Modicon M241 Logic Controller

Programming Guide

12/2015

EIO0000001432.04

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, 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

The purpose of this document is to help you to program and operate your Modicon M241 Logic Controller with the SoMachine software.

NOTE: Read and understand this document and all related documents before installing, operating, or maintaining your Modicon M241 Logic Controller.

The Modicon M241 Logic Controller users should read through the entire document to understand all features.

Validity Note

This document has been updated for the release of SoMachine V4.1 SP2.

Related Documents

Title of Documentation	Reference Number
SoMachine Programming Guide	EIO0000000067 (ENG); EIO0000000069 (FRE); EIO0000000068 (GER); EIO0000000071 (SPA); EIO0000000070 (ITA); EIO0000000072 (CHS)
Modicon M241 Logic Controller Hardware Guide	EIO0000001456 (ENG); EIO0000001457 (FRE); EIO0000001458 (GER); EIO0000001459 (SPA); EIO0000001460 (ITA); EIO0000001461 (CHS)
Modicon TM2 Expansion Modules Configuration Programming Guide	EIO0000000396 (ENG); EIO0000000397 (FRE); EIO0000000398 (GER); EIO0000000399 (SPA); EIO0000000400 (ITA); EIO0000000401 (CHS)

Title of Documentation	Reference Number
Modicon TM3 Expansion Modules Configuration Programming Guide	EIO0000001402 (ENG); EIO0000001403 (FRE); EIO0000001404 (GER); EIO0000001405 (SPA); EIO0000001406 (ITA); EIO0000001407 (CHS)
Modicon TM4 Expansion Modules Programming Guide	EIO0000001802 (ENG); EIO0000001803 (FRE); EIO0000001804 (GER); EIO0000001805 (SPA); EIO0000001806 (ITA); EIO0000001807 (CHS)
Modicon TMC4 Cartridges Programming Guide	EIO0000001790 (ENG); EIO0000001791 (FRE); EIO0000001792 (GER); EIO0000001793 (SPA); EIO0000001794 (ITA); EIO0000001795 (CHS)
Modicon M241 Logic Controller PLCSystem Library Guide	EIO0000001438 (ENG); EIO0000001439 (FRE); EIO0000001440 (GER); EIO0000001441 (SPA); EIO0000001442 (ITA); EIO0000001443 (CHS)
Modicon M241 Logic Controller HSC Library Guide	EIO0000001444 (ENG); EIO0000001445 (FRE); EIO0000001446 (GER); EIO0000001447 (SPA); EIO0000001448 (ITA); EIO0000001449 (CHS)
Modicon M241 Logic Controller PTO/PWM Library Guide	EIO0000001450 (ENG); EIO0000001451 (FRE); EIO0000001452 (GER); EIO0000001453 (SPA); EIO0000001454 (ITA); EIO0000001455 (CHS)

Title of Documentation	Reference Number
SoMachine Controller Assistant User Guide	EIO0000001671 (ENG); EIO0000001672 (FRE); EIO0000001673 (GER); EIO0000001675 (SPA); EIO0000001674 (ITA); EIO0000001676 (CHS)

You can download these technical publications and other technical information from our website at http://download.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.

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
2004/108/EC	Electromagnetic Compatibility Directive
2006/95/EC	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 *EC Machinery Directive* (*EC/2006/42*) 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
About the Modicon M241 Logic Controller

M241 Logic Controller Description

Overview

The M241 Logic Controller has various powerful features and can service a wide range of applications.

Software configuration, programming, and commissioning is accomplished with the SoMachine software described in the SoMachine Programming Guide and the M241 Logic Controller Programming Guide.

Programming Languages

The M241 Logic Controller is configured and programmed with the SoMachine software, which supports the following IEC 61131-3 programming languages:

- IL: Instruction List
- ST: Structured Text
- FBD: Function Block Diagram
- SFC: Sequential Function Chart
- LD: Ladder Diagram

SoMachine software can also be used to program these controllers using CFC (Continuous Function Chart) language.

Power Supply

The power supply of the M241 Logic Controller is 24 Vdc or 100...240 Vac.

Real Time Clock

The M241 Logic Controller includes a Real Time Clock (RTC) system.

Run/Stop

The M241 Logic Controller can be operated externally by the following:

- a hardware Run/Stop switch
- a Run/Stop operation by a dedicated digital input, defined in the software configuration. For more information, refer to Configuration of Digital Inputs *(see page 84)*.
- a SoMachine software command

Memory

This table describes the different types of memory:

Memory Type	Size	Used to
RAM	64 Mbytes, of which 8 Mbytes available for the application	execute the application.
Non-volatile	128 Mbytes	save the program and data in case of a power interruption.

Embedded Inputs/Outputs

The following embedded I/O types are available, depending on the controller reference:

- Regular inputs
- Fast inputs associated with counters
- Regular sink/source transistor outputs
- Fast sink/source transistor outputs associated with pulse generators
- Relay outputs

Removable Storage

The M241 Logic Controllers include an embedded SD card slot.

The main uses of the SD card are:

- Initializing the controller with a new application
- Updating the controller firmware
- Applying post configuration files to the controller
- Applying recipes
- Receiving data logging files

Embedded Communication Features

The following types of communication ports are available depending on the controller reference:

- CANopen Master
- Ethernet
- USB Mini-B
- Serial Line 1
- Serial Line 2

M241 Logic Controller

Reference	Digital Inputs	Digital Outputs	Communication Ports	Terminal Type	Power supply
TM241C24R	6 regular inputs(1) 8 fast inputs (counters)(2)	6 2A relay outputs 4 source fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port	Removable screw terminal blocks	100...240 Vac
TM241CE24R	6 regular inputs(1) 8 fast inputs (counters)(2)	6 2A relay outputs 4 source fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port	Removable screw terminal blocks	100...240 Vac
TM241CEC24R	6 regular inputs(1) 8 fast inputs (counters)(2)	6 2A relay outputs 4 source fast outputs (pulse generators)(3)	2 serial line ports 1 Ethernet port 1 CANopen master port 1 USB programming port	Removable screw terminal blocks	100...240 Vac
TM241C24T	6 regular inputs(1) 8 fast inputs (counters)(2)	Source outputs 6 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port	Removable screw terminal blocks	24 Vdc
TM241CE24T	6 regular inputs(1) 8 fast inputs (counters)(2)	Source outputs 6 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port	Removable screw terminal blocks	24 Vdc
TM241CEC24T	6 regular inputs(1) 8 fast inputs (counters)(2)	Source outputs 6 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port 1 CANopen master port	Removable screw terminal blocks	24 Vdc
TM241C24U	6 regular inputs(1) 8 fast inputs (counters)(2)	Sink outputs 6 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port	Removable screw terminal blocks	24 Vdc

(1) The regular inputs have a maximum frequency of 5 kHz.
(2) The fast inputs can be used either as regular inputs or as fast inputs for counting or event functions.
(3) The fast transistor outputs can be used either as regular transistor outputs, as reflex outputs for counting function (HSC), or as fast transistor outputs for pulse generator functions (FG / PTO / PWM).

Reference	Digital Inputs	Digital Outputs	Communication Ports	Terminal Type	Power supply
TM241CE24U	6 regular inputs(1) 8 fast inputs (counters)(2)	Sink outputs 6 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port	Removable screw terminal blocks	24 Vdc
TM241CEC24U	6 regular inputs(1) 8 fast inputs (counters)(2)	Sink outputs 6 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port 1 CANopen master port	Removable screw terminal blocks	24 Vdc
TM241C40R	16 regular inputs(1) 8 fast inputs (counters)(2)	12 2A relay outputs 4 source fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port	Removable screw terminal blocks	100...240 Vac
TM241CE40R	16 regular inputs(1) 8 fast inputs (counters)(2)	12 2A relay outputs 4 source fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port	Removable screw terminal blocks	100...240 Vac
TM241C40T	16 regular inputs(1) 8 fast inputs (counters)(2)	Source outputs 12 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port	Removable screw terminal blocks	24 Vdc
TM241CE40T	16 regular inputs(1) 8 fast inputs (counters)(2)	Source outputs 12 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port	Removable screw terminal blocks	24 Vdc
TM241C40U	16 regular inputs(1) 8 fast inputs (counters)(2)	Sink outputs 12 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port	Removable screw terminal blocks	24 Vdc

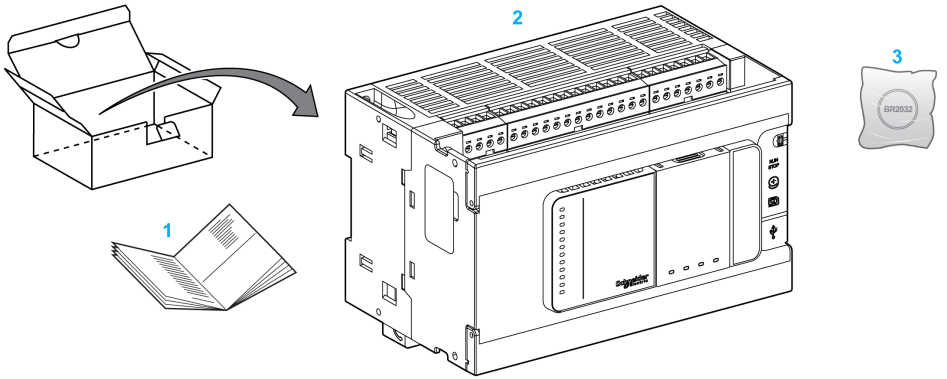
(1) The regular inputs have a maximum frequency of 5 kHz.
(2) The fast inputs can be used either as regular inputs or as fast inputs for counting or event functions.
(3) The fast transistor outputs can be used either as regular transistor outputs, as reflex outputs for counting function (HSC), or as fast transistor outputs for pulse generator functions (FG / PTO / PWM).

Reference	Digital Inputs	Digital Outputs	Communication Ports	Terminal Type	Power supply
TM241CE40U	16 regular inputs(1) 8 fast inputs (counters)(2)	Sink outputs 12 regular transistor outputs 4 fast outputs (pulse generators)(3)	2 serial line ports 1 USB programming port 1 Ethernet port	Removable screw terminal blocks	24 Vdc

(1) The regular inputs have a maximum frequency of 5 kHz.
(2) The fast inputs can be used either as regular inputs or as fast inputs for counting or event functions.
(3) The fast transistor outputs can be used either as regular transistor outputs, as reflex outputs for counting function (HSC), or as fast transistor outputs for pulse generator functions (FG / PTO / PWM).

Delivery Content

The following figure presents the content of the delivery for a M241 Logic Controller:



1 M241 Logic Controller Instruction Sheet
2 M241 Logic Controller
3 Lithium carbon monofluoride battery, type Panasonic BR2032.

Chapter 2
How to Configure the Controller

How to Configure the Controller

Introduction

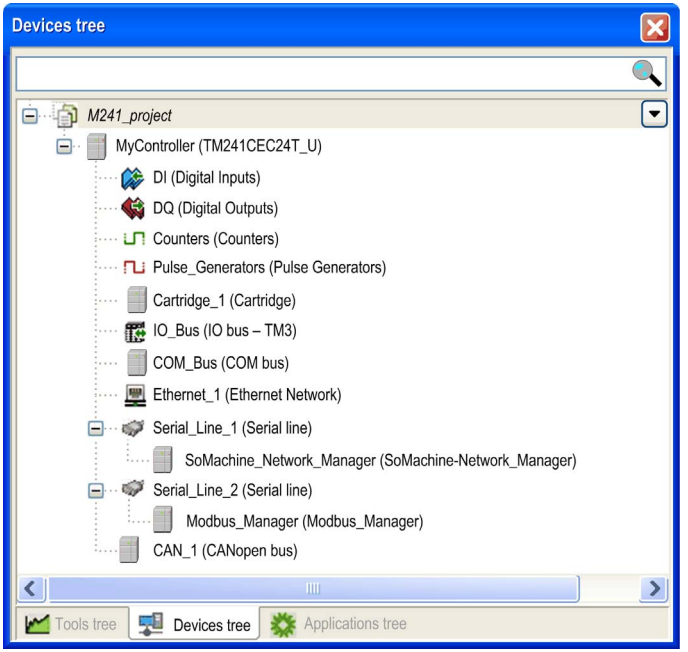
First, create a new project or open an existing project in the SoMachine software.

Refer to the *SoMachine Programming Guide* for information on how to:

- add a controller to your project
- add expansion modules to your controller
- replace an existing controller
- convert a controller to a different but compatible device

Devices Tree

The **Devices tree** presents a structured view of the current hardware configuration. When you add a controller to your project, a number of nodes are added to the **Devices tree**, depending on the functions the controller provides.



Item	Use to Configure...
DI	Embedded digital inputs of the logic controller
DQ	Embedded digital outputs of the logic controller
Counters	Embedded counting functions (HSC)
Pulse_Generators	Embedded pulse generator functions (PTO/PWM/FG)
Cartridge_x	Cartridges plugged into the logic controller
IO_Bus	Expansion modules connected to the logic controller
COM_Bus	Communications bus of the logic controller
Ethernet_x	Embedded Ethernet, serial line, or CANopen communications interfaces **NOTE:** Ethernet and CANopen are only available on some references.
Serial_Line_x	
CAN_x	

Applications Tree

The **Applications tree** allows you to manage project-specific applications as well as global applications, POUs, and tasks.

Tools Tree

The **Tools tree** allows you to configure the HMI part of your project and to manage libraries.

Chapter 3
Libraries

Libraries

Introduction

Libraries provide functions, function blocks, data types and global variables that can be used to develop your project.

The **Library Manager** of SoMachine provides information about the libraries included in your project and allows you to install new ones. For more information on the **Library Manager**, refer to the Functions and Libraries User Guide.

Modicon M241 Logic Controller

When you select a Modicon M241 Logic Controller for your application, SoMachine automatically loads the following libraries:

Library name	Description
IoStandard	**CmploMgr** configuration types, **ConfigAccess**, Parameters and help functions: manages the I/Os in the application.
Standard	Contains functions and function blocks which are required matching IEC61131-3 as standard POUs for an IEC programming system. Link the standard POUs to the project (standard.library).
Util	Analog Monitors, BCD Conversions, Bit/Byte Functions, Controller Datatypes, Function Manipulators, Mathematical Functions, Signals.
PLCCommunication *(see SoMachine, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*	**SysMem, Standard**. These functions facilitate communications between specific devices. Most of them are dedicated to Modbus exchange. Communication functions are processed asynchronously with regard to the application task that called the function.
M241 PLCSystem *(see Modicon M241 Logic Controller, System Functions and Variables, PLCSystem Library Guide)*	Contains functions and variables to get information and send commands to the controller system.
M241 HSC *(see Modicon M241 Logic Controller, High Speed Counting, HSC Library Guide)*	Contains function blocks and variables to get information and send commands to the Fast Inputs/Outputs of the Modicon M241 Logic Controller. These function blocks permit you to implement HSC (High Speed Counting) functions on the Fast Inputs/Outputs of the Modicon M241 Logic Controller.

