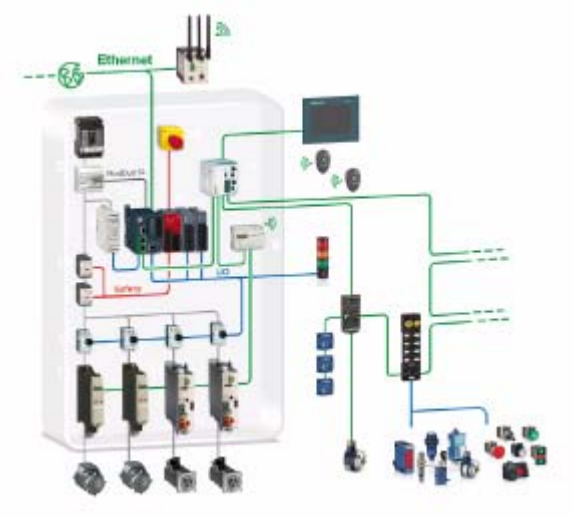
SoMachine
Modbus TCP IOScanner
User Guide

09/2014



EIO0000001949.00

www.schneider-electric.com

Schneider Electric

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, 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 your Modbus TCP IOScanner network.
- Install and configure your Modbus TCP IOScanner network.
- Operate and maintain your Modbus TCP IOScanner 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 with the release of SoMachine V4.1 Modbus TCP IOScanner add-on.

The technical characteristics of the devices described in this manual also appear online.

The characteristics that are presented in this manual should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the manual and online information, use the online information as your reference.

Related Documents

Title of Documentation	Reference Number
Modicon M251 Logic Controller - Programming Guide	EIO0000001462 (ENG), EIO0000001463 (FRE), EIO0000001464 (GER), EIO0000001465 (SPA), EIO0000001466 (ITA), EIO0000001467 (CHS)
SoMachine - Programming Guide	EIO0000000067 (ENG); EIO0000000069 (FRE); EIO0000000068 (GER); EIO0000000071 (SPA); EIO0000000070 (ITA); EIO0000000072 (CHS)
Essential Guide: Networks, connectivity and Web servers	DIA6ED2130205EN (ENG)

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

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.

Chapter 1
Modbus TCP IOScanner Presentation

What Is in This Chapter?

This chapter contains the following topics:

Modbus TCP IOScanner Overview

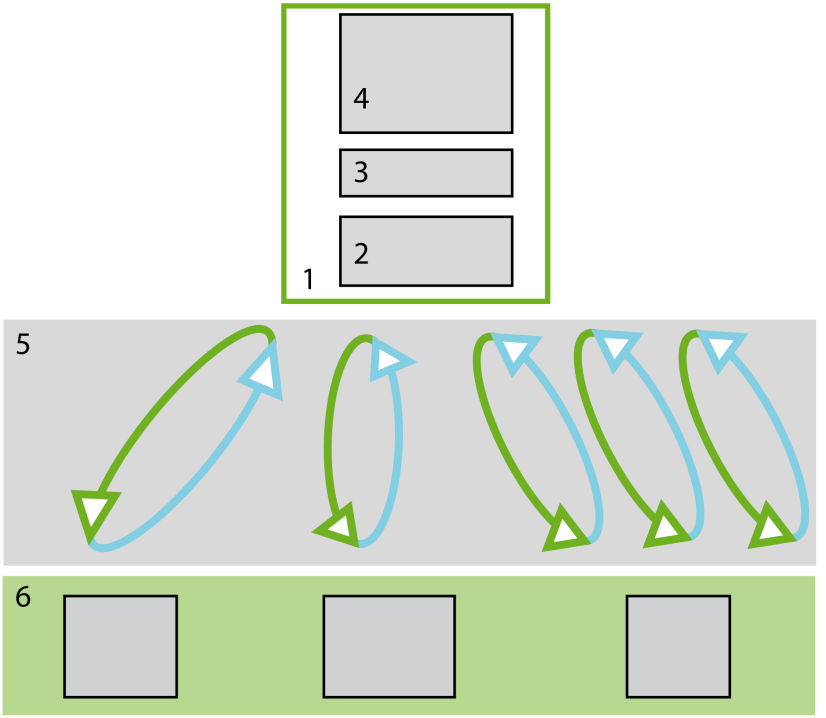
Presentation

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.

The Modbus TCP IOScanner relies on the Modbus TCP standard. The core of this standard is a master/slave network model. The unique master is the controller.

The communication with the slaves is accomplished using Modbus TCP channels *(see page 14)*.

Principle



1 Controller
2 I/O images
3 Application interface *(see page 43)*
4 Application
5 Modbus channels *(see page 14)*
6 Slave devices *(see page 14)*

System Architecture

The Modbus TCP IOScanner relies on:

- Ethernet network including the controller, the slaves and the infrastructure equipment *(see page 12)*,
- Software configuration *(see page 13)*.

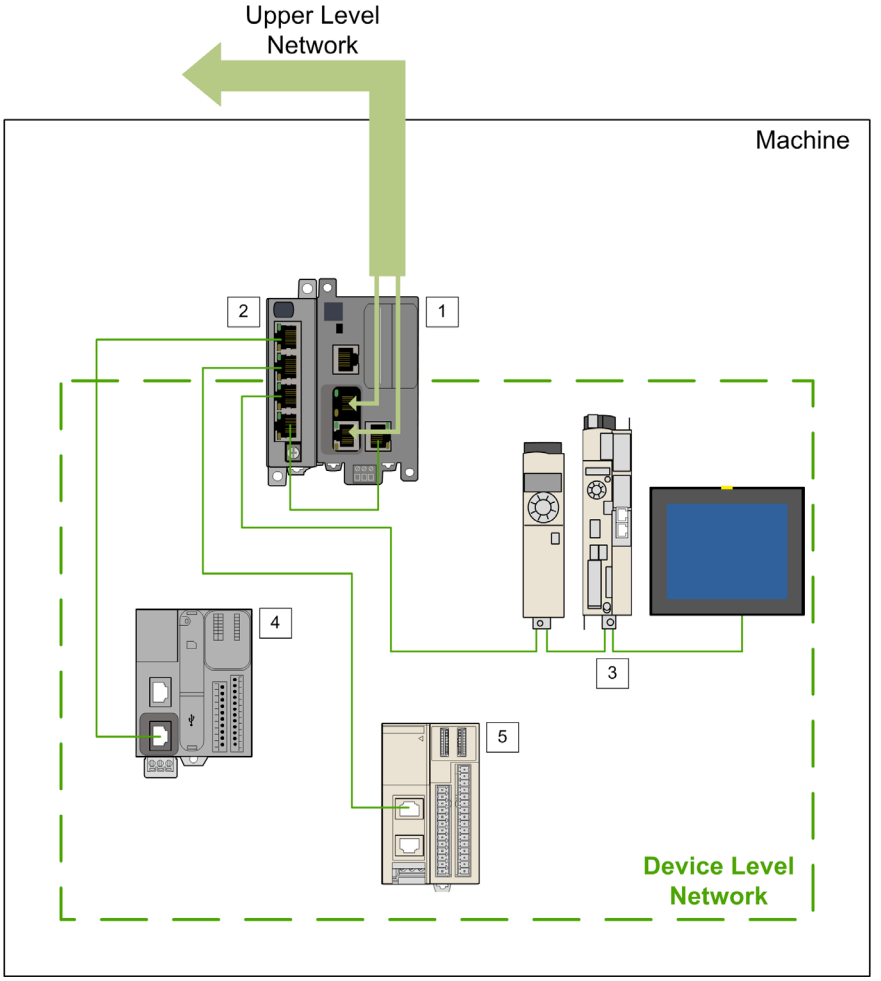
Controller Compatibility

The Modbus TCP IOScanner service is available on the TM251MESE controller.

