S;
    Timer_Output: BOOL;
    Timer_ElapsedTime: TIME;
END_VAR
```

```
Timer_ON(
    IN:=Timer_RunCd,
    PT:=Timer_PresetValue,
    Q=>Timer_Output,
    ET=>Timer_ElapsedTime);
```

Glossary

!

%IW

According to the IEC standard, %IW represents an input word register (for example, a language object of type analog IN).

%QW

According to the IEC standard, %QW represents an output word register (for example, a language object of type analog OUT).

A

application

A program including configuration data, symbols, and documentation.

ATV

The model prefix for Altivar drives (for example, ATV312 refers to the Altivar 312 variable speed drive).

B

byte

A type that is encoded in an 8-bit format, ranging from 00 hex to FF hex.

C

CFC

(*continuous function chart*) A graphical programming language (an extension of the IEC 61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to the inputs of other blocks to create complex expressions.

CIP

(*common industrial protocol*) When a CIP is implemented in a network application layer, it can communicate seamlessly with other CIP-based networks without regard to the protocol. For example, the implementation of CIP in the application layer of an Ethernet TCP/IP network creates an EtherNet/IP environment. Similarly, CIP in the application layer of a CAN network creates a DeviceNet environment. In that case, devices on the EtherNet/IP network can communicate with devices on the DeviceNet network through CIP bridges or routers.

configuration

The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

control network

A network containing logic controllers, SCADA systems, PCs, HMI, switches, ...

Two kinds of topologies are supported:

- flat: all modules and devices in this network belong to same subnet.
- 2 levels: the network is split into an operation network and an inter-controller network.

These two networks can be physically independent, but are generally linked by a routing device.

controller

Automates industrial processes (also known as programmable logic controller or programmable controller).

D

device network

A network that contains devices connected to a specific communication port of a logic controller. This controller is seen as a master from the devices point of view.

DHCP

(*dynamic host configuration protocol*) An advanced extension of BOOTP. DHCP is more advanced, but both DHCP and BOOTP are common. (DHCP can handle BOOTP client requests.)

DTM

(*device type manager*) Classified into 2 categories:

- Device DTMs connect to the field device configuration components.
- CommDTMs connect to the software communication components.

The DTM provides a unified structure for accessing device parameters and configuring, operating, and diagnosing the devices. DTMs can range from a simple graphical user interface for setting device parameters to a highly sophisticated application capable of performing complex real-time calculations for diagnosis and maintenance purposes.

E

EDS

(*electronic data sheet*) A file for fieldbus device description that contains, for example, the properties of a device such as parameters and settings.

expansion bus

An electronic communication bus between expansion I/O modules and a controller.

F

FB

(*function block*) A convenient programming mechanism that consolidates a group of programming instructions to perform a specific and normalized action, such as speed control, interval control, or counting. A function block may comprise configuration data, a set of internal or external operating parameters and usually 1 or more data inputs and outputs.

FDR

(*fast device replacement*): A service supported by the device, that facilitate the replacement of an inoperable equipment.

function block diagram

One of the 5 languages for logic or control supported by the standard IEC 61131-3 for control systems. Function block diagram is a graphically oriented programming language. It works with a list of networks where each network contains a graphical structure of boxes and connection lines representing either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

H

health bit

Variable that indicates the communication state of the channels.

health timeout

Represents the maximal time (in ms) between a request of the Modbus IO scanner and a response of the slave.

I

I/O

(*input/output*)

IL

(*instruction list*) A program written in the language that is composed of a series of text-based instructions executed sequentially by the controller. Each instruction includes a line number, an instruction code, and an operand (refer to IEC 61131-3).

Input Assembly

Assemblies are blocks of data exchanged between network devices and the logic controller. An Input Assembly generally contains status information from a slave or Target device, read by the master or Originator.

INT

(*integer*) A whole number encoded in 16 bits.

L

LD

(*ladder diagram*) A graphical representation of the instructions of a controller program with symbols for contacts, coils, and blocks in a series of rungs executed sequentially by a controller (refer to IEC 61131-3).

M

MAC address

(*media access control address*) A unique 48-bit number associated with a specific piece of hardware. The MAC address is programmed into each network card or device when it is manufactured.

O

ODVA

(*open DeviceNet vendors association*) The family of network technologies that are built on CIP (EtherNet/IP, DeviceNet, and CompoNet).

Originator

In EtherNet/IP, the device that initiates a CIP connection for implicit or explicit messaging communications or that initiates a message request for un-connected explicit messaging.

See also *target*

Output Assembly

Assemblies are blocks of data exchanged between network devices and the logic controller. An Output Assembly generally contains command sent by the master or Originator, to the slave or Target devices.

P

post configuration

(*post configuration*) An option that allows to modify some parameters of the application without changing the application. Post configuration parameters are defined in a file that is stored in the controller. They are overloading the configuration parameters of the application.

POU

(*program organization unit*) A variable declaration in source code and a corresponding instruction set. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

program

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

R

repetition rate

Polling interval of the Modbus request that is sent.

RJ45

A standard type of 8-pin connector for network cables defined for Ethernet.

RPI

(*requested packet interval)* The time period between cyclic data exchanges requested by the scanner. EtherNet/IP devices publish data at the rate specified by the RPI assigned to them by the scanner, and they receive message requests from the scanner with a period equal to RPI.

S

ST

(*structured text*) A language that includes complex statements and nested instructions (such as iteration loops, conditional executions, or functions). ST is compliant with IEC 61131-3.

T

Target

In EtherNet/IP, a device is considered to be the target when it is the recipient of a connection request for implicit or explicit messaging communications.

See also *Originator*

TVDA

(*tested validated documented architectures*) Control system proposals based on Schneider Electric components.TVDAs cover a wide range of machine types and consider machine performance requirements, installation constraints, and target costs. To optimize the implementation effort, each TVDA comes with a detailed component list, wiring diagrams, and commissioning guide, as well as controller and HMI applications to control components of the system.

U

UL

(*underwriters laboratories*) A US organization for product testing and safety certification.

V

variable

A memory unit that is addressed and modified by a program.

Index