Library name	Description
M241 PTOPWM *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)*	Contains function blocks and variables to get information and send commands to the Fast Inputs/Outputs of the Modicon M241 Logic Controller. These function blocks permit you to implement PTO (Pulse Train Output) and PWM (Pulse Width Modulation) functions on the Fast Outputs of the Modicon M241 Logic Controller.
Relocation Table *(see page 36)*	Allows you organization of data to optimize exchanges between the Modbus client and the controller, by regrouping non-contiguous data into a contiguous table of registers.

Chapter 4
Supported Standard Data Types

Supported Standard Data Types

Supported Standard Data Types

The controller supports the following IEC data types:

Data Type	Lower Limit	Upper Limit	Information Content
BOOL	FALSE	TRUE	1 Bit
BYTE	0	255	8 Bit
WORD	0	65,535	16 Bit
DWORD	0	4,294,967,295	32 Bit
LWORD	0	2^{64}-1	64 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32,768	32,767	16 Bit
UINT	0	65,535	16 Bit
DINT	-2,147,483,648	2,147,483,647	32 Bit
UDINT	0	4,294,967,295	32 Bit
LINT	-2^{63}	2^{63}-1	64 Bit
ULINT	0	2^{64}-1	64 Bit
REAL	1.175494351e-38	3.402823466e+38	32 Bit
STRING	1 character	255 characters	1 character = 1 byte
WSTRING	1 character	255 characters	1 character = 1 word
TIME	-	-	32 Bit

For more information on `ARRAY`, `LTIME`, `DATE`, `TIME`, `DATE_AND_TIME`, and `TIME_OF_DAY`, refer to the SoMachine Programming Guide.

Chapter 5
Memory Mapping

Introduction

This chapter describes the memory maps and sizes of the different memory areas in the Modicon M241 Logic Controller. These memory areas are used to store user program logic, data and the programming libraries.

What Is in This Chapter?

This chapter contains the following topics:

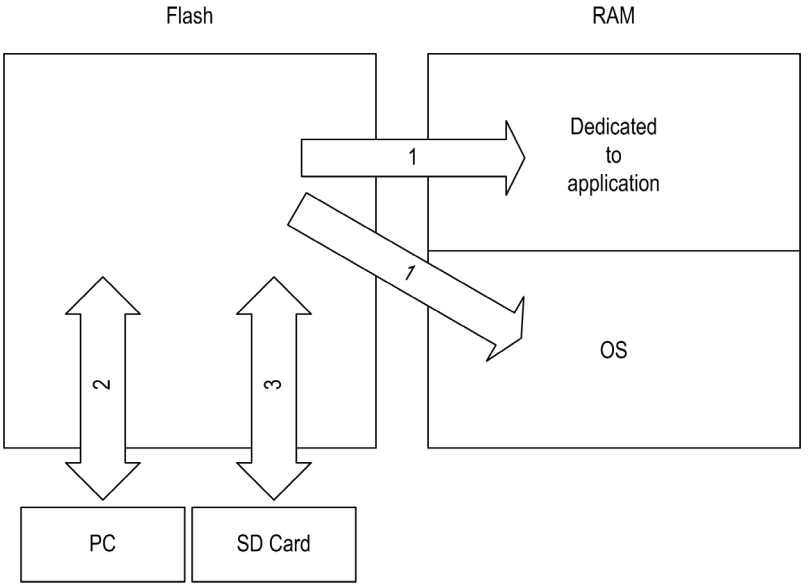
Controller Memory Organization

Introduction

The controller memory is composed of 2 types of physical memory:

- The Flash memory *(see page 32)* contains files (application, configuration files, and so on).
- The RAM (Random Access Memory) *(see page 30)* is used for application execution.

Files Transfers in Memory



Item	Controller State	File Transfer Events	Connection	Description
1	–	Initiated automatically at Power ON and Reboot	Internal	Files transfer from Flash memory to RAM. The content of the RAM is overwritten.
2	All states except INVALID_OS (1)	Initiated by user	Ethernet or USB programming port	Files can be transferred via: ● Web server *(see page 118)* ● FTP server *(see page 132)* ● SoMachine

(1) If the controller is in the INVALID_OS state, the only accessible Flash memory is the SD card and only for firmware upgrades.

Item	Controller State	File Transfer Events	Connection	Description
3	All states	Initiated automatically by script (data transfer) or by power cycle (cloning) when an SD card is connected	SD card	Up/download with SD card
(1) If the controller is in the INVALID_OS state, the only accessible Flash memory is the SD card and only for firmware upgrades.				

NOTE: All files in Flash memory can be read, written, or erased, no matter the controller state. The modification of files in Flash memory does not affect a running application. Any changes to files in Flash memory are taken into account at the next reboot.

RAM Memory Organization

Introduction

This section describes the RAM (Random Access Memory) size for different areas of the Modicon M241 Logic Controller.

Memory Mapping

The RAM size is 64 Mbytes.

The RAM is composed of 2 areas:

- dedicated application memory
- OS memory

This table describes the dedicated application memory:

Area	Element	Size
System area 192 Kbytes	System Area Mappable Addresses %MW0...%MW59999	125 Kbytes
	System and diagnostic variables (%MW60000...%MW60199) This memory is accessible through Modbus requests only. These must be read-only requests.	
	Dynamic Memory Area: Read Relocation Table *(see page 36)* (%MW60200...%MW61999) This memory is accessible through Modbus requests only. These can be read or write requests. However, if this memory is declared in the relocation table, these must be read-only requests.	
	System and diagnostic variables (%MW62000...%MW62199) This memory is accessible through Modbus requests only. These can be read or write requests.	
	Dynamic Memory Area: Write Relocation Table *(see page 36)* (%MW62200...%MW63999) This memory is accessible through Modbus requests only. These can be read or write requests. However, if this memory is declared in the relocation table, it must use write-only requests.	
	Reserved	3 Kbytes
	Retain and Persistent data *(see page 32)*	64 Kbytes
User area 8 Mbytes	Symbols	Dynamic allocation
	Variables	
	Application	
	Libraries	

System and Diagnostic Variables

Variables	Description
PLC_R	Structure of controller read-only system variables.
PLC_W	Structure of controller read/write system variables.
ETH_R	Structure of Ethernet read-only system variables.
ETH_W	Structure of Ethernet read/write system variables.
PROFIBUS_R	Structure of PROFIBUS DP read-only system variables.
SERIAL_R	Structure of Serial Lines read-only system variables.
SERIAL_W	Structure of Serial Lines read/write system variables.
TM3_MODULE_R	Structure of TM3 modules read-only system variables.

For more information on system and diagnostic variables, refer to *M241 PLCSystem Library Guide*.

Memory Addressing

This table describes the memory addressing for the address sizes Double Word (%MD), Word (%MW), Byte (%MB), and Bit (%MX):

Double Words	Words	Bytes	Bits		
%MD0	%MW0	%MB0	%MX0.7	...	%MX0.0
		%MB1	%MX1.7	...	%MX1.0
	%MW1	%MB2	%MX2.7	...	%MX2.0
		%MB3	%MX3.7	...	%MX3.0
%MD1	%MW2	%MB4	%MX4.7	...	%MX4.0
		%MB5	%MX5.7	...	%MX5.0
	%MW3	%MB6	%MX6.7	...	%MX6.0
		%MB7	%MX7.7	...	%MX7.0
%MD2	%MW4	%MB8	%MX8.7	...	%MX8.0
		...	...	...	...
	...	...	...	...	...
		...	...	...	...

Example of overlap of memory ranges:

%MD0 contains %MB0 (...) %MB3, %MW0 contains %MB0 and %MB, %MW1 contains %MB2 and %MB3.

NOTE: The Modbus communication is asynchronous with the application.

Flash Memory Organization

Introduction

The Flash memory contains the file system used by the controller.

File Type

The Modicon M241 Logic Controller manages the following file types:

Type	Description
Boot application	This file resides in Flash memory and contains the compiled binary code of the executable application. Each time the controller is rebooted, the executable application is extracted from the boot application and copied into the controller RAM (1).
Application source	Source file that can be uploaded from Flash memory to the PC if the source file is not available on the PC (2).
Post configuration	File that contains Ethernet, serial line, and firewall parameters. The parameters specified in the file override the parameters in the Executable application at each reboot.
Data logging	Files in which the controller logs events as specified by the user application.
HTML page	HTML pages displayed by the web server for the website embedded in the controller.
Operating System (OS)	Controller firmware that can be written to Flash memory. The firmware file is applied at next reboot of the controller.
Retain variable	Remanent variables
Retain-persistent variable	

(1) The creation of a boot application is optional in SoMachine, according to application properties. Default option is to create the boot application on download. When you download an application from SoMachine to the controller, you are transferring only the binary executable application directly to RAM.

(2) SoMachine does not support uploading of either the executable application or the boot application to a PC for modification. Program modifications must be made to the application source. When you download your application, you have the option to store the source file to Flash memory.

File Organization

This table shows the file organization of the flash memory:

Disk	Directory	File	Content	Up/Downloaded Data Type
/sys	OS	M241FW1v_XX.YY (1)	Firmware of core 1	Firmware
		M241FW2v_XX.YY (1)	Firmware of core 2	
		Version.ini	Control file for firmware version	
	OS/FWM	xxxxx.bin	Firmware of TM4 module	–
	Web	Index.htm	HTML pages served by the web server for the website embedded in the controller.	Website
		Conf.htm		–
		...		–
/usr	App	Application.app	Boot application	Application
		Application.crc		–
		Application.map		–
		Archive.prj (2)	Application source	–
	App/MFW	DeviceID_X.fw (2)	Expansion modules Firmware	Firmware
	Cfg	Machine.cfg (2)	Post configuration file *(see page 197)*	Configuration
		CodesysLateConf.cfg (2)	• Name of application to launch • Routing table (main/sub net)	Configuration

(1): v_XX.YY represents the version
(2): if any

Disk	Directory	File	Content	Up/Downloaded Data Type
/usr	Log	*UserDefinedLogName_1*.log	All *.log files created using the data logging functions *(see SoMachine, Data Logging Functions, DataLogging Library Guide)*. You must specify the total number of files created and the names and contents of each log file.	log file
		...	–	–
		UserDefinedLogName_n.log	–	–
	Rcp		Main directory for Recipe	–
	Syslog	Crash.log (2)	This file contains a record of detected system errors. For use by Schneider Electric Technical Support.	Log file
		System.log (2)	This file contains system event data that is also visible in SoMachine online by viewing the **Log** tab of the **Controller Device Editor** *(see page 76)*.	–
		*.log	This file contains a record of firmware system events. For use by Schneider Electric Technical Support.	–
	/data	–	–	Retained and persistent data
	/sd0	–	–	SD card. Removable
		–	User files	–

(1): v_XX.YY represents the version
(2): if any

NOTE: Use the sysFile, sysDir and CAAFile libraries to access /sd0, and /usr. For more information on the function blocks of these libraries, refer to the CoDeSys Libraries topic in the SoMachine online help.

Backup Data Logging File

Data logging files can become large to the point of exceeding the space available in the file system. Therefore, you should develop a method to archive the log data periodically on an SD card. You could split the log data into several files, for example `LogMonth1, LogMonth2`, and use the **ExecuteScript** command *(see Modicon M241 Logic Controller, System Functions and Variables, PLCSystem Library Guide)* to copy the first file to an SD card. Afterwards, you may remove it from the internal file system while the second file is accumulating data. If you allow the data logging file to grow and exceed the limits of the file size, you could lose data.

NOTICE

LOSS OF DATA

Back up your *.log files to an SD card on a regular schedule that avoids saturating the available free space in your controller file system.

Failure to follow these instructions can result in equipment damage.

Relocation Table

Introduction

The **Relocation Table** allows you to organize data to optimize communication between the controller and other equipment by regrouping non-contiguous data into a contiguous table of located registers, accessible through Modbus.

NOTE: A relocation table is considered as an object. Only one relocation table object can be added to a controller.

Relocation Table Description

This table describes the **Relocation Table** organization:

Register	Description
60200...61999	Dynamic Memory Area: Read Relocation Table
62200...63999	Dynamic Memory Area: Write Relocation Table

For further information, refer to *M241 PLCSystem Library Guide*.

Adding a Relocation Table

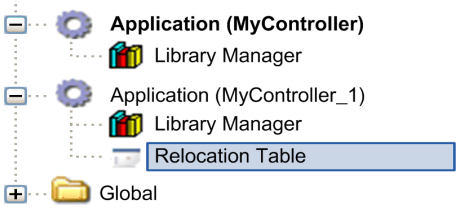
This table describes how to add a **Relocation Table** to your project:

Step	Action
1	Select the **Application** node in the **Applications tree** tab.
2	Click  .
3	Click **Add other objects** → **Relocation Table...**. **Result:** The **Add Relocation Table** window is displayed.
4	Click **Add**. **Result:** The new relocation table is created and initialized. **NOTE:** As a Relocation Table is unique for a controller, its name is **Relocation Table** and cannot be changed.