Architecture

Ethernet Network

This figure presents a typical Modbus TCP IOScanner architecture.



1 Controller / Modbus master
2 TM4ES4 used as a standalone Ethernet switch
3 Daisy chained slaves
4 Modbus slave
5 I/O island

The controller is connected to the upper level network as well as to the device level network.

The device level network is controlled by the Modbus TCP IOScanner.

The controller can be used as a gateway *(see page 20)* between the two networks.

Software Configuration

The network area of slaves and each I/O is configured by software:

- The communication configuration defines the addressing and communication periods.
- The device configuration defines the device behavior.

Slave configuration enables variable attribution in order to optimize the monitoring.

I/O configuration adjusts the monitoring quality according to the network bandwidth.

You can set several parameters *(see page 15)* to optimize the performance.

Optional Services

The Modbus TCP IOScanner can be associated to several optional services:

- DHCP: Dynamic Host Configuration Protocol server assigns an IP address to a slave device when it requests one.
- FDR: Fast Device Replacement server configures a replaced remote device controlled by the Modbus TCP IOScanner without stopping the application.
- Web server *(see page 49)*.

Principles

Overview

The Modbus TCP IOScanner reads inputs and writes outputs of the slave devices.

The communication between Modbus TCP IOScanner and the slave devices is accomplished using Modbus channels.

The communication in the Modbus TCP IOScanner is configured with the SoMachine software.

Slaves Types

There are three different types of Modbus TCP IOScanner slaves in the SoMachine software:

- **Advantys OTB slave** devices are used for remote digital and analog I/Os.
  Use SoMachine software for the specific configuration of the device and of the associated I/O modules.
- **Predefined slave** devices are common Modbus devices coming with predefined set of communication parameters.
  Use a dedicated software and/or a local HMI to configure the devices. With the FDT/DTM technology, predefined slave devices with advanced settings can be configured in SoMachine, refer to the Device Type Manager User Guide.
- **Generic slave** devices are used for all other Modbus slave devices.
  The entire device configuration is done with a third-party software and/or a local HMI. With the FDT/DTM technology, some devices can be configured in SoMachine, refer to the Device Type Manager User Guide.

Modbus 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.

Communication Configuration Parameters

Configure each slave in the Modbus TCP IOScanner network using SoMachine software.

This table presents the communication configuration parameters:

Parameter	Description
IP address	The IP address of the slave in the Ethernet network.
Health timeout	Time value expressed in ms. If the Modbus TCP IOScanner does not detect a reply from the slave after this delay, an error state occurs.
Repetition rate	Time value expressed in ms. This represents the delay between two sendings of a request. This value must be lower than the Health timeout.
Channel ID	Unique identifier for a channel. This value is automatically created by the SoMachine software when a new channel is added. You can read this value in the **Modbus TCP Channel Configuration** tab or, for the Advantys OTB slave, in the **Modbus TCP Slave Configuration** tab.

Chapter 2
Network Installation

What Is in This Chapter?

This chapter contains the following topics:

Network Planning

Purpose

A planned network increases the installation efficiency as well as decreases the installation time and costs. The interfacing of materials (switches, cables, ports) must be preliminarily designed in order to plan the network.

Network Design

To design and plan the Modbus TCP IOScanner 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 on 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

NOTE: Do not use a hub to set up a Modbus TCP IOScanner network.

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.

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

In standard installation, you can use these cables:

Reference	Description	Details	Length
490NTW000••	Ethernet shielded cable for DTE connections	Standard 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		Standard 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 environment, equipped with RJ45 connectors at each end. CE compliant	1, 2, 3, 5, or 10 m (3.28, 6.56, 9.84, 16.4, 32.81 ft)
TCSECU3M3M••S4		Cable for harsh environment, equipped with RJ45 connectors at each end. UL compliant	1, 2, 3, 5, or 10 m (3.28, 6.56, 9.84, 16.4, 32.81 ft)

In case of fiber optic network, 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)

IP Address Configuration

Prerequisites

Each device in the Modbus TCP IOScanner network obtains its own IP address. All IP addresses must be unique.

NOTE: Assign Class C IP addresses for Modbus TCP IOScanner network.

Configure the IP addresses in two stages:

Stage	Description
1	Using the SoMachine software, configure the controller port supporting the Modbus TCP IOScanner: ● IP address, ● Subnet mask, ● Default gateway. **NOTE:** Only use fixed addresses. **NOTE:** Do not use the post configuration file to address the controller ports.
2	Configure each slave device of the Modbus TCP IOScanner network: ● IP address, ● Subnet mask, ● Default gateway.

Slave Address Assignment

In the Modbus TCP IOScanner network, assign the IP address of the slaves using the following method (depending on the device type):

- By DHCP server, to manage all the IP addresses of the Modbus TCP IOScanner network from the SoMachine software, or if you need the FDR service,
- By third-party software, or local HMI,
- By an advanced settings configuration on Modbus serial line, through the FDT/DTM technology, refer to the Device Type Manager User Guide.

FDR Service

Some slaves support the FDR (Fast Device Replacement) 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 slave, refer to the slave device documentation.

Slave Master IP Address Parameter

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

If the device...	Then ...
Is configured to use the Modbus TCP IOScanner	Configure the **Master IP address** parameter inside the device, see below.
Is not configured to use the Modbus TCP IOScanner	Use `0.0.0.0` for the **Master IP address** parameter in the device.

The **Master IP address** parameter of the slave has to be set to the IP address of the controller supporting the Modbus TCP IOScanner (**Ethernet 2** port).

To configure this parameter in the slave, refer to the slave documentation.

Slave Gateway Parameter

The gateway parameter of the slaves has to be set to the IP address of the Ethernet port of the controller supporting the Modbus TCP IOScanner (**Ethernet 2**).

A configuration tool has to reach the slaves in order to set their parameters.

If the configuration tool...	Then...
Is connected on the upper level network	Update the slave gateway parameter, see below.
Is connected on the device level network	The gateway parameter is not used.
Uses a protocol other than TCP/IP	The gateway parameter is not used.

To configure this parameter in the slave, refer to the slave documentation.

NOTE: If the DHCP service is used to address the slaves, the gateway parameter is set in the controller DHCP table.

PC Routing

The PC supporting the configuration tool must be configured in order to communicate with the slaves.

If the slave is configured...	Then...
As a pre-defined slave through advanced settings (FDT/DTM)	No specific computer parameterization is needed. **NOTE:** The computer configuration is not altered.
Through another tool	Update the PC routing table, 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 `route ADD Destination MASK Subnet_Mask Gateway` command.

This table represents the parameters to update in this command syntax:

Parameter	Value
Destination	IP address of the Modbus TCP IOScanner network.
Subnet_Mask	Subnet mask of the Modbus TCP IOScanner network.
Gateway	Address of the controller port connected to the upper level network.

To verify these parameters in a Windows command prompt, execute the `route PRINT` command.

To remove this route from the PC, in a Windows command prompt, execute the `route DELETE Destination` command where `Destination` is the IP address of the Modbus TCP IOScanner network entered beforehand.

Network Tests

Purpose

Before operating the Modbus TCP IOScanner, test the network.

Verify the following:

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

Some usual testing methods are presented below.

Status LED

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

Verification Using a Computer

With a computer, verify that each slave device is connected and addressed:

Step	Action
1	Connect the computer in the Modbus TCP IOScanner network.
2	Access the command prompt.
3	Use a `ping xxx.xxx.xxx.xxx` command to reach each slave. `xxx.xxx.xxx.xxx` = IP address of the slave to test. **NOTE:** The command `ping -h` displays the help for the `ping` command.