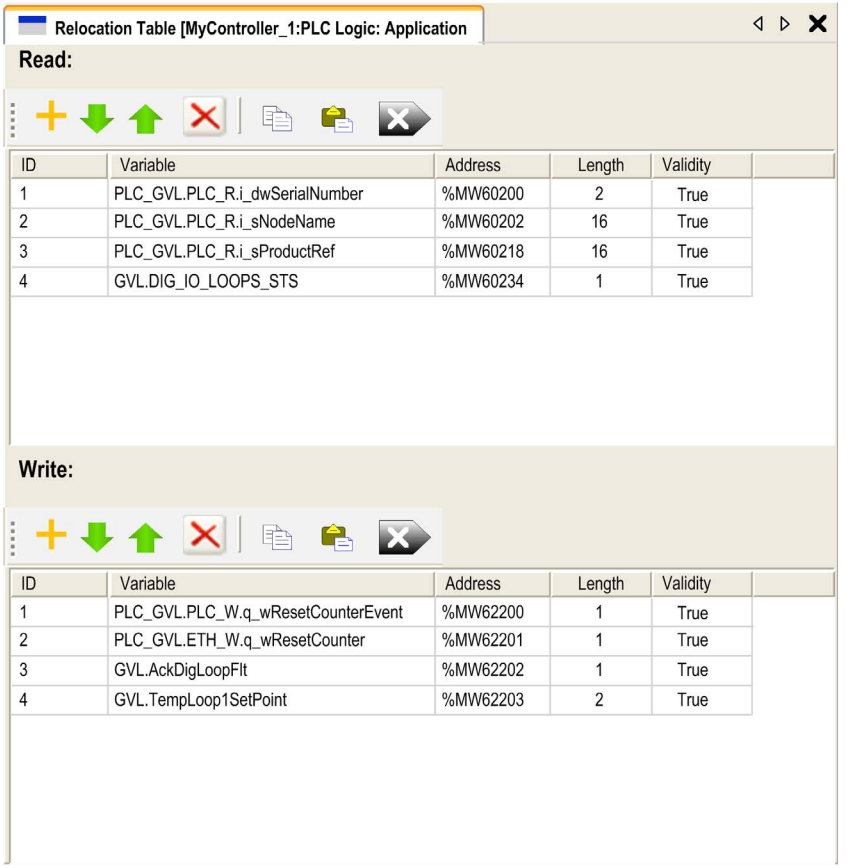
Relocation Table Editor

The relocation table editor allows you to organize your variables in the relocation table.

To access the relocation table editor, double-click the **Relocation Table** node in the **Tools tree** tab:



This picture describes the relocation table editor:



Icon	Element	Description
	New Item	Adds an element to the list of system variables.
	Move Down	Moves down the selected element of the list.
	Move Up	Moves up the selected element of the list.
	Delete Item	Removes the selected elements of the list.
	Copy	Copies the selected elements of the list.
	Paste	Pastes the elements copied.
	Erase Empty Item	Removes all the elements of the list for which the "Variable" column is empty.
-	ID	Automatic incremental integer (not editable).
-	Variable	The name or the full path of a variable (editable).
-	Address	The address of the system area where the variable is stored (not editable).
-	Length	Variable length in word.
-	Validity	Indicates if the entered variable is valid (not editable).

NOTE: If a variable is undefined after program modifications, the content of the cell is displayed in red, the related **Validity** cell is False, and **Address** is set to -1.

Chapter 6
Tasks

Introduction

The **Task Configuration** node in the **Applications tree** allows you to define one or more tasks to control the execution of your application program.

The task types available are:
- Cyclic
- Freewheeling
- Event
- External event

This chapter begins with an explanation of these task types and provides information regarding the maximum number of tasks, the default task configuration, and task prioritization. In addition, this chapter introduces the system and task watchdog functions and explains its relationship to task execution.

What Is in This Chapter?

This chapter contains the following topics:

Maximum Number of Tasks

Maximum Number of Tasks

The maximum number of tasks you can define for the Modicon M241 Logic Controller are:

- Total number of tasks = 19
- Cyclic tasks = 5
- Freewheeling tasks = 1
- Event tasks = 8
- External Event tasks = 8

Special Considerations for Freewheeling

A Freewheeling task *(see page 44)* does not have a fixed duration. In Freewheeling mode, each task scan starts when the previous scan has been completed and after a period of system processing (30% of the total duration of the Freewheeling task). If the system processing period is reduced to less than 15% for more than 3 seconds due to interruptions by other tasks, a system error is detected. For more information, refer to the System Watchdog *(see page 46)*.

NOTE: You may wish to avoid using a Freewheeling task in a multi-task application when some high priority and time-consuming tasks are running. Doing so may provoke a task Watchdog Timeout. You should not assign CANopen to a freewheeling task. CANopen should be assigned to a cyclic task.

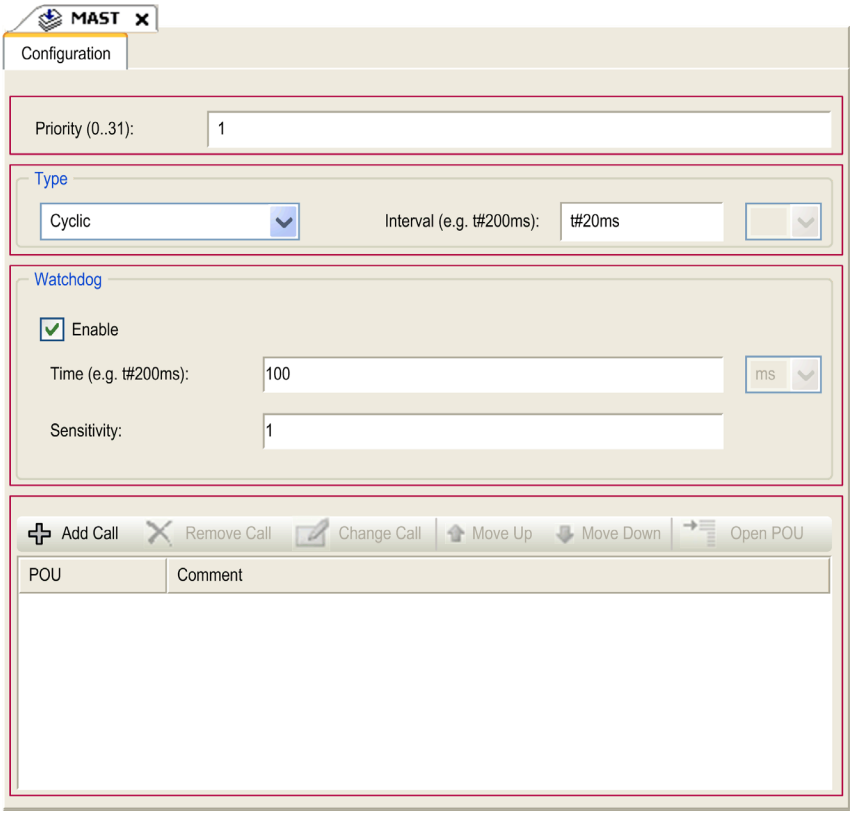
Task Configuration Screen

Screen Description

This screen allows you to configure the tasks. Double-click the task that you want to configure in the **Applications tree** to access this screen.

Each configuration task has its own parameters that are independent of the other tasks.

The **Configuration** window is composed of 4 parts:



The table describes the fields of the **Configuration** screen:

Field Name	Definition
Priority	Configure the priority of each task with a number from 0 to 31 (0 is the highest priority, 31 is the lowest). Only one task at a time can be running. The priority determines when the task will run: ● a higher priority task will pre-empt a lower priority task ● tasks with same priority will run in turn (2 ms time-slice) **NOTE:** Do not assign tasks with the same priority. If there are yet other tasks that attempt to pre-empt tasks with the same priority, the result could be indeterminate and unpredicable. For important safety information, refer to Task Priorities *(see page 47)*.
Type	These task types are available: ● Cyclic *(see page 43)* ● Event *(see page 45)* ● External *(see page 45)* ● Freewheeling *(see page 44)*
Watchdog	To configure the watchdog *(see page 46)*, define these 2 parameters: ● **Time**: enter the timeout before watchdog execution. ● **Sensitivity**: defines the number of expirations of the watchdog timer before the controller stops program execution and enters a HALT state.
POUs	The list of POUs *(see SoMachine, Programming Guide)* (Programming Organization Units) controlled by the task is defined in the task configuration window: ● To add a POU linked to the task, use the command **Add Call** and select the POU in the **Input Assistant** editor. ● To remove a POU from the list, use the command **Remove Call**. ● To replace the currently selected POU of the list by another one, use the command **Change Call**. ● **POU**s are executed in the order shown in the list. To move the **POU**s in the list, select a **POU** and use the command **Move Up** or **Move Down**. **NOTE:** You can create as many POUs as you want. An application with several small POUs, as opposed to one large POU, can improve the refresh time of the variables in online mode.

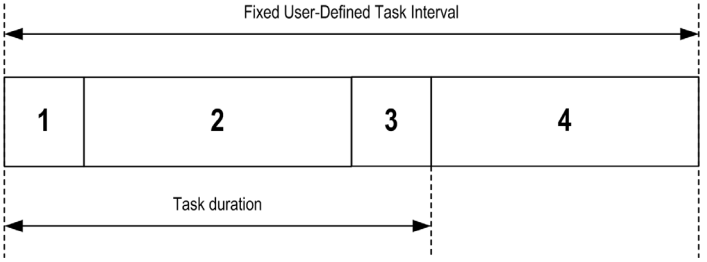
Task Types

Introduction

The following section describes the various task types available for your program, along with a description of the task type characteristics.

Cyclic Task

A Cyclic task is assigned a fixed cycle time using the Interval setting in the Type section of Configuration subtab for that task. Each Cyclic task type executes as follows:



1. **Read Inputs:** The physical input states are written to the %I input memory variables and other system operations are executed.
2. **Task Processing:** The user code (POU, and so on) defined in the task is processed. The %Q output memory variables are updated according to your application program instructions but not yet written to the physical outputs during this operation.
3. **Write Outputs:** The %Q output memory variables are modified with any output forcing that has been defined; however, the writing of the physical outputs depends upon the type of output and instructions used.
   For more information on defining the bus cycle task, refer to the SoMachine Programming Guide and Modicon M241 Logic Controller Settings *(see page 79)*.
   For more information on I/O behavior, refer to Controller States Detailed Description *(see page 57)*.
4. **Remaining Interval time:** The controller firmware carries out system processing and any other lower priority tasks.

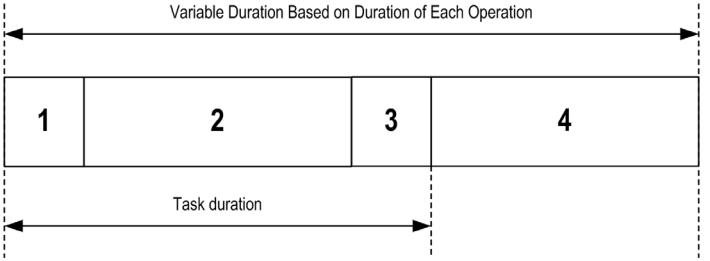
NOTE: If you define too short a period for a cyclic task, it will repeat immediately after the write of the outputs and without executing other lower priority tasks or any system processing. This will affect the execution of all tasks and cause the controller to exceed the system watchdog limits, generating a system watchdog exception.

NOTE: When the task cycle time is set to a value less than 3 ms, the actual task duration should first be monitored through the Task Monitoring screen during commissioning to ensure that it is consistently lower than the configured task cycle time. If greater, the task cycle may not be respected without causing a task cycle watchdog time-out and the controller transitioning to a HALT state. To avoid this condition to a certain degree, when the task cycle time is set to a value of less than 3 ms, real limits of +1 ms are imposed if, on any given cycle, the calculated cycle time slightly exceeds the configured cycle time.

NOTE: Get and set the interval of a Cyclic Task by application using the **GetCurrentTaskCycle** and **SetCurrentTaskCycle** function. (Refer to Toolbox Advance Library Guide for further details.)

Freewheeling Task

A Freewheeling task does not have a fixed duration. In Freewheeling mode, each task scan begins when the previous scan has been completed and after a short period of system processing. Each Freewheeling task type executes as follows:



1. **Read Inputs:** The physical input states are written to the %I input memory variables and other system operations are executed.
2. **Task Processing:** The user code (POU, and so on) defined in the task is processed. The %Q output memory variables are updated according to your application program instructions but not yet written to the physical outputs during this operation.
3. **Write Outputs:** The %Q output memory variables are modified with any output forcing that has been defined; however, the writing of the physical outputs depends upon the type of output and instructions used.
   For more information on defining the bus cycle task, refer to the SoMachine Programming Guide and Modicon M241 Logic Controller Settings *(see page 79)*.
   For more information on I/O behavior, refer to Controller States Detailed Description *(see page 57)*.
4. **System Processing:** The controller firmware carries out system processing and any other lower priority tasks (for example: HTTP management, Ethernet management, parameters management).

Event Task

This type of task is event-driven and is initiated by a program variable. It starts at the rising edge of the boolean variable associated to the trigger event unless pre-empted by a higher priority task. In that case, the Event task will start as dictated by the task priority assignments.

For example, if you have defined a variable called `my_Var` and would like to assign it to an Event, proceed as follows:

Step	Action
1	Double-click the **TASK** in the **Applications tree**.
2	Select **Event** from the **Type** list in the **Configuration** tab.
3	Click the **Input Assistant** button  to the right of the **Event** field. **Result**: The **Input Assistant** window appears.
4	Navigate in the tree of the **Input Assistant** dialog box to find and assign the `my_Var` variable.

NOTE: When the event task is triggered at a too high frequency, the controller will go to the HALT state (Exception). The maximum acceptable rate of events is 5 events per millisecond. In this situation, the message 'ISR Count Exceeded' is logged in the application log page.

External Event Task

This type of task is event-driven and is initiated by the detection of a hardware or hardware-related function event. It starts when the event occurs unless pre-empted by a higher priority task. In that case, the External Event task will start as dictated by the task priority assignments.

For example, an External event task could be associated with an HSC Stop event. To associate the **BLOCK0_HSCSTOP** event to an External event task, select it from the **External** event drop-down list on the **Configuration** tab.

Depending on the controller, there are up to 4 types of events that can be associated with an External event task:

- Rising edge on an advanced input (`DI8...DI11`)
- HSC thresholds
- HSC Stop
- CAN Sync

NOTE: CAN Sync is a specific event object, depending on the **CANopen manager** configuration.

System and Task Watchdogs

Introduction

Two types of watchdog functionality are implemented for the Modicon M241 Logic Controller:

- **System Watchdogs**: These watchdogs are defined in and managed by the controller firmware. These are not configurable by the user.
- **Task Watchdogs**: These watchdogs are optional watchdogs that you can define for each task. These are managed by your application program and are configurable in SoMachine.

System Watchdogs

Three system watchdogs are defined for the Modicon M241 Logic Controller. They are managed by the controller firmware and are therefore sometimes referred to as hardware watchdogs in the SoMachine online help. When one of the system watchdogs exceeds its threshold conditions, an error is detected.

The threshold conditions for the 3 system watchdogs are defined as follows:

- If all of the tasks require more than 85% of the processor resources for more than 3 seconds, a system error is detected. The controller enters the HALT state.
- If the total execution time of the tasks with priorities between 0 and 24 reaches 100% of processor resources for more than 1 second, an application error is detected. The controller responds with an automatic reboot into the EMPTY state.
- If the lowest priority task of the system is not executed during an interval of 10 seconds, a system error is detected. The controller responds with an automatic reboot into the EMPTY state.

NOTE: System watchdogs are not configurable by the user.

Task Watchdogs

SoMachine allows you to configure an optional task watchdog for every task defined in your application program. (Task watchdogs are sometimes also referred to as software watchdogs or control timers in the SoMachine online help). When one of your defined task watchdogs reaches its threshold condition, an application error is detected and the controller enters the HALT state.

When defining a task watchdog, the following options are available:

- **Time**: This defines the allowable maximum execution time for a task. When a task takes longer than this, the controller will report a task watchdog exception.
- **Sensitivity**: The sensitivity field defines the number of task watchdog exceptions that must occur before the controller detects an application error.

To access the configuration of a task watchdog, double-click the **Task** in the **Applications tree**.

NOTE: For more information on watchdogs, refer to SoMachine Programming Guide.

Task Priorities

Task Priority Configuration

You can configure the priority of each task between 0 and 31 (0 is the highest priority, 31 is the lowest). Each task must have a unique priority. If you assign the same priority to more than one task, execution for those tasks is indeterminate and unpredictable, which may lead to unintended consequences.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Do not assign the same priority to different tasks.

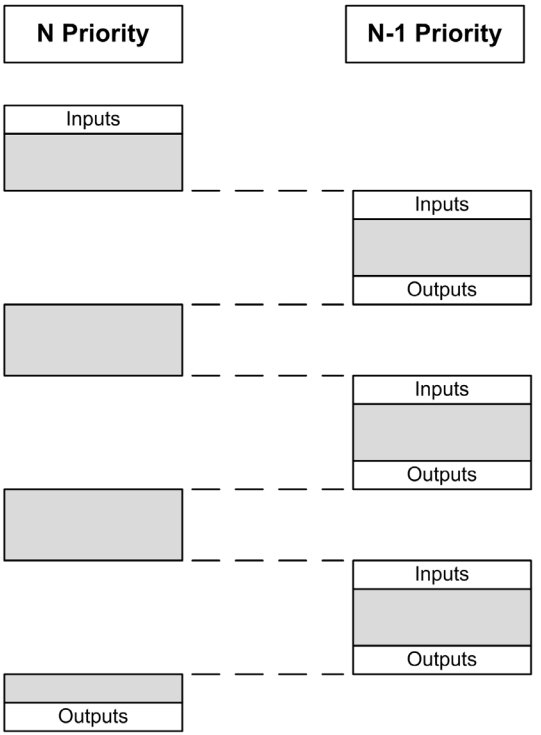
Failure to follow these instructions can result in death, serious injury, or equipment damage.

Task Priority Suggestions

- Priority 0 to 24: Controller tasks. Assign these priorities to tasks with a high availability requirement.
- Priority 25 to 31: Background tasks. Assign these priorities to tasks with a low availability requirement.

Task Priorities of Embedded I/Os

When a task cycle starts, it can interrupt any task with lower priority (task preemption). The interrupted task will resume when the higher priority task cycle is finished.



NOTE: If the same input is used in different tasks the input image may change during the task cycle of the lower priority task.
To improve the likelihood of proper output behavior during multitasking, a warning message is displayed if outputs in the same byte are used in different tasks.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Map your inputs so that tasks do not alter the input images in an unexpected manner.

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

Task Priorities of TM2/TM3 Modules and CANopen I/Os

You can select the task that drives TM3 and CANopen physical exchanges. In the **PLC settings**, select **Bus cycle task** to define the task for the exchange. By default, the task is set to **MAST**. This definition at the controller level can be overridden by the I/O bus configuration *(see page 101)*. During the read and write phases, all physical I/Os are refreshed at the same time. TM3/TM2 and CANopen data is copied into a virtual I/O image during a physical exchanges phase, as shown in this figure:

Inputs are read from the I/O image table at the beginning of the task cycle. Outputs are written to the I/O image table at the end of the task.

NOTE: Event tasks cannot drive the TM3/TM2 bus cycle.

Default Task Configuration

Default Task Configuration

Default Task Configuration

The MAST task can be configured in Freewheeling or Cyclic mode. The MAST task is automatically created by default in Cyclic mode. Its preset priority is medium (15), its preset interval is 20 ms, and its task watchdog service is activated with a time of 100 ms and a sensitivity of 1. Refer to Task Priorities *(see page 47)* for more information on priority settings. Refer to Task Watchdogs *(see page 46)* for more information on watchdogs.

Designing an efficient application program is important in systems approaching the maximum number of tasks. In such an application, it can be difficult to keep the resource utilization below the system watchdog threshold. If priority reassignments alone are not sufficient to remain below the threshold, some lower priority tasks can be made to use fewer system resources if the SysTaskWaitSleep function is added to those tasks. For more information about this function, see the optional SysTask library of the system / SysLibs category of libraries.

NOTE: Do not delete or change the name of the MAST task. Otherwise, SoMachine detects an error when you attempt to build the application, and you will not be able to download it to the controller.

Chapter 7
Controller States and Behaviors

Introduction

This chapter provides you with information on controller states, state transitions, and behaviors in response to system events. It begins with a detailed controller state diagram and a description of each state. It then defines the relationship of output states to controller states before explaining the commands and events that result in state transitions. It concludes with information about Remanent variables and the effect of SoMachine task programming options on the behavior of your system.

What Is in This Chapter?

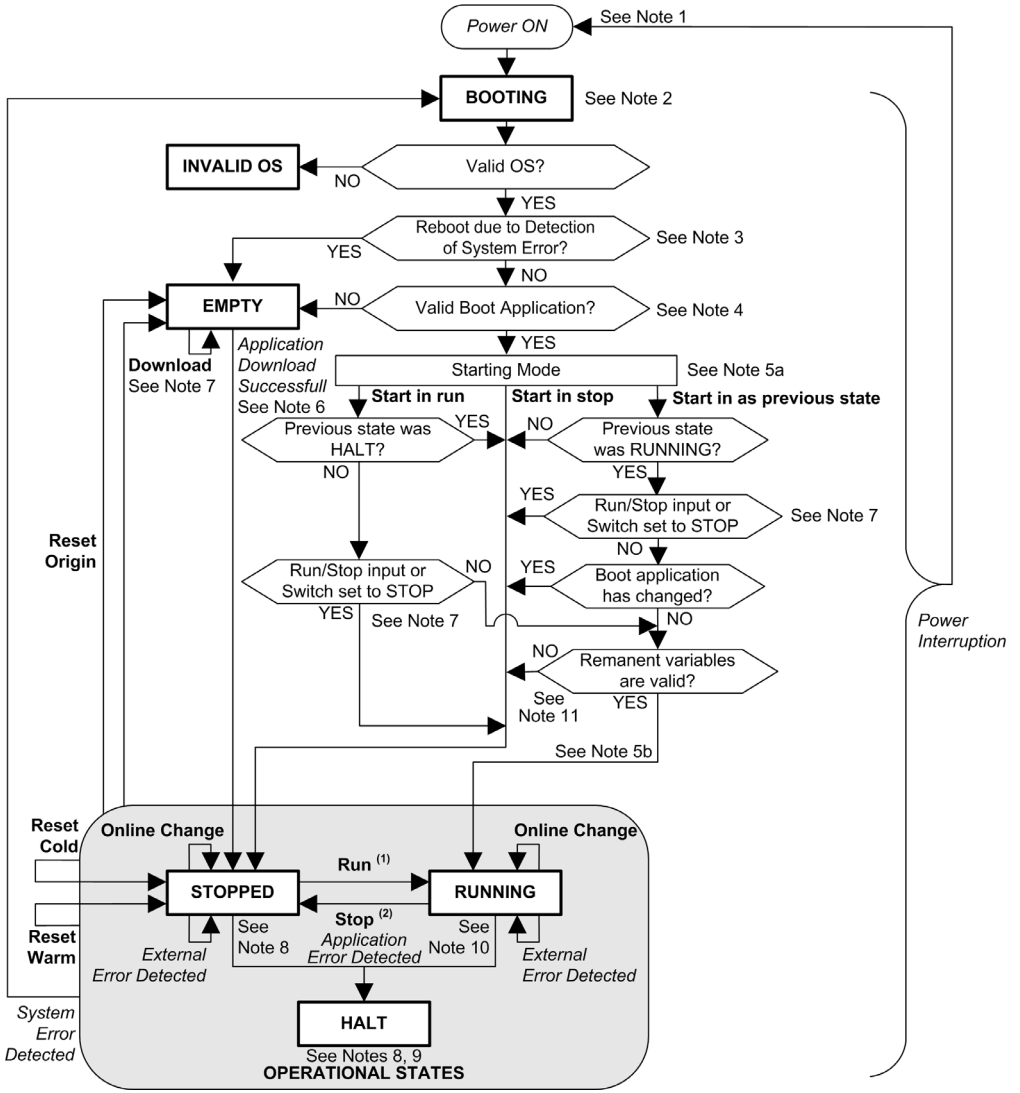
This chapter contains the following sections:

Section 7.1
Controller State Diagram

Controller State Diagram

Controller State Diagram

This diagram describes the controller operating mode:



Legend:
- Controller states are indicated in **ALL-CAPS BOLD**
- User and application commands are indicated in **Bold**
- System events are indicated in *Italics*
- Decisions, decision results, and general information are indicated in normal text

(1) For details on STOPPED to RUNNING state transition, refer to Run Command *(see page 65)*.

(2) For details on RUNNING to STOPPED state transition, refer to Stop Command *(see page 65)*.

Note 1

The Power Cycle (Power Interruption followed by a Power ON) deletes all output forcing settings. Refer to Controller State and Output Behavior *(see page 62)* for further details.

Note 2

The outputs will assume their initialization states.

Note 3

In some cases, when a system error is detected, it will cause the controller to reboot automatically into the EMPTY state as if no Boot application were present in the Flash memory. However, the Boot application is not deleted from the Flash memory. In this case, the ERR LED (Red) flashes regularly.

Note 4

After verification of a valid Boot application the following events occur:
- The application is loaded into RAM.
- The Post Configuration *(see page 197)* file settings (if any) are applied.

During the load of the boot application, a Check context test occurs to assure that the Remanent variables are valid. If the Check context test is invalid, the boot application will load but the controller will assume STOPPED state *(see page 68)*.

Note 5a

The **Starting Mode** is set in the **PLC settings** tab of the **Controller Device Editor** *(see page 79)*.

Note 5b

When a power interruption occurs, the controller continues in the RUNNING state for at least 4 ms before shutting down. If you have configured and provide power to the Run/Stop input from the same source as the controller, the loss of power to this input will be detected immediately, and the controller will behave as if a STOP command was received. Therefore, if you provide power to the controller and the Run/Stop input from the same source, your controller will normally reboot into the STOPPED state after a power interruption when **Starting Mode** is set to **Start as previous state**.

Note 6

During a successful application download the following events occur:

- The application is loaded directly into RAM.
- By default, the Boot application is created and saved into the Flash memory.

Note 7

The default behavior after downloading an application program is for the controller to enter the STOPPED state irrespective of the Run/Stop input setting, the Run/Stop switch position or the last controller state before the download.

However, there are 2 considerations in this regard:

Online Change: An online change (partial download) initiated while the controller is in the RUNNING state returns the controller to the RUNNING state if successful and provided the Run/Stop input is configured and set to Run or Run/Stop switch is set to Run. Before using the **Login with online change** option, test the changes to your application program in a virtual or non-production environment and confirm that the controller and attached equipment assume their expected conditions in the RUNNING state.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Always verify that online changes to a RUNNING application program operate as expected before downloading them to controllers.

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

NOTE: Online changes to your program are not automatically written to the Boot application, and will be overwritten by the existing Boot application at the next reboot. If you wish your changes to persist through a reboot, manually update the Boot application by selecting **Create boot application** in the online menu (the controller must be in the STOPPED state to achieve this operation).

Multiple Download: SoMachine has a feature that allows you to perform a full application download to multiple targets on your network or fieldbus. One of the default options when you select the **Multiple Download...** command is the **Start all applications after download or online change** option, which restarts all download targets in the RUNNING state, provided their respective Run/Stop inputs are commanding the RUNNING state, but irrespective of their last controller state before the multiple download was initiated. Deselect this option if you do not want all targeted controllers to restart in the RUNNING state. In addition, before using the **Multiple Download** option, test the changes to your application program in a virtual or non-production environment and confirm that the targeted controllers and attached equipment assume their expected conditions in the RUNNING state.

⚠ WARNING
UNINTENDED EQUIPMENT OPERATION Always verify that your application program will operate as expected for all targeted controllers and equipment before issuing the "Multiple Download..." command with the "Start all applications after download or online change" option selected. **Failure to follow these instructions can result in death, serious injury, or equipment damage.**

NOTE: During a multiple download, unlike a normal download, SoMachine does not offer the option to create a Boot application. You can manually create a Boot application at any time by selecting **Create boot application** in the **Online menu** on all targeted controllers (the controller must be in the STOPPED state for this operation).

Note 8

The SoMachine software platform allows many powerful options for managing task execution and output conditions while the controller is in the STOPPED or HALT states. Refer to Controller States Description *(see page 57)* for further details.

Note 9

To exit the HALT state it is necessary to issue one of the Reset commands (Reset Warm, Reset Cold, Reset Origin), download an application or cycle power.

In case of non-recoverable event (hardware watchdog or internal error), a cycle power is mandatory.

Note 10

The RUNNING state has 2 exception conditions:

- RUNNING with External Error: this exception condition is indicated by the I/O LED, which displays solid Red. You may exit this state by clearing the external error (probably changing the application configuration). No controller commands are required.
- RUNNING with Breakpoint: this exception condition is indicated by the RUN LED, which displays a single flash. Refer to Controller States Description *(see page 57)* for further details.

Note 11

The boot application can be different from the application loaded. It can happen when the boot application was downloaded through SD card, FTP, or file transfer or when an online change was performed without creating the boot application.