Verification Using a Web Server

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

Step	Action
1	Access the controller Web server.
2	Open the **Ethernet Diagnostic** page.
3	Use the **Remote ping** function *(see page 49)* on each slave device.

Chapter 3
Modbus TCP IOScanner Configuration

What Is in This Chapter?

This chapter contains the following topics:

Adding a Slave on the Modbus TCP IOScanner

Overview

This section describes how to add a slave on the **Modbus TCP IOScanner**.

These slaves are divided in 3 categories:

- Pre-defined devices for Schneider Electric Modbus devices (ATV, LXM, and ZBRN),
- Advantys OTB for Modbus TCP module with configurable I/Os,
- Generic devices for all other Modbus TCP slaves.

For each generic device, you must define the Modbus requests to send to this device by adding channels. A channel corresponds to a Modbus request and can have its own repetition rate.

Add a Slave on the Modbus TCP IOScanner

To add a slave on the **Modbus TCP IOScanner**, select the chosen device in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the **Ethernet_2** node of the **Devices tree**.

For more information on adding a slave 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)*

Configuring a Modbus TCP IOScanner

Prerequisites

Before configuring the Modbus TCP IOScanner:

- Set the IP address of the Ethernet 2 to **fixed mode**. It must be different from `0.0.0.0`.
- The connected devices must be in the same subnet as the Ethernet 2 port

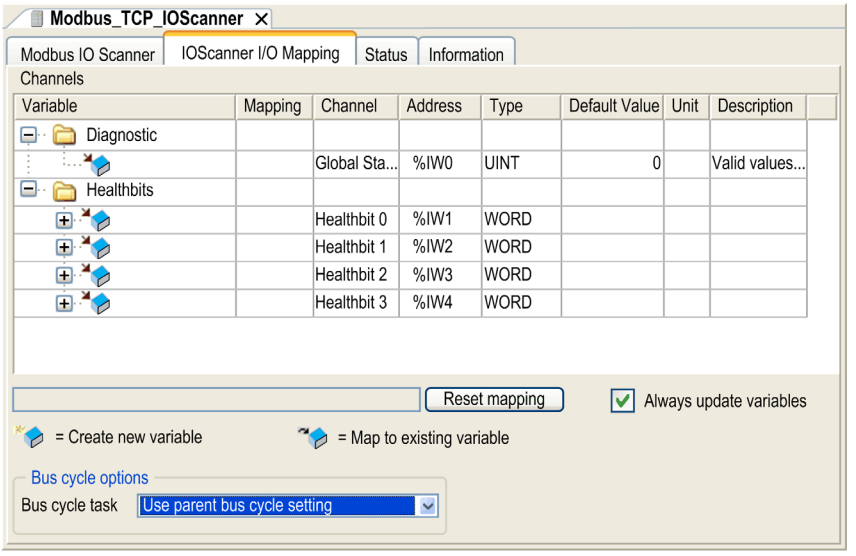
For more information on IP address, refer to Ethernet Configuration *(see Modicon M251 Logic Controller, Programming Guide)*.

Add a Modbus TCP IOScanner

The Modbus TCP IOScanner node is automatically added when a slave is added on the **Ethernet 2** node *(see page 26)*.

Configure a Modbus TCP IOScanner

To configure a Modbus TCP IOScanner, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **Modbus_TCP_IOScanner**. **Result**: The configuration window is displayed.
2	Select the **IOScanner I/O Mapping** tab. 

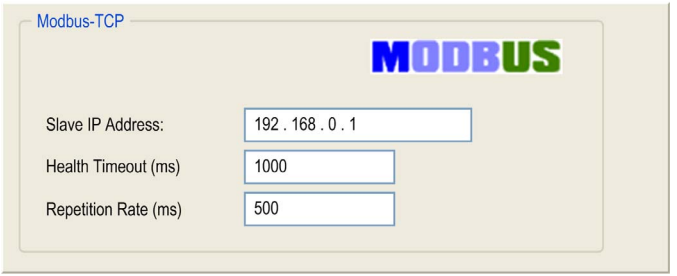
Step	Action
3	Select the **Bus cycle task** in the list: ● **Use parent bus cycle setting** (by default), ● **MAST**, or ● An existing task of the project. **NOTE:** The **Bus cycle task** parameter inside the I/O mapping editor of the device that contains the Modbus TCP IOScanner defines the task responsible 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.

NOTE: When the Modbus TCP IOScanner is configured, the post configuration file for the Ethernet 2 network is ignored.

Configuring an Advantys OTB Distributed I/O Module on the Modbus TCP IOScanner

Configure an OTB Slave

To configure the OTB slave added on the **Modbus TCP IOScanner**, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click the Advantys OTB device node. Result: The configuration window is displayed. 
2	In the **Slave IP Address** field, enter the IP address assigned to the Advantys OTB.
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 27)* or through the Web server. The health timeout applies to all the channels of the slave.
4	Enter a **Repetition Rate (ms)** value (by default 20). The **Health Timeout (ms)** value must be greater than the **Repetition Rate (ms)** value.
5	Configure the I/Os of the Advantys OTB device in the **OTB I/O Configuration** tab.
6	Add and configure TM2 expansion modules attached to the OTB.
7	Call a `CONFIGURE_OTB` function block *(see page 60)* 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 Modbus TCP IOScanner.

TM3 Modules Compatibility

TM3 modules are not compatible with the Advantys OTB.

TM2 Modules Compatibility

This table lists the TM2 modules compatible with the Advantys OTB:

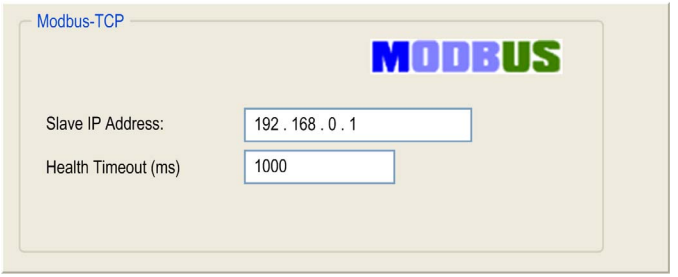
Reference	Type
TM2AMI2HT	2 analog inputs
TM2AMI2LT	2 analog inputs
TM2AMI4LT	4 analog inputs
TM2AMI8HT	8 analog inputs
TM2ARI8HT	8 analog inputs
TM2ARI8LRJ	8 analog inputs
TM2ARI8LT	8 analog inputs
TM2AMO1HT	1 analog output
TM2AVO2HT	2 analog outputs
TM2AMM3HT	2 analog inputs 1 analog output
TM2AMM6HT	4 analog inputs 2 analog outputs
TM2ALM3LT	2 analog inputs 1 analog output
TM2DAI8DT	8 digital inputs Signal type: AC type
TM2DDI8DT	8 digital inputs Signal type: Sink/Source
TM2DDI16DT	16 digital inputs Signal type: Sink/Source
TM2DDI16DK	16 digital inputs Signal type: Sink/Source
TM2DDI32DK	32 digital inputs Signal type: Sink/Source
TM2DRA8RT	8 contacts in 1 common line Output type: relay (NO contacts)
TM2DRA16RT	16 contacts in 2 common lines Output type: relay (NO contacts)
TM2DDO8UT	8 transistor outputs in 1 common line Signal type: Sink
TM2DDO8TT	8 transistor outputs in 1 common line Signal type: Source
TM2DDO16UK	16 transistor outputs in 1 common line Signal type: Sink

Reference	Type
TM2DDO16TK	16 transistor outputs in 1 common line Signal type: Source
TM2DDO32UK	32 transistor outputs in 2 common lines Signal type: Sink
TM2DDO32TK	32 transistor outputs in 2 common lines Signal type: Source
TM2DMM8DRT	4 digital inputs Signal type: Sink/Source 1 common line with 4 contacts Output type: relay (NO contacts)
TM2DMM24DRF	16 digital inputs Signal type: Sink/Source 2 common lines with 8 contacts each Output type: relay (NO contacts)