Section 7.2
Controller States Description

Controller States Description

Introduction

This section provides a detailed description of the controller states.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Never assume that your controller is in a certain controller state before commanding a change of state, configuring your controller options, uploading a program, or modifying the physical configuration of the controller and its connected equipment.
- Before performing any of these operations, consider the effect on all connected equipment.
- Before acting on a controller, always positively confirm the controller state by viewing its LEDs, confirming the condition of the Run/Stop input, verifying the presence of output forcing, and reviewing the controller status information via SoMachine.(1)

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

(1) The controller states can be read in the PLC_R.i_wStatus system variable of the M241 PLCSystem library *(see Modicon M241 Logic Controller, System Functions and Variables, PLCSystem Library Guide)*

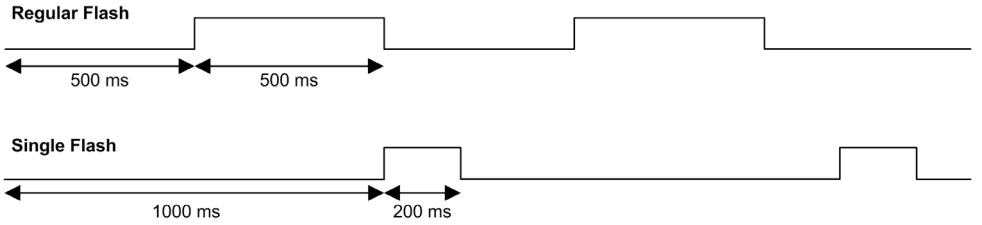
Controller States Table

The following table describes the controller states:

Controller State	Description	LED		
		RUN (Green)	ERR (Red)	I/O (Red)
BOOTING	The controller executes the boot firmware and its own internal self-tests. It then checks the checksum of the firmware and user applications.	ON	OFF	OFF
		OFF	ON	ON
		OFF	ON	OFF
INVALID_OS	There is not a valid firmware file present in the flash memory. The controller does not execute the application. Refer to the Firmware Upgrade section to restore a correct state.	OFF	Regular flash	OFF

Controller State	Description	LED		
		RUN (Green)	ERR (Red)	I/O (Red)
EMPTY	The controller has no application.	OFF	Single flash	OFF
EMPTY after a system error detected	This state is the same as the normal EMPTY state. But the application is present, and is intentionally not loaded. A next reboot (power cycle), or a new application download, will restore correct state.	OFF	Fast flash	OFF
RUNNING	The controller is executing a valid application.	ON	OFF	OFF
RUNNING with breakpoint	This state is same as the RUNNING state with the following exceptions: ● The task-processing portion of the program does not resume until the breakpoint is cleared. ● The LED indications are different. ● For more information on breakpoint management, refer to SoMachine, Programming Guide.	Single flash	OFF	OFF
RUNNING with external error detected	Configuration, TM3, SD card, or other I/O error detected. When I/O LED is ON, the details about the detected error can be found in `PLC_R.i_lwSystemFault_1` and `PLC_R.i_lwSystemFault_2`. Any of the detected error conditions reported by these variables cause the I/O LED to be ON.	ON	OFF	ON
STOPPED	The controller has a valid application that is stopped. See details of the STOPPED state *(see page 59)* for an explanation of the behavior of outputs and field buses in this state.	Regular flash	OFF	OFF
STOPPED with external error detected	Configuration, TM3, SD card, or other I/O error detected.	Regular flash	OFF	ON
HALT	The controller stops executing the application because it has detected an application error	Regular flash	ON	–
Boot Application not saved	The controller has an application in memory that differs from the application in Flash memory. At next power cycle, the application will be changed by the one from Flash memory.	ON or regular flash	Single flash	OFF

This figure shows the difference between the regular flash and single flash:



Details of the STOPPED State

The following statements are true for the STOPPED state:

- The input configured as the Run/Stop input remains operational.
- The output configured as the Alarm output remains operational and goes to a value of 0.
- Ethernet, Serial (Modbus, ASCII, and so on), and USB communication services remain operational and commands written by these services can continue to affect the application, the controller state, and the memory variables.
- All outputs initially assume their configured default state (**Keep current values** or **Set all outputs to default**) or the state dictated by output forcing if used. For output used by a PTO function, the default value is ignored, in order not to generate an extra pulse. The subsequent state of the outputs depends on the value of the **Update IO while in stop** setting and on commands received from remote devices.

Task and I/O Behavior When Update IO While In Stop Is Selected

When the **Update IO while in stop** setting is selected:

- The Read Inputs operation continues normally. The physical inputs are read and then written to the `%I` input memory variables.
- The Task Processing operation is not executed.
- The Write Outputs operation continues. The `%Q` output memory variables are updated to reflect either the **Keep current values** configuration or the **Set all outputs to default** configuration, adjusted for any output forcing, and then written to the physical outputs.
  NOTE: Expert functions cease operating. For example, a counter will be stopped.

  - If **Keep current values** configuration is selected:

  The PTO, PWM, frequency Generator outputs and also the HSC reflex outputs are set to 0.

  - If **Set all outputs to default** configuration is selected:

  The PTO outputs are set to 0.

  The PWM, frequency generator outputs and also the HSC reflex outputs are set to the configured default values.

CAN Behavior When Update IO While In Stop Is Selected

The following is true for the CAN buses when the **Update IO while in stop** setting is selected:

- The CAN bus remains fully operational. Devices on the CAN bus continue to perceive the presence of a functional CAN Master.
- TPDO and RPDO continue to be exchanged.
- The optional SDO, if configured, continue to be exchanged.
- The Heartbeat and Node Guarding functions, if configured, continue to operate.
- If the **Behaviour for outputs in Stop** field is set to **Keep current values**, the TPDOs continue to be issued with the last actual values.
- If the **Behaviour for outputs in Stop** field is **Set all outputs to default** the last actual values are updated to the default values and subsequent TPDOs are issued with these default values.

Task and I/O Behavior When Update IO While In Stop Is Not Selected

When the **Update IO while in stop** setting is not selected, the controller sets the I/O to either the **Keep current values** or **Set all outputs to default** condition (as adjusted for output forcing if used). After this, the following becomes true:

- The Read Inputs operation ceases. The `%I` input memory variablea are frozen at their last values.
- The Task Processing operation is not executed.
- The Write Outputs operation ceases. The `%Q` output memory variables can be updated via the Ethernet, Serial, and USB connections. However, the physical outputs are unaffected and retain the state specified by the configuration options.
  NOTE: Expert functions cease operating. For example, a counter will be stopped.

  - If **Keep current values** configuration is selected:

  The PTO, PWM, frequency generator outputs and also the HSC reflex outputs are set to 0.

  - If **Set all outputs to default** configuration is selected:

  The PTO outputs are set to 0.

  The PWM, frequency generator outputs and also the HSC reflex outputs are set to the configured default values.

CAN Behavior When Update IO While In Stop Is Not Selected

The following is true for the CAN buses when the **Update IO while in stop** setting is not selected:

- The CAN Master ceases communications. Devices on the CAN bus assume their configured fallback states.
- TPDO and RPDO exchanges cease.
- Optional SDO, if configured, exchanges cease.
- The Heartbeat and Node Guarding functions, if configured, stop.
- The current or default values, as appropriate, are written to the TPDOs and sent once before stopping the CAN Master.

Section 7.3
State Transitions and System Events

Overview

This section begins with an explanation of the output states possible for the controller. It then presents the system commands used to transition between controller states and the system events that can also affect these states. It concludes with an explanation of the Remanent variables, and the circumstances under which different variables and data types are retained through state transitions.

What Is in This Section?

This section contains the following topics:

Controller States and Output Behavior

Introduction

The Modicon M241 Logic Controller defines output behavior in response to commands and system events in a way that allows for greater flexibility. An understanding of this behavior is necessary before discussing the commands and events that affect controller states. For example, typical controllers define only 2 options for output behavior in stop: fallback to default value or keep current value.

The possible output behaviors and the controller states to which they apply are:

- managed by **Application Program**
- keep **Current Values**
- set All **Outputs to Default**
- hardware **Initialization Values**
- software **Initialization Values**
- **Output Forcing**

Managed by Application Program

Your application program manages outputs normally. This applies in the RUNNING and RUNNING with External Error detected states.

Keep Current Values

Select this option by choosing **Keep current values** in the **Behavior for outputs in Stop** drop-down menu of the **PLC settings** subtab of the **Controller Editor**. To access the Controller Editor, right-click on the controller in the device tree and select **Edit Object**.

This output behavior applies in the STOPPED controller state. It also applies to CAN bus in the HALT controller state. Outputs are set to and maintained in their current state, although the details of the output behavior vary greatly depending on the setting of the **Update I/O while in stop** option and the actions commanded via configured fieldbusses.Refer to Controller States Description *(see page 57)* for more details on these variations.

Set All Outputs to Default

Select this option by choosing **Set all outputs to default** in the **Behavior for outputs in Stop** drop-down menu of the **PLC settings** subtab of the **Controller Editor**. To access the **Controller Editor**, right-click on the controller in the device tree and select **Edit Object**.

This output behavior applies when the application is going from RUN state to STOPPED state or if the application is going from RUN state to HALT state. It also applies to CAN bus in the HALT controller state. Outputs are set to and maintained in their current state, although the details of the output behavior vary greatly depending on the setting of the **Update I/O while in stop** option and the actions commanded via configured fieldbusses.Refer to Controller States Description *(see page 57)* for more details on these variations.

The outputs driven by a PTO expert function will not apply the default value.

Hardware Initialization Values

This output state applies in the BOOTING, EMPTY (following power cycle with no boot application or after the detection of a system error), and INVALID_OS states.

In the initialization state, analog, transistor, and relay outputs assume the following values:

- For an analog output: Z (high impedance)
- For a fast transistor output: Z (high impedance)
- For a regular transistor output: 0 Vdc
- For a relay output: Open

Software Initialization Values

This output state applies when downloading or when resetting the application. It applies at the end of the download or at the end of a reset warm or cold.

The software **Initialization Values** are the initialization values of outputs images (%I, %Q, or variables mapped on %I or %Q).

By default, they are set to 0 but it is possible to map the I/O in a GVL and assign to the outputs a value different from 0.

Output Forcing

The controller allows you to force the state of selected outputs to a defined value for the purposes of system testing, commissioning, and maintenance.

You are only able to force the value of an output while your controller is connected to SoMachine.

To do so, use the **Force values** command in the **Debug** menu.

Output forcing overrides all other commands to an output irrespective of the task programming that is being executed.

When you logout of SoMachine when output forcing has been defined, you are presented with the option to retain output forcing settings. If you select this option, the output forcing continues to control the state of the selected outputs until you download an application or use one of the Reset commands.

When the option **Update I/O while in stop**, if supported by your controller, is checked (default state), the forced outputs keep the forcing value even when the logic controller is in STOP.

Output Forcing Considerations

The output you wish to force must be contained in a task that is currently being executed by the controller. Forcing outputs in unexecuted tasks, or in tasks whose execution is delayed either by priorities or events will have no effect on the output. However, once the task that had been delayed is executed, the forcing will take effect at that time.

Depending on task execution, the forcing could impact your application in ways that may not be obvious to you. For example, an event task could turn on an output. Later, you may attempt to turn off that output but the event is not being triggered at the time. This would have the effect of the forcing being apparently ignored. Further, at a later time, the event could trigger the task at which point the forcing would take effect.

The outputs driven by a PTO, PWM, and HSC expert function cannot be forced.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- You must have a thorough understanding of how forcing will affect the outputs relative to the tasks being executed.
- Do not attempt to force I/O that is contained in tasks that you are not certain will be executed in a timely manner, unless your intent is for the forcing to take affect at the next execution of the task whenever that may be.
- If you force an output and there is no apparent affect on the physical output, do not exit SoMachine without removing the forcing.

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

Commanding State Transitions

Run Command

Effect: Commands a transition to the RUNNING controller state.

Starting Conditions: BOOTING or STOPPED state.

Methods for Issuing a Run Command:

- Run/Stop Input: If configured, command a rising edge to the Run/Stop input (assuming the Run/Stop switch is in the RUN position). Set the Run/Stop to 1 for all of the subsequent options to be effective.
  Refer to Run/Stop Input for more information.
- SoMachine Online Menu: Select the **Start** command.
- RUN command from Web Server
- By an external call via Modbus request using the PLC_W.q_wPLCControl and PLC_W.q_uiOpenPLCControl system variables of the M241 PLCSystem library.
- **Login with online change** option: An online change (partial download) initiated while the controller is in the RUNNING state returns the controller to the RUNNING state if successful.
- **Multiple Download** Command: sets the controllers into the RUNNING state if the **Start all applications after download or online change** option is selected, irrespective of whether the targeted controllers were initially in the RUNNING, STOPPED, HALT, or EMPTY state.
- The controller is restarted into the RUNNING state automatically under certain conditions.

Refer to Controller State Diagram for further details.

Stop Command

Effect: Commands a transition to the STOPPED controller state.

Starting Conditions: BOOTING, EMPTY, or RUNNING state.

Methods for Issuing a Stop Command:

- Run/Stop Input: If configured, command a value of 0 to the Run/Stop input. Refer to Run/Stop Input for more information.
- SoMachine Online Menu: Select the **Stop** command.
- STOP command from WebServer
- By an internal call by the application or an external call via Modbus request using the PLC_W. q_wPLCControl and PLC_W. q_uiOpenPLCControl system variables of the M241 PLCSystem library.
- **Login with online change** option: An online change (partial download) initiated while the controller is in the STOPPED state returns the controller to the STOPPED state if successful.
- **Download** Command: implicitly sets the controller into the STOPPED state.
- **Multiple Download** Command: sets the controllers into the STOPPED state if the **Start all applications after download or online change** option is not selected, irrespective of whether the targeted controllers were initially in the RUNNING, STOPPED, HALT, or EMPTY state.

- REBOOT by Script: The file transfer script on an SD card can issue a REBOOT as its final command. The controller will be rebooted into the STOPPED state provided the other conditions of the boot sequence allow this to occur. Refer to Reboot *(see page 68)* for further details.
- The controller is restarted into the STOPPED state automatically under certain conditions.

Refer to Controller State Diagram for further details.

Reset Warm

Effect: Resets all variables, except for the remanent variables, to their default values. Places the controller into the STOPPED state.

Starting Conditions: RUNNING, STOPPED, or HALT states.