Configuring a Pre-Defined Slave on the Modbus TCP IOScanner

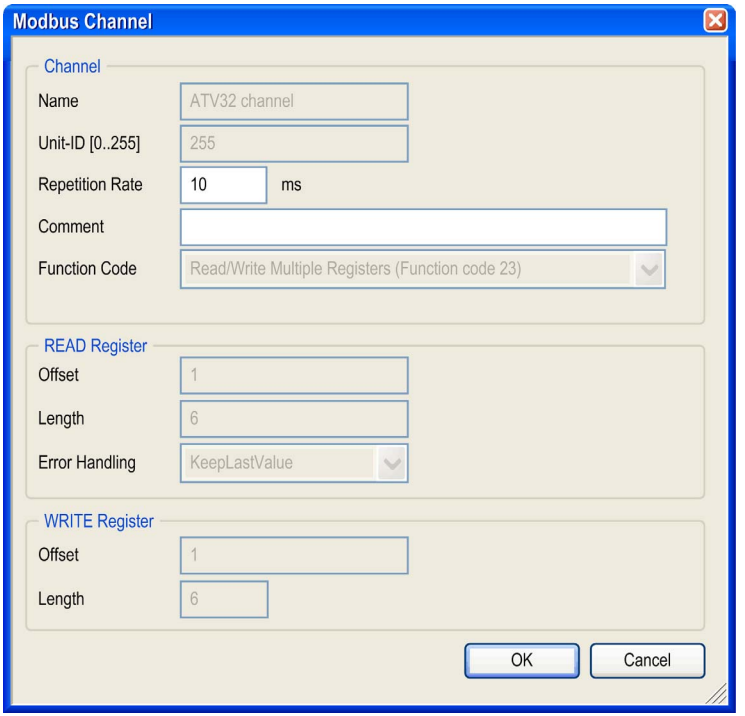
Configure a Pre-Defined Slave Added on the Modbus TCP IOScanner

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

Step	Action
1	In the **Devices tree**, double-click the added slave node. Result: The configuration window is displayed. 
2	In the **Slave IP Address**, enter the IP address of the Modbus slave.
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 27)* or through the Web server. The health timeout applies to all the channels of the slave.
4	For devices with advanced settings, some additional settings can be required. Refer to the Device Type Manager User Guide.

Edit the Modbus TCP Channel

To edit Modbus channel parameters for a pre-defined slave, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click the added slave node.
2	Select the **Modbus TCP Channel Configuration** tab and click the **Edit...** button. Result: The **Modbus Channel** window is displayed. 
3	Enter the **Repetition Rate** for the channel. The repetition rate is the polling interval of the Modbus requests.
4	You can enter a **Comment** for the channel.
5	Click **OK**.

Configuring a Generic Device on the Modbus TCP IOScanner

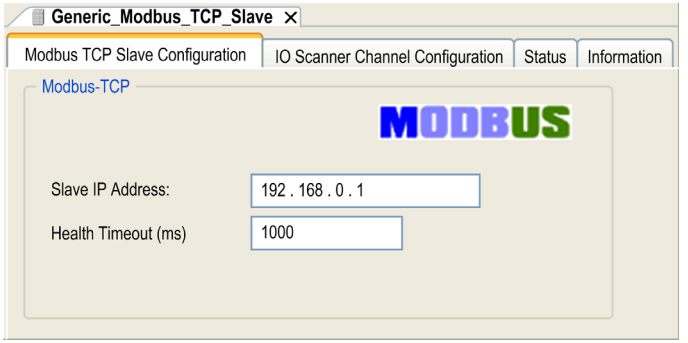
Overview

To configure a generic device added on the Modbus TCP IOScanner, complete the parameters in these two tabs:

- **Modbus TCP Slave Configuration**
- **IO Scanner Channel Configuration**

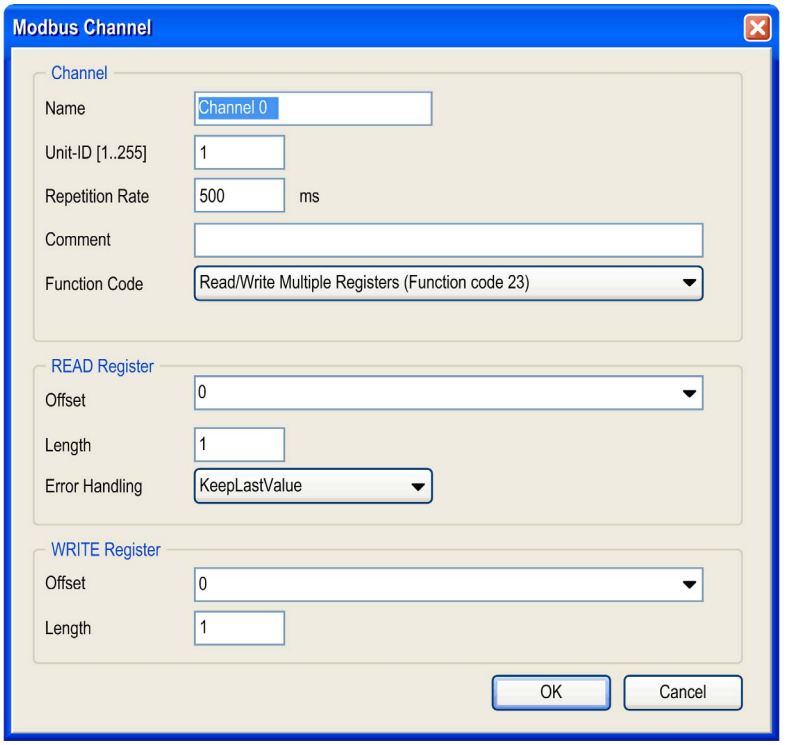
Modbus TCP Slave Configuration Tab

To configure the parameters in the **Modbus TCP Slave Configuration** tab, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **Generic_Modbus_TCP_Slave**. Result: The configuration window is displayed. 
2	Enter a **Slave IP Address** value (by default 192.168.0.1).
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 27)*.

IO Scanner Channel Configuration Tab

To configure the parameters in the **IO Scanner Channel Configuration** tab, proceed as follows:

Step	Action
1	Click the **IO Scanner Channel Configuration** tab: 
2	To remove a channel, select it and click **Delete**.
3	To change the parameters of a channel, select the channel and click **Edit**.
4	To add a channel, click **Add Channel**. This dialog box is displayed: 

Step	Action

5	In the **Channel** area, you can define:
	● **Name**: optional string for naming the channel
	● **Unit-ID [1..255]**: unit ID of the Modbus TCP slave device (by default 255). See note.
	● **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 all 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).

6	Click **OK** to validate the configuration of the channel.
7	Repeat steps 4 to 6 to create other channels that define the Modbus communication with the device. For each Modbus request, you must create a channel.

NOTE: 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.

Chapter 4
Modbus TCP IOScanner Operation

What Is in This Chapter?

This chapter contains the following topics:

Modbus TCP IOScanner Resource Verification

Purpose

The **Modbus TCP IO Scanner Resources** tab allows estimating the load on the Modbus TCP IOScanner functionality. 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
- number of channels
- repetition rate

Load Estimation

This equation allows estimating the load on the Modbus TCP IOScanner component:

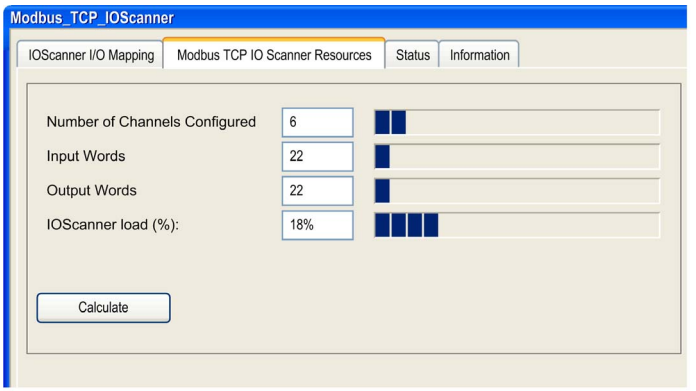
$$\text{IOScanner load (\%)} = \sum_{\text{channel}=1}^{\text{Nb Channels}} \frac{50}{\text{Repetition Rate}_{\text{channel}}}$$

In the SoMachine software, an automatic load calculation is available:

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

Description

This picture presents the **Modbus TCP IO Scanner Resources** tab:



NOTE: Load must be lower than 100%

Modbus TCP IOScanner Operating Modes

Modbus TCP IOScanner States

The Modbus TCP IOScanner state define the behavior of the different devices in the Modbus TCP IOScanner network. For each state, monitoring information (health bit, communication states, and so on) is specific.

The Modbus TCP IOScanner state depends on the controller state:

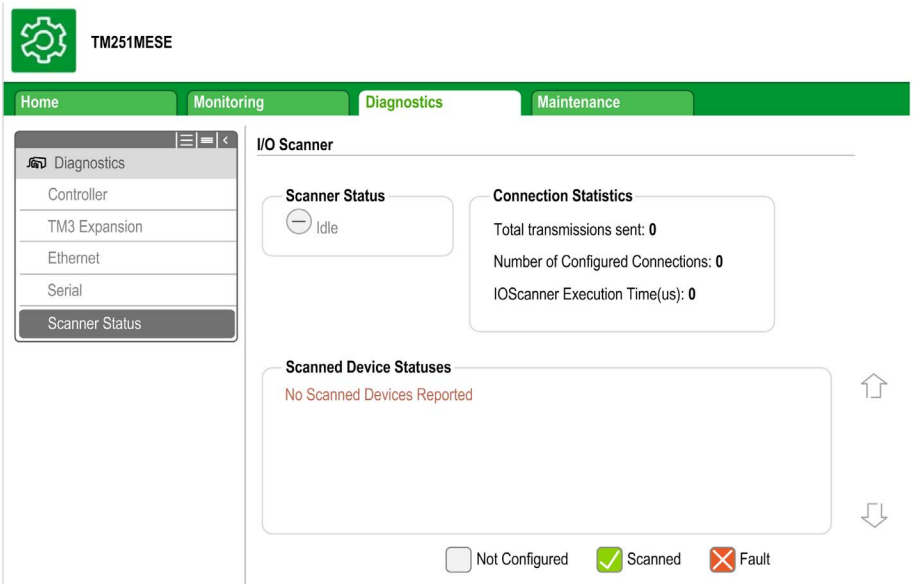
Controller state	Modbus TCP IOScanner state
EMPTY	IDLE
STOPPED	STOPPED
HALT	STOPPED
RUNNING	OPERATIONAL
RUNNING with breakpoint	OPERATIONAL with a specific behavior

Controller EMPTY State

TCP/IP connections are closed.

Slave device states are managed according to their individual mode of operation. In EMPTY state, the Modbus TCP IOScanner is not created. Therefore, there are neither Health bits nor I/O images available.

This picture presents the Web server page in this state:



Controller STOPPED State

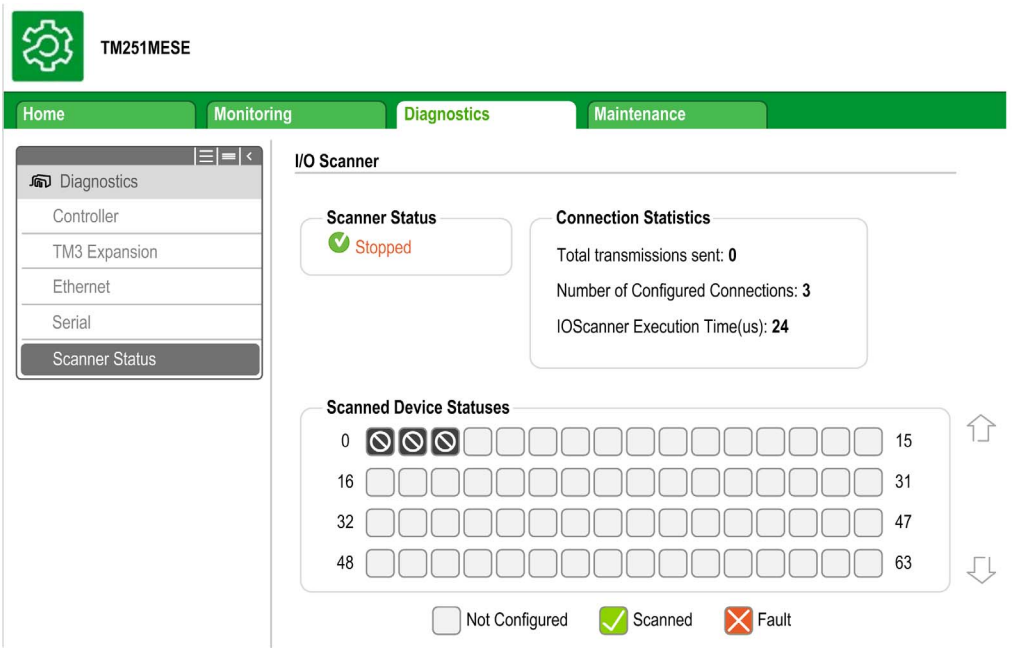
TCP/IP connections are closed. When the Modbus TCP IOScanner switches from OPERATIONAL state to STOPPED state, all connections with the slaves are closed in half-sided mode.

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

This table presents the SoMachine variables:

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 controller entered the STOPPED state and therefore may not reflect the actual state of the input thereafter.

This picture presents the Web server page in this state:



Controller HALT State

Same behavior as the controller STOPPED state.

Controller RUNNING State

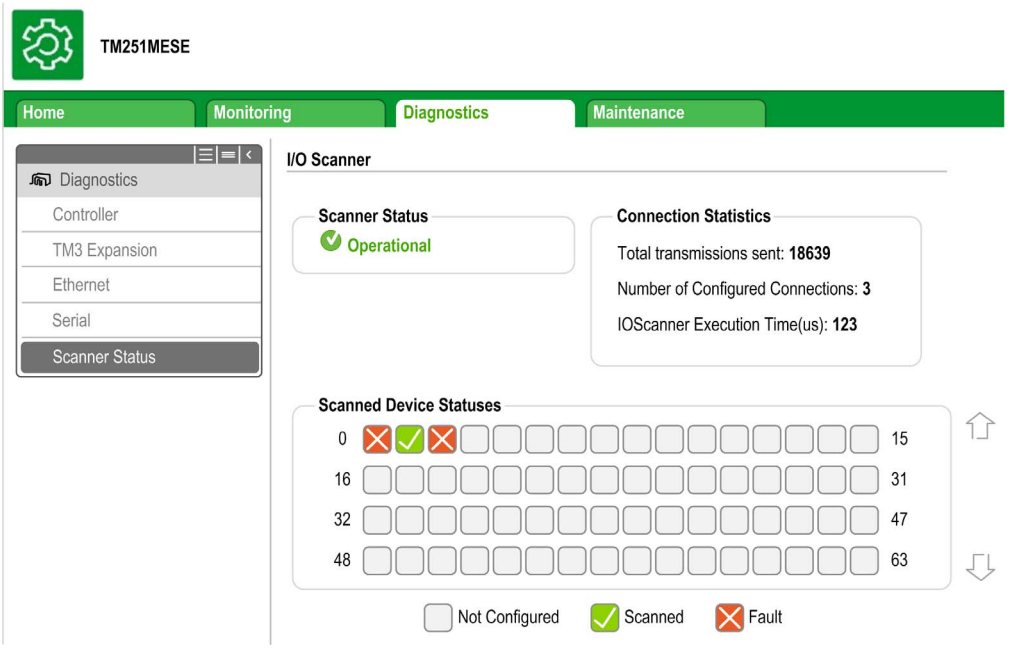
TCP/IP connections are open.