Methods for Issuing a Reset Warm Command:

- SoMachine Online Menu: Select the **Reset warm** command.
- By an internal call by the application or an external call via Modbus request using the PLC_W. q_wPLCControl and PLC_W. q_uiOpenPLCControl system variables of the M241 PLCSystem library.

Effects of the Reset Warm Command:

1. The application stops.
2. Forcing is erased.
3. Diagnostic indications for errors are reset.
4. The values of the retain variables are maintained.
5. The values of the retain-persistent variables are maintained.
6. All non-located and non-remanent variables are reset to their initialization values.
7. The values of the first 1000 `%MW` registers are maintained.
8. The values of `%MW1000` to `%MW59999` registers are reset to 0.
9. All fieldbus communications are stopped and then restarted after the reset is complete.
10. All I/O are briefly reset to their initialization values and then to their user-configured default values.
11. The Post Configuration *(see page 197)* file is read.

For details on variables, refer to Remanent Variables *(see page 72)*.

Reset Cold

Effect: Resets all variables, except for the retain-persistent type of remanent variables, to their initialization values. Places the controller into the STOPPED state.

Starting Conditions: RUNNING, STOPPED, or HALT states.

Methods for Issuing a Reset Cold Command:

- SoMachine Online Menu: Select the **Reset cold** command.
- By an internal call by the application or an external call via Modbus request using the PLC_W. q_wPLCControl and PLC_W. q_uiOpenPLCControl system variables of the M241 PLCSystem library.

Effects of the Reset Cold Command:

1. The application stops.
2. Forcing is erased.
3. Diagnostic indications for errors are reset.
4. The values of the retain variables are reset to their initialization value.
5. The values of the retain-persistent variables are maintained.
6. All non-located and non-remanent variables are reset to their initialization values.
7. The values of the first 1000 `%MW` registers are maintained.
8. The values of `%MW1000` to `%MW59999` registers are reset to 0.
9. All fieldbus communications are stopped and then restarted after the reset is complete.
10. All I/O are briefly reset to their initialization values and then to their user-configured default values.
11. The Post Configuration file is read *(see page 197)*.

For details on variables, refer to Remanent Variables *(see page 72)*.

Reset Origin

Effect: Resets all variables, including the remanent variables, to their initialization values. Erases all user files on the controller. Places the controller into the EMPTY state.

Starting Conditions: RUNNING, STOPPED, or HALT states.

Methods for Issuing a Reset Origin Command:

- SoMachine Online Menu: Select the **Reset origin** command.

Effects of the Reset Origin Command:

1. The application stops.
2. Forcing is erased.
3. All user files (Boot application, data logging, Post Configuration) are erased.
4. Diagnostic indications for errors are reset.
5. The values of the retain variables are reset.
6. The values of the retain-persistent variables are reset.
7. All non-located and non-remanent variables are reset.
8. The values of the first 1000 `%MW` registers are reset to 0.
9. The values of `%MW1000` to `%MW59999` registers are reset to 0.
10. All fieldbus communications are stopped.

11. Embedded Expert I/O are reset to their previous user-configured default values.
12. All other I/O are reset to their initialization values.

For details on variables, refer to Remanent Variables *(see page 72)*.

Reboot

Effect: Commands a reboot of the controller.

Starting Conditions: Any state.

Methods for Issuing the Reboot Command:

- Power cycle
- REBOOT by Script

Effects of the Reboot:

1. The state of the controller depends on a number of conditions:

a. The controller state will be RUNNING if:
The Reboot was provoked by a power cycle or a reboot by script and:
- the **Starting Mode** is set to **Start in run**, and if the Run/Stop input is not configured, and if the controller was not in HALT state before the power cycle, and if the remanent variables are valid.
- the **Starting Mode** is set to **Start in run**, and if the Run/Stop input is configured and set to RUN, and if the controller was not in HALT state before the power cycle, and if the remanent variables are valid.
- the **Starting Mode** is set to **Start in as previous state**, and Controller state was RUNNING before the power cycle, and if the Run/Stop input is set to not configured and the boot application has not changed and the remanent variables are valid.
- the **Starting Mode** is set to **Start in as previous state**, and Controller state was RUNNING before the power cycle, and if the Run/Stop input is configured and is set to RUN.

b. The controller state will be STOPPED if:
The Reboot was provoked by a Power cycle or a reboot by script and:
- the **Starting Mode** is set to **Start in stop**.
- the **Starting Mode** is set to **Start in as previous state** and the controller state was not RUNNING before the power cycle.
- the **Starting Mode** is set to **Start in as previous state** and the controller state was RUNNING before the power cycle, and if the Run/Stop input is set to not configured, and if the boot application has changed.
- the **Starting Mode** is set to **Start in as previous state** and the controller state was RUNNING before the power cycle, and if the Run/Stop input is set to not configured, and if the boot application has not changed, and if the remanent variables are not valid.
- the **Starting Mode** is set to **Start in as previous state** and the controller state was RUNNING before the power cycle, and if the Run/Stop input is configured and is set to STOP.
- the **Starting Mode** is set to **Start in run** and if the controller state was HALT before the power cycle.
- the **Starting Mode** is set to **Start in run**, and if the controller state was not HALT before the power cycle, and if the Run/Stop input is configured and is set to STOP.

c. The controller state will be EMPTY if:
- There is no boot application or the boot application is invalid, or
- The reboot was provoked by specific System Errors.

d. The controller state will be INVALID_OS if there is no valid firmware.

2. Forcing is maintained if the boot application is loaded successfully. If not, forcing is erased.
3. Diagnostic indications for errors are reset.
4. The values of the retain variables are restored if saved context is valid.
5. The values of the retain-persistent variables are restored if saved context is valid.
6. All non-located and non-remanent variables are reset to their initialization values.
7. The values of the first 1000 `%MW` registers are restored if saved context is valid.
8. The values of `%MW1000` to `%MW59999` registers are reset to 0.
9. All fieldbus communications are stopped and restarted after the boot application is loaded successfully.
10. All I/O are reset to their initialization values and then to their user-configured default values if the controller assumes a STOPPED state after the reboot.
11. The Post Configuration file is read *(see page 197)*.

For details on variables, refer to Remanent Variables *(see page 72)*.

NOTE: The Check context test concludes that the context is valid when the application and the remanent variables are the same as defined in the Boot application.

NOTE: If you provide power to the Run/Stop input from the same source as the controller, the loss of power to this input will be detected immediately, and the controller will behave as if a STOP command was received. Therefore, if you provide power to the controller and the Run/Stop input from the same source, your controller will normally reboot into the STOPPED state after a power interruption when **Starting Mode** is set to **Start in as previous state**.

NOTE: If you make an online change to your application program while your controller is in the RUNNING or STOPPED state but do not manually update your Boot application, the controller will detect a difference in context at the next reboot, the remanent variables will be reset as per a Reset cold command, and the controller will enter the STOPPED state.

Download Application

Effect: Loads your application executable into the RAM memory. Optionally, creates a Boot application in the Flash memory.

Starting Conditions: RUNNING, STOPPED, HALT, and EMPTY states.

Methods for Issuing the Download Application Command:

- SoMachine:
  2 options exist for downloading a full application:
  - Download command.
  - Multiple Download command.

  For important information on the application download commands, refer to Controller State Diagram.
- FTP: Load Boot application file to the Flash memory using FTP. The updated file is applied at the next reboot.
- SD card: Load Boot application file using an SD card in the controller SD card slot. The updated file is applied at the next reboot. Refer to File Transfer with SD Card for further details.

Effects of the SoMachine Download Command:

1. The existing application stops and then is erased.
2. If valid, the new application is loaded and the controller assumes a STOPPED state.
3. Forcing is erased.
4. Diagnostic indications for errors are reset.
5. The values of the retain variables are reset to their initialization values.
6. The values of any existing retain-persistent variables are maintained.
7. All non-located and non-remanent variables are reset to their initialization values.
8. The values of the first 1000 `%MW` registers are maintained.
9. The values of `%MW1000` to `%MW59999` registers are reset to 0.
10. All fieldbus communications are stopped and then any configured fieldbus of the new application is started after the download is complete.
11. Embedded Expert I/O are reset to their previous user-configured default values and then set to the new user-configured default values after the download is complete.
12. All other I/O are reset to their initialization values and then set to the new user-configured default values after the download is complete.
13. The Post Configuration file is read *(see page 197)*.

For details on variables, refer to Remanent Variables *(see page 72)*.

Effects of the FTP or SD Card Download Command:

There are no effects until the next reboot. At the next reboot, the effects are the same as a reboot with an invalid context. Refer to Reboot *(see page 68)*.

Error Detection, Types, and Management

Error Management

The controller detects and manages three types of errors:

- external errors
- application errors
- system errors

This table describes the types of errors that may be detected:

Type of Error Detected	Description	Resulting Controller State
External Error	External errors are detected by the system while RUNNING or STOPPED but do not affect the ongoing controller state. An external error is detected in the following cases: ● A connected device reports an error to the controller. ● The controller detects an error with an external device, for example, when the external device is communicating but not properly configured for use with the controller. ● The controller detects an error with the state of an output. ● The controller detects a communication interruption with a device. ● The controller is configured for an expansion module that is not present or not detected. ● The boot application in Flash memory is not the same as the one in RAM.	RUNNING with External Error Detected Or STOPPED with External Error Detected
Application Error	An application error is detected when improper programming is encountered or when a task watchdog threshold is exceeded.	HALT
System Error	A system error is detected when the controller enters a condition that cannot be managed during runtime. Most such conditions result from firmware or hardware exceptions, but there are some cases when incorrect programming can result in the detection of a system error, for example, when attempting to write to memory that was reserved during runtime, or when a system watchdog time-out occurs. **NOTE:** There are some system errors that can be managed by runtime and are therefore treated like application errors.	BOOTING → EMPTY

NOTE: Refer to the M241 PLCSystem library Guide for more detailed information on diagnostics.

Remanent Variables

Overview

Remanent variables retain their values in the event of power outages, reboots, resets, and application program downloads. There are multiple types of remanent variables, declared individually as "retain" or "persistent", or in combination as "retain-persistent".

NOTE: For this controller, variables declared as persistent have the same behavior as variables declared as retain-persistent.

This table describes the behavior of remanent variables in each case:

Action	VAR	VAR RETAIN	VAR GLOBAL PERSISTENT RETAIN
Online change to application program	X	X	X
Stop	X	X	X
Power cycle	-	X	X
Reset warm	-	X	X
Reset cold	-	-	X
Reset origin	-	-	-
Download of application program	-	-	X
X The value is maintained **-** The value is reinitialized			

NOTE: The first 1000 `%MW` are automatically retained and persistent if no variable is associated to them. Their values are kept after a reboot / Reset warm / Reset cold. The other `%MW` are managed as VAR.

For example, if you have in your program:

```
VAR myVariable AT %MW0 : WORD; END_VAR
```

`%MW0` will behave like `myVariable` (not retained and not persistent).

Adding Retain Persistent Variables

Declare retain persistent (**VAR GLOBAL PERSISTENT RETAIN**) symbols in the **PersistentVars** window:

Step	Action
1	Select the **Application** node in the **Applications tree**.
2	Click  .
3	Choose **Add other objects** → **Persistent variables**
4	Click **Add**. **Result:** The **PersistentVars** window is displayed.

Chapter 8
Controller Device Editor

Introduction

This chapter describes how to configure the controller.

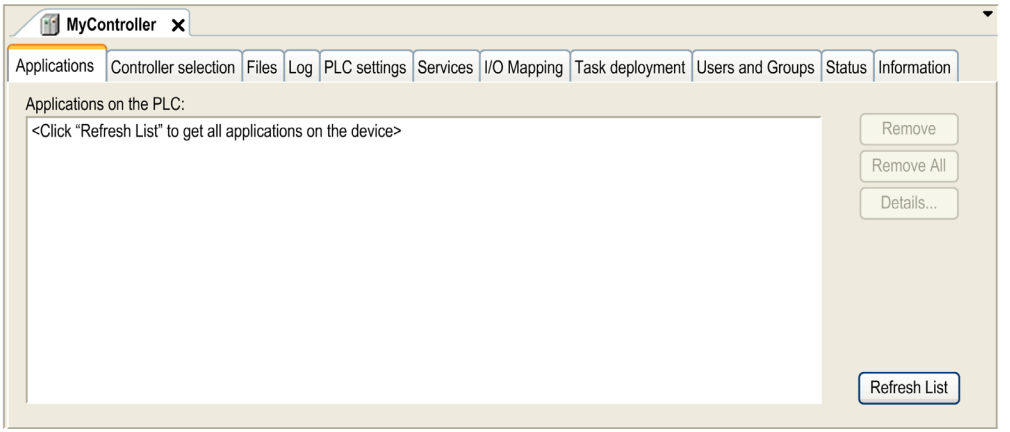
What Is in This Chapter?

This chapter contains the following topics:

Controller Parameters

Controller Parameters

To open the device editor, double-click **MyController** in the **Devices tree**:



Tabs Description

Tab	Description	Restriction
Applications	Presents the application running on the controller and allows removing the application from the controller.	Online mode only
Controller selection *(see page 78)*	Manages the connection from PC to the controller: ● helping you find a controller in a network, ● presenting the list of available controllers, so you can connect to the selected controller and manage the application in the controller, ● helping you physically identify the controller from the device editor, ● helping you change the communication settings of the controller. The controller list is detected through the NetManage or through the Active Path based on the communication settings. To access the **Communication settings**, click **Project** → **Project Settings...** in the menu bar. For more information, refer to the SoMachine Programming Guide (*Communication Settings*).	Online mode only
Files *(see page 32)*	File management between the PC and the controller. Only one logic controller disk at a time can be seen through this tab. When an SD card is inserted, this file displays the content of the SD card. Otherwise, this tab displays the content of the */usr* directory of the internal flash memory of the controller.	Online mode only

Tab	Description	Restriction
Log	View the controller log file.	Online mode only
PLC settings *(see page 79)*	Configuration of: ● application name ● I/O behavior in stop ● bus cycle options	–
Services *(see page 81)*	Lets you configure the online services of the controller (RTC, device identification).	Online mode only
I/O Mapping	Mapping of the input and output channels of an I/O device on project (application) variables.	–
Task deployment	Displays a list of I/Os and their assignments to tasks.	After compilation only
Users and Groups	The **Users and Groups** tab is provided for devices supporting online user management. It allows setting up users and access-rights groups and assigning them access rights to control the access on SoMachine projects and devices in online mode. For more details, refer to the SoMachine Programming Guide.	–
Status	No information delivered.	–
Information	Displays general information about the device (name, description, provider, version, image).	–

Controller Selection

Introduction

This tab allows you to manage the connection from the PC to the controller:

- Helping you find a controller in a network.
- Presenting the list of controllers, so you can connect to the selected controller and manage the application inside the controller.
- Helping you physically identify the controller from the device editor.
- Helping you change the communication settings of the controller.