Slave devices are managed by the controller.

This table presents the SoMachine variables:

Variable	Value	Comments
Health bit value	0...1	0: No reply from the slave 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 *(see page 43)* which drives the Modbus TCP IOScanner.

This picture presents the Web server page in this state:



Controller RUNNING with Breakpoint State

TCP/IP connections are open.

Slave devices are managed by the controller.

This table presents the SoMachine variables:

Variable	Value	Comments
Health bit value	1	-
Input image	Last read value	Input values are those when the controller entered the RUNNING with breakpoint state and therefore may not reflect the actual state of the input thereafter.

Application Interface

Overview

The application interface is a set of functions and variables that enable the communication between the application and the Modbus TCP IOScanner:

- Bus cycle task
- Status variables
- I/O image variables
- Function blocks

Bus Cycle Task

The Modbus TCP IOScanner and the application exchange data at each cycle of an application task.

The **Bus Cycle Task** parameter enables you to select the application task that manages the Modbus TCP IOScanner:

- **Use parent bus cycle setting**: associate the Modbus TCP IOScanner with the application task that manages the controller.
- **MAST**: associate the Modbus TCP IOScanner with the MAST task.
- Another existing task: you can create a task and associate it to the Modbus TCP IOScanner.

For more information about the application tasks, refer to the SoMachine Programming Guide.

Status Variables

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.

I/O Image Variables

The Modbus TCP IOScanner collects and writes data from/to the slave devices. These variables are called I/O images.

Variables Addresses

Each variable gets its own address:

Variable	Type	Amount
I/O image	**%IW** for inputs **%QW** for outputs	One array of words is created per channel.
Health bit	**%IW**	Four consecutive words
Global scanner status	**%IW**	One word

The I/O mapping *(see page 47)* organizes the variables:

- Verifying the addresses
- Assigning names to the variables (mapping)

Function Blocks to Control the Modbus TCP IOScanner

Several function blocks *(see page 55)* are used by the application to communicate with the controller and the slaves:

- `CONFIGURE_OTB`
- `IOS_GETSTATE`
- `IOS_START`
- `IOS_GETHEALTH`
- `IOS_STOP`

Function Blocks to Control ATV and Lexium Devices

Use the PLC Open function blocks to control ATV and Lexium devices. These function blocks can be accessed in the Modbus TCP Altivar library. For more information, refer to the ATV Modbus TCP Function blocks Library Guide and LXM Modbus TCP Function blocks Library Guide.

Chapter 5
Modbus TCP IOScanner Maintenance

What Is in This Chapter?

This chapter contains the following topics:

Diagnostics: SoMachine Online Mode

Overview

In online mode, you can monitor the Modbus TCP IOScanner in SoMachine using the following methods:

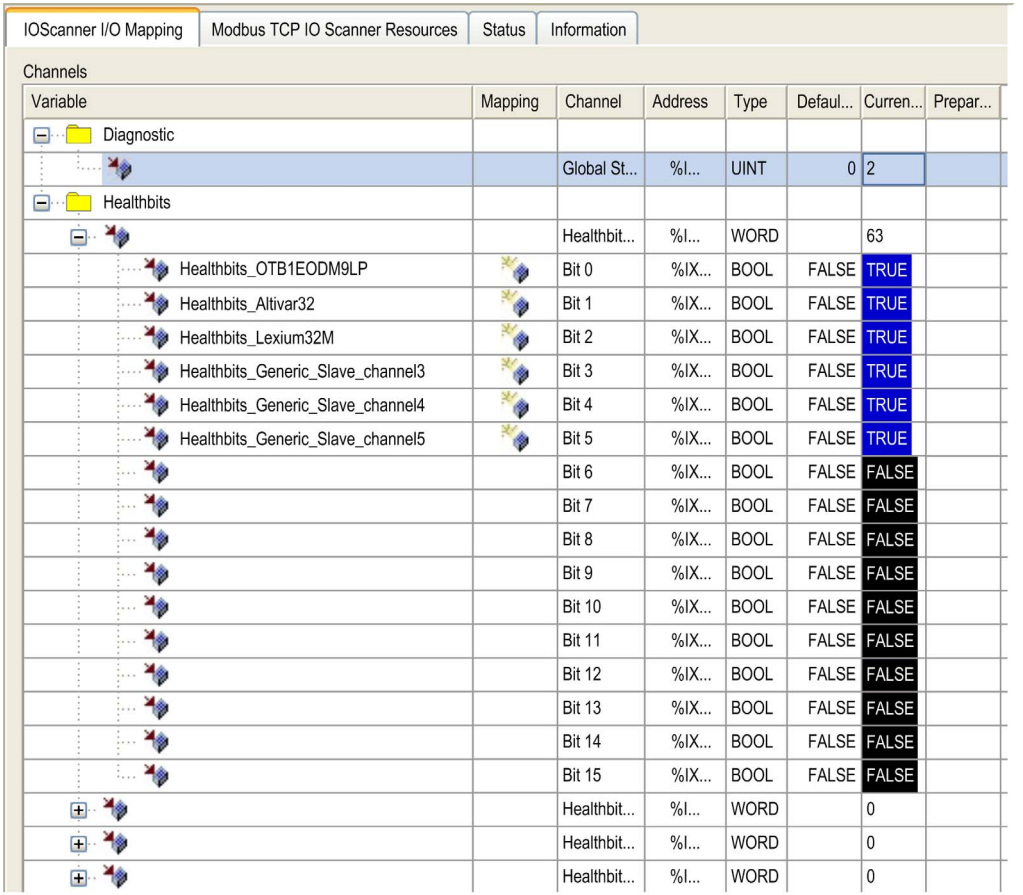
- Icons in the **Devices tree**
- Status tab of the Modbus TCP IOScanner and the different slave devices
- I/O mapping tab of the Modbus TCP IOScanner
- I/O mapping tabs of the slave devices
- Modbus TCP IOScanner resources tab

Devices Tree

The communication status of the Modbus TCP IOScanner and the slaves is presented with icons in the **Devices Tree**:

Icon	Meaning
	The communication with the device is ok. **NOTE:** Modbus TCP IOScanner is always presented with this icon.
	The controller is unable to communicate with the device. **NOTE:** When the Modbus TCP IOScanner is STOPPED, all devices are presented with this icon.

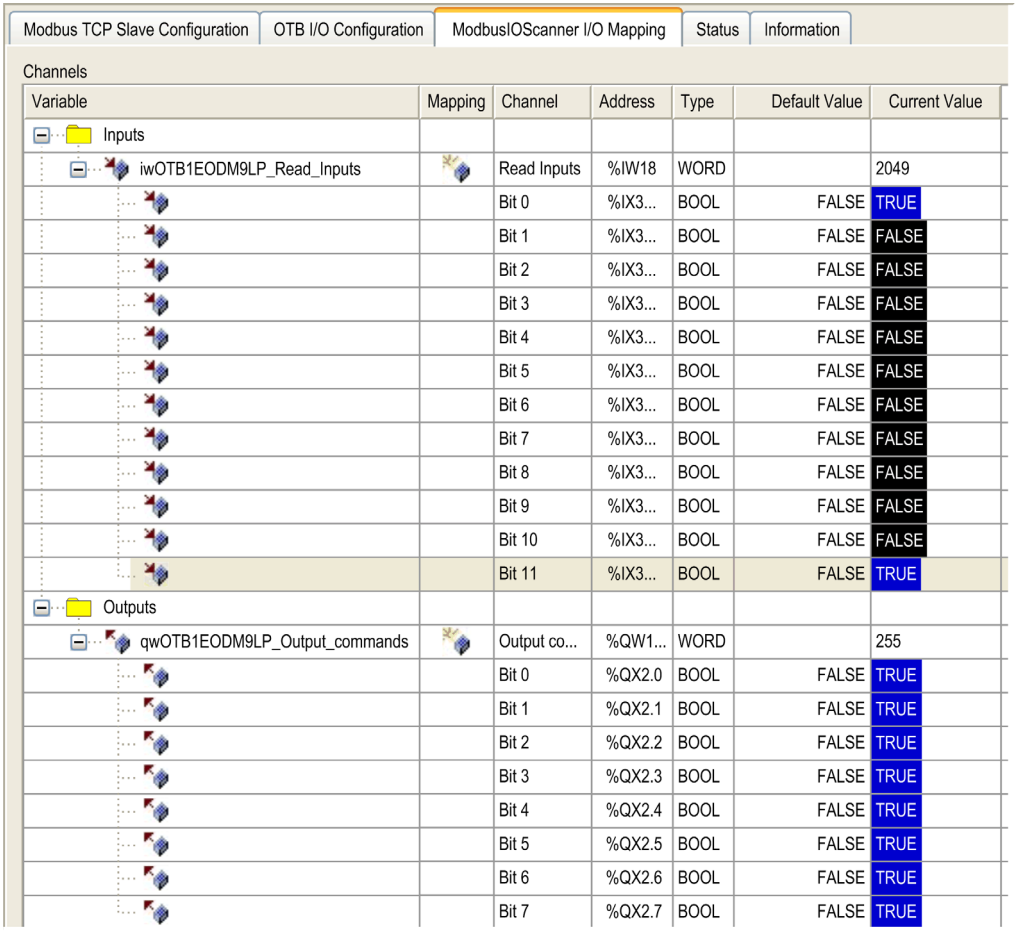
Modbus TCP IOScanner Mapping



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.
Current value		Monitor the Modbus TCP IOScanner network	For boolean values (health bit): • **TRUE** = 1 • **FALSE** = 0

Slave Mapping

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

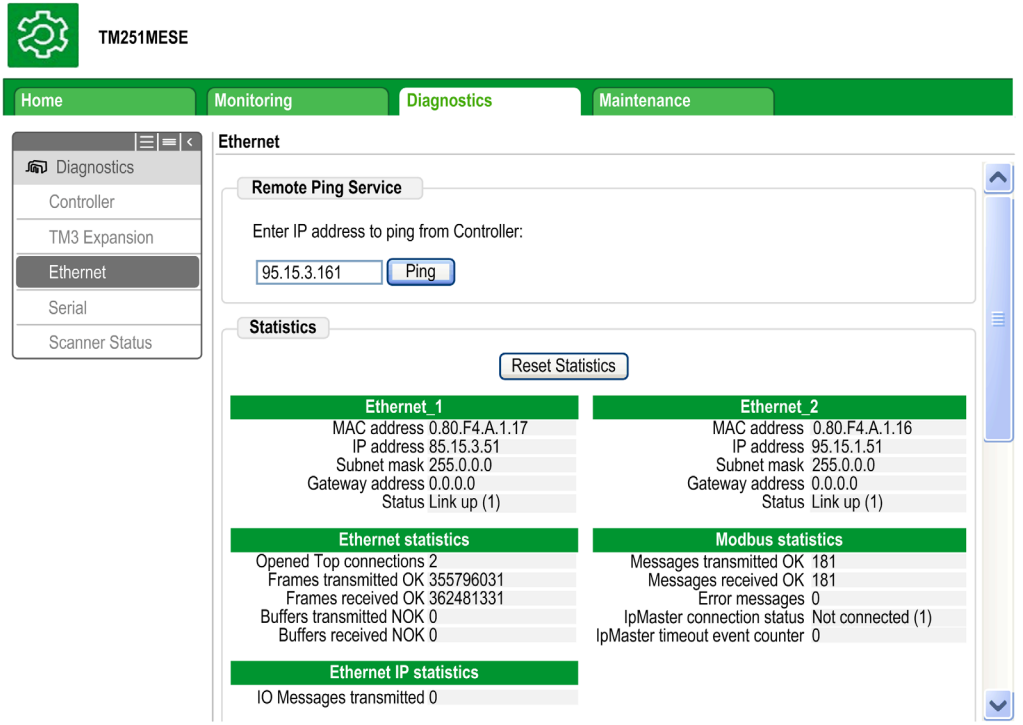


Variable	Mapping	Channel	Address	Type	Default Value	Current Value
Inputs						
iwOTB1EODM9LP_Read_Inputs		Read Inputs	%IW18	WORD		2049
		Bit 0	%IX3...	BOOL	FALSE	TRUE
		Bit 1	%IX3...	BOOL	FALSE	FALSE
		Bit 2	%IX3...	BOOL	FALSE	FALSE
		Bit 3	%IX3...	BOOL	FALSE	FALSE
		Bit 4	%IX3...	BOOL	FALSE	FALSE
		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.	For boolean values (each bit): • **TRUE** = 1 • **FALSE** = 0

Diagnostics: Web Server

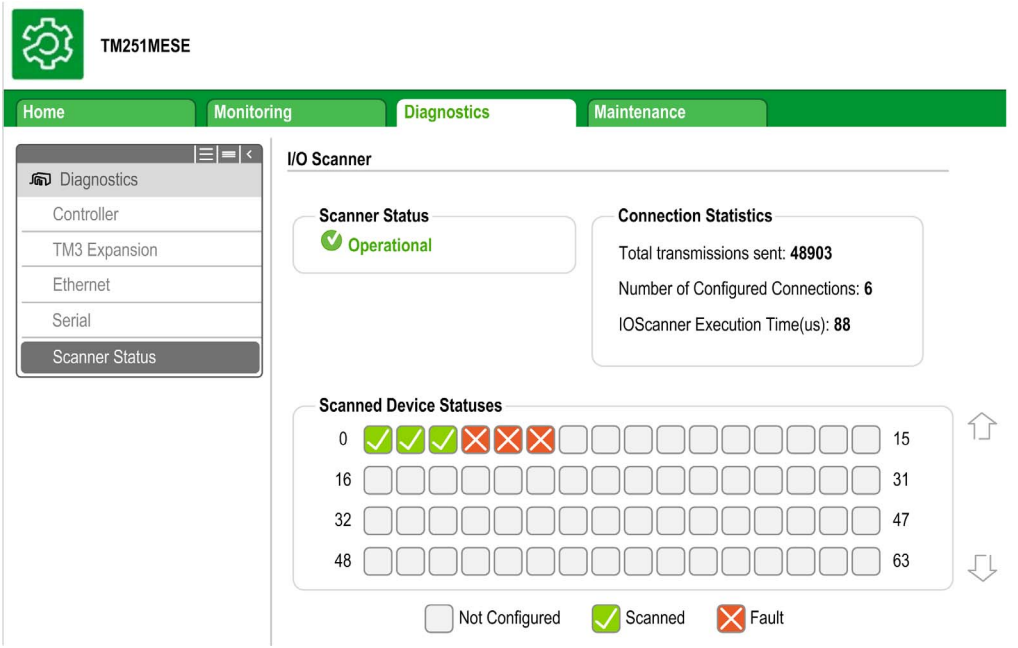
Ethernet Page



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

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

Scanner Status Page



This table presents the different statuses of the channels presented on the **Scanner Status** page:

Icon	Health bit value	Meaning
	1	Request and reply are ongoing on time.
	0	An error is detected, the communications are closed.
	–	This ID does not correspond to a configured channel.
	0	The Modbus TCP IOScanner is stopped, the communications are closed.