Process Communication Settings

The **Process communication settings** window lets you change the Ethernet communication settings. To do so, click **Controller selection** tab. The list of controllers available in the network appears. Select and right-click the required row and click **Process communication settings ...** in the context menu.

You can configure the Ethernet settings in the **Process communication settings** window in 2 ways:

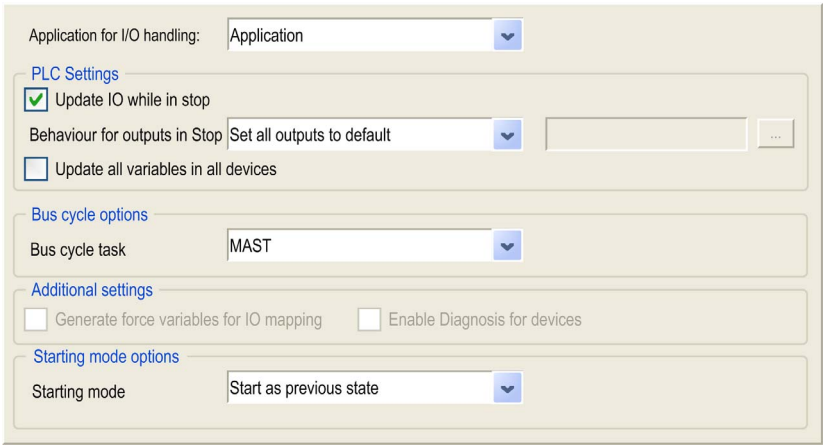
- Without the **Save settings permanently** option:
  Configure the communication parameters and click **OK**. These settings are immediately taken into account and are not kept if the controller is reset. For the next resets, the communication parameters configured into the application are taken into account.
- With the **Save settings permanently** option:
  You can also verify the **Save settings permanently** option before you click **OK**. Once this option is checked, the Ethernet parameters configured here are always taken into account on reset instead of the Ethernet parameters configured into the SoMachine application.

For more information on the **Controller selection** view of the device editor, refer to the SoMachine Programming Guide.

PLC Settings

Overview

The figure below presents the **PLC Settings** tab:



Element		Description
Application for I/O handling		By default, set to Application because there is only one application in the controller.
PLC settings	**Update IO while in stop**	If this option is activated (default), the values of the input and output channels get also updated when the controller is stopped.
	Behavior for outputs in Stop	From the selection list, choose one of the following options to configure how the values at the output channels should be handled in case of controller stop: ● **Keep current values** ● **Set all outputs to default**
	Update all variables in all devices	If this option is activated, then for all devices of the current controller configuration all I/O variables will get updated in each cycle of the bus cycle task. This corresponds to the option **Always update variables**, which can be set separately for each device in the **I/O Mapping** dialog.
Bus cycle options	**Bus cycle task**	This configuration setting is the parent for all **Bus cycle task** parameters used in the application device tree. Some devices with cyclic calls, such as a **CANopen manager**, can be attached to a specific task. In the device, when this setting is set to **Use parent bus cycle setting**, the setting set for the controller is used. The selection list offers all tasks currently defined in the active application. The default setting is the MAST task. **NOTE: <unspecified>** means that the task is in "slowest cyclic task" mode.

Element		Description
Additional settings	**Generate force variables for IO mapping**	Not used.
	Enable Diagnosis for devices	Not used.
Starting mode Options	**Starting mode**	This option defines the starting mode on a power-on. For further information, refer to State behavior diagram *(see page 53)*. Select with this option one of these starting modes: ● Start as previous state ● Start in stop ● Start in run

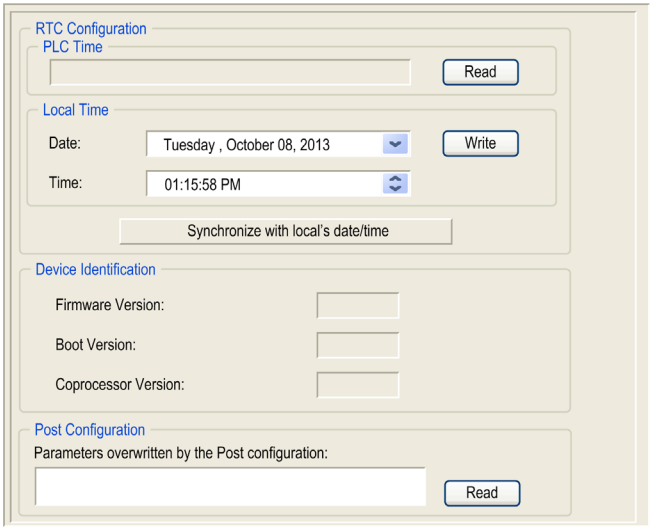
Services

Services Tab

The **Services** tab is divided in 3 parts:

- RTC Configuration
- Device Identification
- Post Configuration

The figure below shows the **Services** tab:



NOTE: To have controller information, you must be connected to the controller.

Element		Description
RTC Configuration	**PLC time**	Displays the date/time read from the controller. This read-only field is initially empty. To read and display the date/time saved on the controller, click the **Read** button.
	Local time	Lets you define a date and a time that are sent to the controller by clicking the **Write** button. A message box informs you on the success of the command. Local time fields are initialized with the current PC settings.
	Synchronize with local date/time	Lets you directly send the current PC settings. A message box informs you on the success of the command.
Device Identification		Displays the Firmware version, the Boot Version, and the Coprocessor Version of the selected controller, if connected.
Post Configuration		Displays the application parameters overwritten by the Post configuration *(see page 197)*.

Chapter 9
Embedded Inputs and Outputs Configuration

Embedded I/Os Configuration

Overview

The embedded I/O function allows configuration of the controller inputs and outputs.

The M241 logic controller provides:

I/O Type	24 I/O References	40 I/O References
	TM241•24•	TM241•40•
Fast inputs	8	8
Regular inputs	6	16
Fast outputs	4	4
Regular outputs	6	12

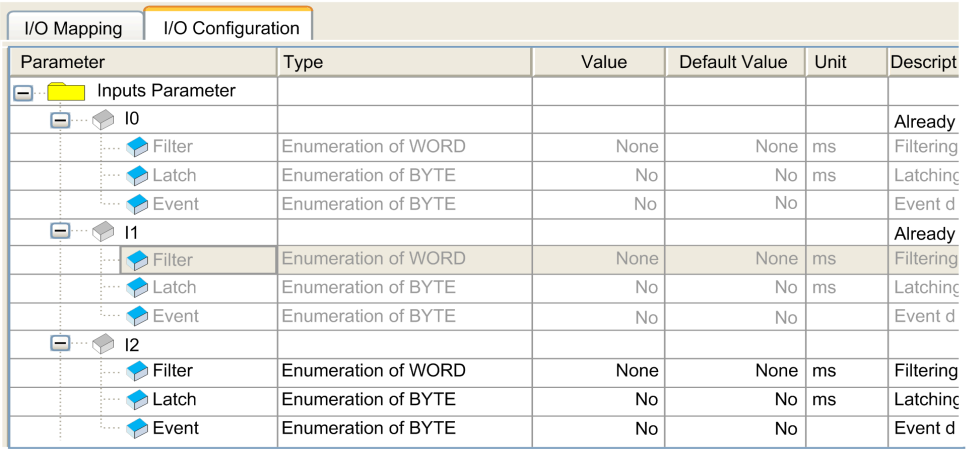
Accessing the I/O Configuration Window

Follow these steps to access the I/O configuration window:

Step	Description
1	Double-click **DI** (digital inputs) or **DQ** (digital outputs) in the **Devices tree**. Refer to Devices tree *(see page 21)*.
2	Select the **I/O Configuration** tab.

Configuration of Digital Inputs

This figure shows the **I/O Configuration** tab for digital inputs:



NOTE: For more information on the **I/O Mapping** tab, refer to the SoMachine Programming Guide.

Digital Input Configuration Parameters

For each digital input, you can configure the following parameters:

Parameter	Value	Description	Constraint
Filter	None 1 ms 4 ms (default) 12 ms	Reduces the effect of noise on a controller input.	Available if **Latch** and **Event** are disabled. In the other cases, this parameter is disabled and its value is **None**.
Latch	No* Yes	Allows incoming pulses with amplitude widths shorter than the controller scan time to be captured and recorded.	This parameter is only available for the fast inputs I0 to I7. Available if: **Event** disabled AND **Filter** disabled. Use latch inputs in MAST task only.
Event	No* Rising edge Falling edge Both edges	Event detection	This parameter is only available for the fast inputs I0 to I7. Available if: **Event** disabled AND **Filter** disabled.

Parameter	Value	Description	Constraint
Bounce	0.000 ms 0.001 ms 0.002 ms* 0.005 ms 0.010 ms 0.05 ms 0.1 ms 0.5 ms 1 ms 5 ms	Reduces the effect of bounce on a controller input.	Available if **Latch** is enabled or **Event** is enabled. In the other cases, this parameter is disabled and its value is 0.002.
Run/Stop Input	None I0...I7	The Run/Stop input can be used to run or stop a program in the controller	Select one of the inputs to use as the Run/Stop Input.
* parameter default value			

NOTE: The selection is grey and inactive if the parameter is unavailable.

Run/Stop Input

This table presents the different states:

Input states	Result
State 0	Stops the controller and ignores external Run commands.
A rising edge	From the STOPPED state, initiate a start-up of an application in RUNNING state, if no conflict with Run/Stop switch position.
State 1	The application can be controlled by: ● SoMachine (Run/Stop) ● a hardware Run/Stop switch ● application (Controller command) ● network command (Run/Stop command) Run/Stop command is available through the Web Server command.

NOTE: Run/Stop input is managed even if the option **Update I/O while in stop** is not selected in Controller Device Editor (**PLC settings** tab) *(see page 79)*.
Inputs assigned to configured expert functions cannot be configured as Run/Stop inputs.

For further details about controller states and states transitions, refer to Controller State Diagram *(see page 52)*.

⚠ WARNING

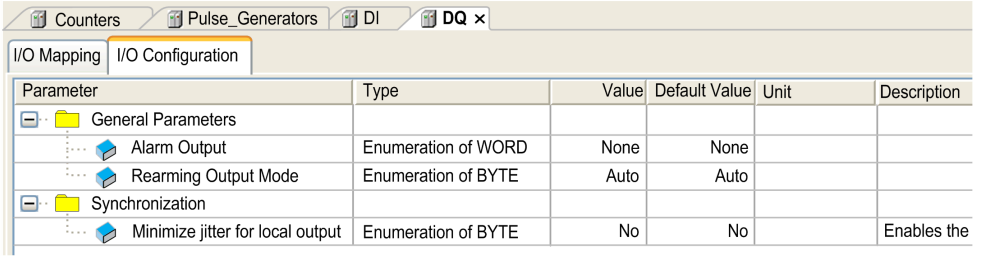
UNINTENDED MACHINE OR PROCESS START-UP

- Verify the state of security of your machine or process environment before applying power to the Run/Stop input.
- Use the Run/Stop input to help prevent the unintentional start-up from a remote location.

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

Configuration of Digital Outputs

This figure shows the **I/O Configuration** tab for digital outputs:



NOTE: For more information on the **I/O Mapping** tab, refer to the SoMachine Programming Guide.

Digital Output Configuration Parameters

This table presents the function of the different parameters:

Parameter	Function
General Parameters	
Alarm Output	Select an output to be used as alarm output *(see page 87)*.
Rearming Output Mode	Select the rearming output mode *(see page 87)*.
Synchronization	
Minimize jitter for local Output	Select this option to minimize jitter on local outputs *(see page 87)*.

NOTE: The selection is grey and inactive if the parameter is unavailable.

Alarm Output

This output is set to logical 1 when the controller is in the RUNNING state and the application program is not stopped at a breakpoint.

Outputs assigned to configured expert functions can not be configured as the alarm output.

NOTE: The alarm output is set to 0 when a task is stopped at a breakpoint, the alarm output signals that the controller has stopped executing the application.

Rearming Output Mode

Fast outputs of the Modicon M241 Logic Controller use push/pull technology. In case of detected error (short-circuit or over temperature), the output is put in tri-state and the condition is signaled by status bit and PLC_R.i_wLocalIOStatus.

Two behaviors are possible:

- **Automatic rearming**: as soon as the detected error is corrected, the output is set again according to the current value assigned to it and the diagnostic value is reset.
- **Manual rearming**: when an error is detected, the status is memorized and the output is forced to tri-state until user manually clears the status (see I/O mapping channel).

In the case of a short-circuit or current overload, the common group of outputs automatically enters into thermal protection mode (all outputs in the group are set to 0), and are then periodically rearmed (each second) to test the connection state. However, you must be aware of the effect of this rearming on the machine or process being controlled.

⚠ WARNING

UNINTENDED MACHINE START-UP

Inhibit the automatic rearming of outputs if this feature is an undesirable behavior for your machine or process.

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

Minimize Jitter for Local Output

This option allows the embedded I/Os to be read or set at predictable time intervals, regardless of the task duration. Minimizes jitter on outputs by delaying the write to the physical outputs until the read input operation of the next bus cycle task starts. The end time of a task is often less easy to predict than the start time.

Normal scheduling of input/ouput phases is:

When the **Minimize Jitter for Local Output** option is selected, the scheduling of the IN and OUT phases becomes:

Chapter 10
Expert Functions Configuration

Overview

This chapter describes the expert functions of the M241.

What Is in This Chapter?

This chapter contains the following topics:

Expert Functions Overview

Introduction

The fast inputs and fast outputs available on the M241 logic controller can be connected to expert functions.