NOTE: Clicking an icon open the slave website (if existing). In order to access this website, the computer must reach the slave. For more information, refer to the PC routing *(see page 21)*.

Troubleshooting

Main Issues

Symptom	Possible cause	Resolution
Modbus TCP IOScanner is presented with a red triangle in the **Devices tree**.	The configuration is not compliant with the controller version.	• Clean • Rebuild all • Ensure that the controller has the last firmware version.
A slave is presented with a red triangle in the **Devices tree**.	The controller is unable to communicate with the slave.	• Verify slave wiring and powering. • Verify slave IP address (by using the Remote ping service *(see page 49)* on the IP address of the slave device.). • Verify whether the slave supports the read/write request. • Verify whether the accessed registers are relevant for this slave. • Verify whether the accessed registers are not write-protected. • Verify that the FDR (Fast Device Replacement) service is properly configured inside the slave. • Verify that the **Master IP address** parameter is properly configured inside the slave.
A slave/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 Modbus TCP IOScanner.	Verify the **Modbus TCP IO Scanner Resources** tab.
Some states of the slave are not presented in the application.	The repetition rate is too slow (the value is too high).	Decrease the repetition rate value for the channels associated to this slave.
	The bus cycle task is not fast enough.	• Associate another task to the Modbus TCP IOScanner. • Decrease the cycle value of the associated task.

Appendices

What Is in This Appendix?

The appendix contains the following chapters:

Appendix A
Modbus TCP IOScanner Functions

Overview

This chapter describes the functions included in the `ModbusTCPIOScanner` library

What Is in This Chapter?

This chapter 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 67)*

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
`IOS_GETSTATE`	IosStateCodes *(see page 64)*	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 as the application.

This function call waits for the Modbus TCP IOScanner is 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 67)*.

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
IOS_START	UDINT	0 = successful start Other value = start not successful

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 67)*.

I/O Variable Description

This table describes the input variable:

Input	Type	Comment
channelID	UINT	Channel ID *(see page 15)* 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 at the first application cycle because this function is a synchronous call, and that may take some time.

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 67)*.

I/O Variable Description

This table describes the output variable:

Output	Type	Comment
`IOS_STOP`	UDINT	0 = successful stop Other value = stop not successful

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 Description

This function 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 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

IL and ST Representation

To see the general representation in IL or ST language, refer to Function and Function Block Representation *(see page 67)*.

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.
`Error`	BOOL	Set to TRUE when the configuration ended with an error detected.
`ConfError`	configurationOTBError-Codes *(see page 66)*	Return values: `configurationOTBErrorCodes`
`CommError`	CommunicationError-Codes *(see page 65)*	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
```

Appendix B
Modbus TCP IOScanner Data Types

Overview

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

What Is in This Chapter?

This chapter 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 C
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.

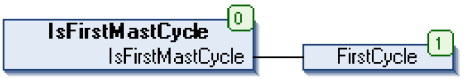
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

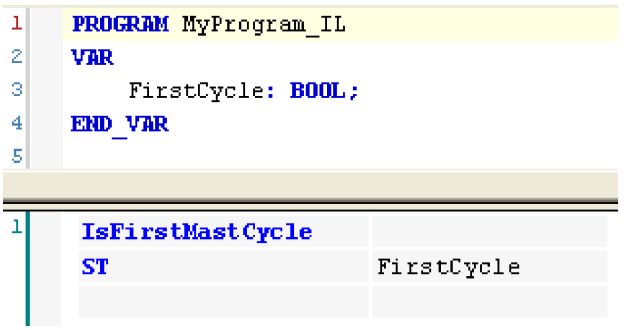
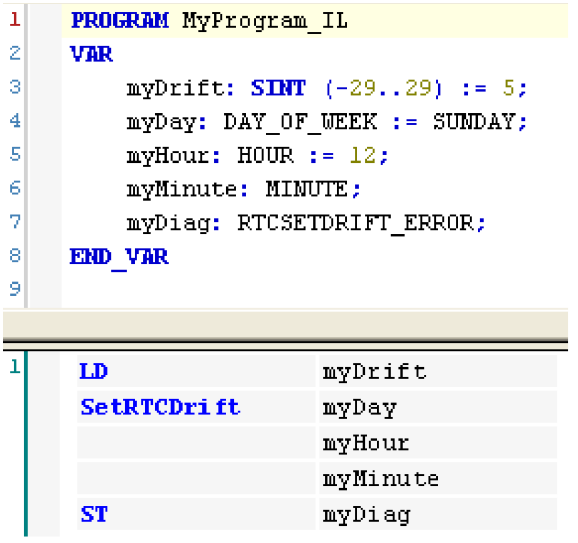
This procedure describes how to insert a function in IL language:

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`	
with input parameters: `SetRTCDrift`	

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

Function	Representation in SoMachine POU IL Editor
IL example of a function without input parameter: `IsFirstMastCycle`	
IL example of a function with input parameters: `SetRTCDrift`	

Using a Function Block in IL Language

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

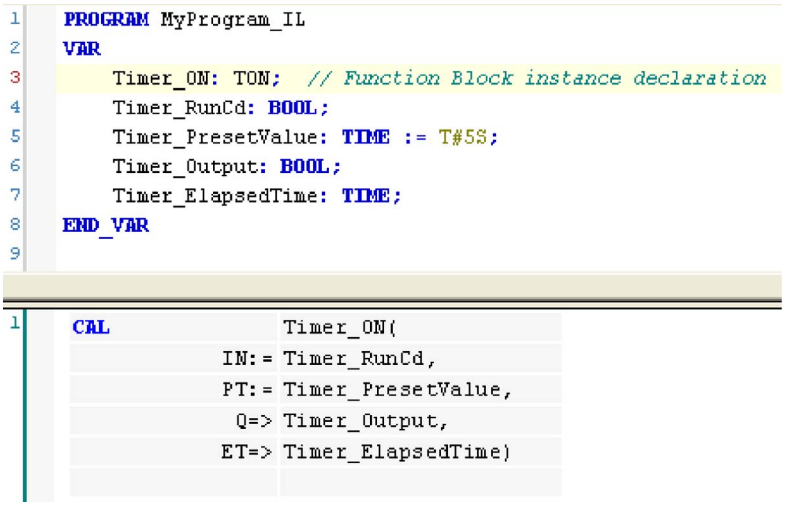
Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function block requires, including the instance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

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`	

The ST language of this function is the following:

Function	Representation in SoMachine 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	

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

Function Block	Representation in SoMachine POU ST Editor

TON	1 PROGRAM MyProgram_ST
	2 VAR
	3 Timer_ON: TON; // Function Block Instance
	4 Timer_RunCd: BOOL;
	5 Timer_PresetValue: TIME := T#5S;
	6 Timer_Output: BOOL;
	7 Timer_ElapsedTime: TIME;
	8 END_VAR
	
	1 Timer_ON(
	2 IN:=Timer_RunCd,
	3 PT:=Timer_PresetValue,
	4 Q=>Timer_Output,
	5 ET=>Timer_ElapsedTime);

Glossary

B

byte

A type that is encoded in an 8-bit format, ranging from `16#00` to `16#FF` in hexadecimal representation.

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.

D

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.)

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*)

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

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).

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

Modbus channel
Communication shuttle that carries a Modbus request between the master and a slave.

P

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.

R

repetition rate
Polling interval of the Modbus request that is sent.

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.

V

variable

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

Index

N

O

P

S

T