The M241 logic controller supports the following expert functions:

Functions		Description
Counters	HSC Simple	The HSC functions can execute fast counts of pulses from sensors, switches, etc. that are connected to the fast inputs. The HSC functions are described in the HSC Library Guide *(see Modicon M241 Logic Controller, High Speed Counting, HSC Library Guide)*
	HSC Main Single Phase	
	HSC Main Dual Phase	
	Frequency Meter	
	Period Meter	
Pulse Generators	PTO *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)*	The PTO function provides 2 pulse train output channels to control 2 independent linear single-axis stepper or servo drives in open loop mode.
	PWM *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)*	The PWM function generates a square wave signal on dedicated output channels with a variable duty cycle.
	Frequency Generator *(see Modicon M241 Logic Controller, PTOPWM, Library Guide)*	The frequency generator function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).

NOTE:

- When a regular input is used as Run/Stop, it can not be used by an expert function.
- When a regular output is used as Alarm, it can not be used by an expert function.

For more details, refer to Embedded Functions Configuration *(see page 89)*.

Configuring an Expert Function

To configure an expert function, proceed as follow:

Step	Description
1	Double-click the **Counters** or **Pulse_Generators** node in the **Devices Tree**. **Result:** The **Counters** or **Pulse_Generators** function window appears: 
2	Double-click **Value** and choose the function type to assign. **Result:** The parameters of the expert function appear.

Expert Function I/O Within Regular I/O

Expert function I/O within regular I/O:

- Inputs can be read through standard memory variables even if configured as expert functions.
- An input cannot be configured as an expert function if it has already been configured as a Run/Stop input.
- An output cannot be configured in an expert function if it has already been configured as an alarm.
- Short-Circuit management still applies on all outputs. Status of outputs are available.
- All I/O that are not used by expert functions can be used as any other regular I/O.

When inputs are used in expert functions (Latch, HSC,…), integrator filter is replaced by anti-bounce filter. Filter value is configured in configuration screen.

Counting Function

Overview

The Counting function can execute fast counts of pulses from sensors, encoders, switches, and so on, that are connected to the dedicated fast inputs.

There are 2 types of embedded counting functions:

- **Simple** type: a single input counter.
- **Main** type: a counter that uses up to 6 fast inputs and 2 reflex outputs.

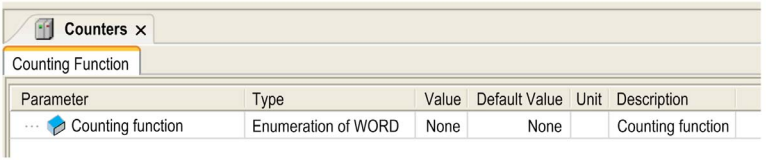
Based on the embedded counting functions, there are 5 types of counters that you can configure in SoMachine:

- **HSC Simple**
- **HSC Main Single Phase**
- **HSC Main Dual Phase**
- **Frequency Meter**
- **Period Meter**

The **Frequency Meter** type and the **Period Meter** type are based on an **HSC Main** type.

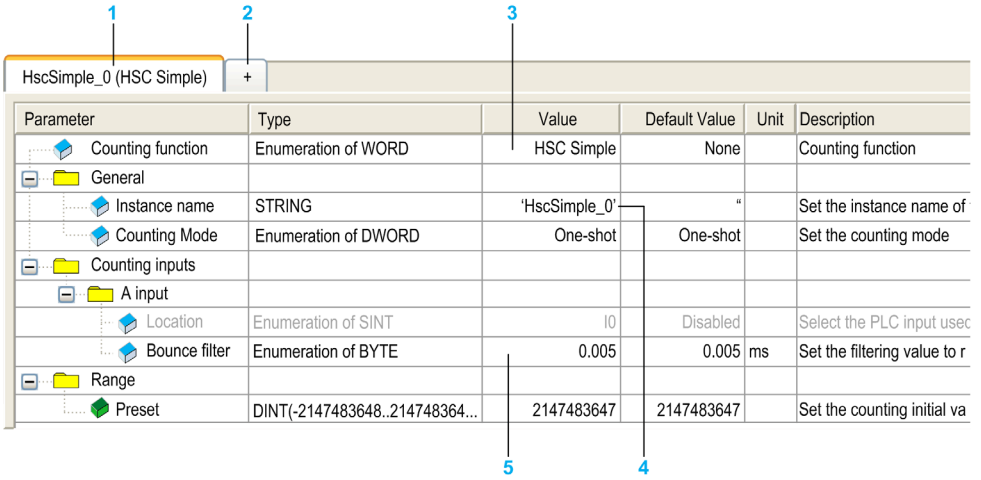
Accessing the Counting Function Configuration Window

Follow these steps to access the embedded counting function configuration window:

Step	Description
1	Double-click **Counters** in the **Devices tree**. The Counting Function window appears: 
2	Double-click **Value** and choose the counting function type to assign.

Counting Function Configuration Window

The following figure shows a sample HSC configuration window:



The following table describes the areas of the **Counters** configuration window:

Number	Action
1	The instance name of the function and the currently configured counting function type .
2	Click **+** to configure a new instance of counting function.
3	Double-click the **Value** column to display a list of the counter function types available.
4	Double-click the **Instance name** value to edit the instance name of the function. The **Instance name** is automatically given by SoMachine. The **Instance name** parameter is editable and allows you to define the instance name. However, whether the **Instance name** is software-defined or user-defined, use the same instance name as an input to the function blocks dealing with the counter, as defined in the **Counters** editor.
5	Configure each parameter by clicking the plus sign next to it to access its settings. The parameters available depend on the mode used.

For detail information on configuration parameters, refer to M241 HSC Library Guide.

Pulse Generators Embedded Function

Overview

The pulse generated embedded functions available with the M241 are:

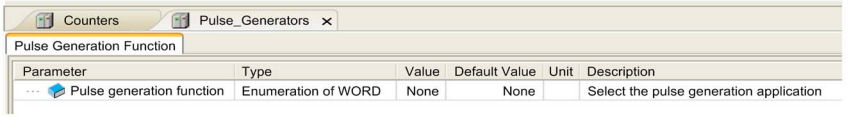
PTO The PTO (Pulse Train Output) implements digital technology that provides precise positioning for open loop control of motor drives.

PWM The PWM (Pulse Width Modulation) function generates a programmable square wave signal on a dedicated output with adjustable duty cycle and frequency.

FG The FG (Frequency Generator) function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).

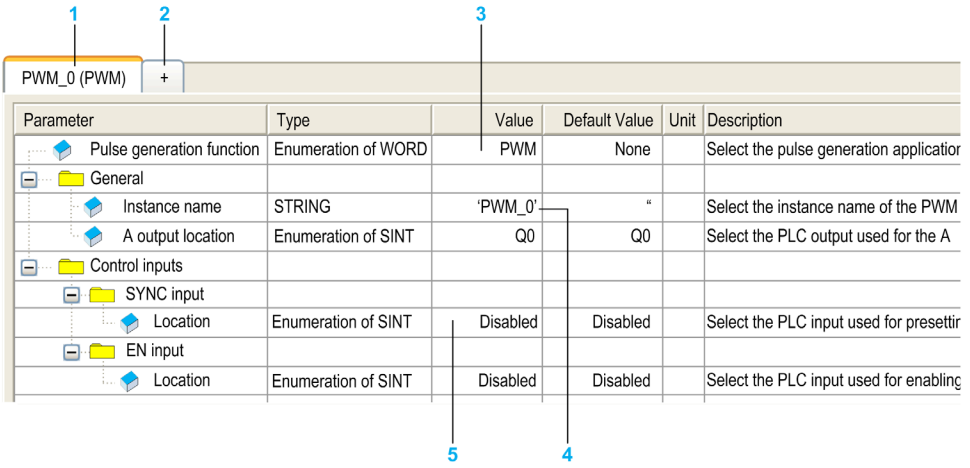
Accessing the Pulse Generators Configuration Window

Follow these steps to access the Pulse Generators configuration window:

Step	Description
1	Double-click **Pulse Generators** on the **Devices tree**. The Pulse Generation Function window appears: 
2	Double-click **Value** and choose the pulse generator function type to assign.

Pulse Generators Configuration Window

The figure shows a sample **Pulse_Generators** configuration window used to configure a PTO, PWM, or FG function:



The following table describes the areas of the **Pulse_Generators** configuration window:

Number	Action
1	The instance name of the function and the currently configured pulse generator function type .
2	Click **+** to configure a new instance of pulse generator function.
3	Double-click the **Value** column to display a list of the pulse generator function types available.
4	Double-click the **Instance name** value to edit the instance name of the function. The **Instance name** is automatically given by SoMachine. The **Instance name** parameter is editable and allows you to define the instance name. However, whether the **Instance name** is software-defined or user-defined, use the same instance name as an input to the function blocks dealing with the counter, as defined in the **Counters** editor.
5	Configure each parameter by clicking the plus sign next to it to access its settings. The parameters available depend on the type of pulse generator used.

For detailed information on configuration parameters, refer to the M241 PTO/PWM/FG Library Guide.

Chapter 11
Cartridge Configuration

TMC4 Cartridge Configuration

Introduction

The Modicon M241 Logic Controller supports the following cartridges:

- TMC4 standard cartridges
- TMC4 application cartridges

For further information about the TMC4 cartridge configuration, refer to the TMC4 Cartridges Programming Guide *(see Modicon TMC4, Cartridges, Programming Guide)*.

⚠ 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.

Adding a TMC4 Cartridge

To add a cartridge to your controller, select the cartridge in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

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)*

Chapter 12
Expansion Modules Configuration

Overview

This chapter describes how to configure the TM4, TM3, and TM2 expansion modules for the Modicon M241 Logic Controller.

What Is in This Chapter?

This chapter contains the following topics:

I/O Configuration General Practices

Match Software and Hardware Configuration

The I/O that may be embedded in your controller is independent of the I/O that you may have added in the form of I/O expansion. It is crucial that the logical I/O configuration within your program matches the physical I/O configuration of your installation. If you add or remove any physical I/O to or from the I/O expansion bus, or, depending on the controller reference, to or from the controller (in the form of cartridges), it is imperative that you update your application configuration. This is also true for any field bus devices you may have in your installation. Otherwise, there is the possibility that the I/O expansions will no longer function while the embedded I/O that may be present in your controller will continue to operate.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Update the configuration of your program each time you add or delete any type of I/O expansions on your I/O bus, or you add or delete any devices on your field bus.

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

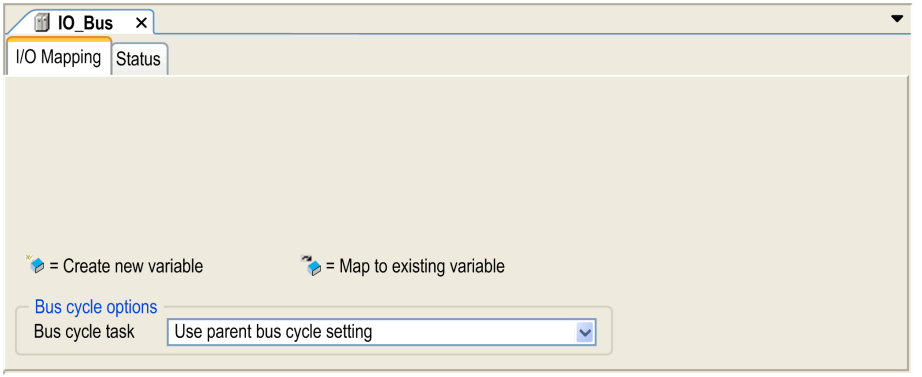
I/O Bus Configuration

Overview

I/O bus configuration provides you the ability to select the task that drives TM3 and CANopen physical exchanges. It can also override the configuration defined in the **PLC settings** *(see page 79)*.

Configuring the I/O Bus

Follow these steps to configure the I/O bus:

Step	Description
1	Double-click **IO_Bus** in the **Devices tree**. **Result**: The **IO_Bus** editor tab appears: 
2	Set the **Bus cycle task** from the list to either of the following: ● **Use parent bus cycle setting** (default) Sets the task for bus exchange as defined in the **PLC settings**. ● **MAST** Sets the Master task for bus exchange irrespective of the task defined in the **PLC settings**.

TM4 Expansion Module Configuration

Introduction

The Modicon M241 Logic Controller supports the TM4 communication expansion modules.

For further information about the TM4 expansion modules configuration, refer to the TM4 Expansion Modules Configuration Programming Guide.

⚠ 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.

Adding an Expansion Module

To add an expansion module to your controller, select the expansion module in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

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)*

TM3/TM2 Expansion Module Configuration

Introduction

The Modicon M241 Logic Controller supports the following expansion modules:

- TM3 expansion modules
  - Digital I/O modules
  - Analog I/O modules
  - Expert I/O modules
  - Safety modules
  - Transmitter and receiver modules
- TM2 expansion modules
  - Digital I/O modules
  - Analog I/O modules
  - Expert modules
  - Communication modules

For further information about the TM3 and TM2 expansion modules configuration, refer to the TM3 Expansion Modules Configuration Programming Guide and TM2 Expansion Modules Configuration Programming Guide respectively.

⚠ 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.

Adding an Expansion Module

To add an expansion module to your controller, select the expansion module in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

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)*

Optional I/O Expansion Modules

Presentation

I/O expansion modules can be marked as optional in the configuration. The **Optional module** feature provides a more flexible configuration by the acceptance of the definition of modules that are not physically attached to the logic controller. Therefore, a single application can support multiple physical configurations of I/O expansion modules, allowing a greater degree of scalability without the necessity of maintaining multiple application files for the same application.

Without the **Optional module** feature, when the logic controller starts up the I/O expansion bus (following a power cycle, application download or initialization command), it compares the configuration defined in the application with the physical I/O modules attached to the I/O bus. Among other diagnostics made, if the logic controller determines that there are I/O modules defined in the configuration that are not physically present on the I/O bus, an error is detected and the I/O bus does not start.

With the **Optional module** feature, the logic controller ignores the absent I/O expansion modules that you have marked as optional, which then allows the logic controller to start the I/O expansion bus.

The logic controller starts the I/O expansion bus at configuration time (following a power cycle, application download, or initialization command) even if optional expansion modules are not physically connected to the logic controller.

The following module types can be marked as optional:
- TM3 I/O expansion modules
- TM2 I/O expansion modules

NOTE: TM3 Transmitter/Receiver modules (TM3XTRA1 and the TM3XREC1) and TMC4 cartridges cannot be marked as optional.

You must be fully aware of the implications and impacts of marking I/O modules as optional in your application, both when those modules are physically absent and present when running your machine or process. Be sure to include this feature in your risk analysis.

⚠ WARNING

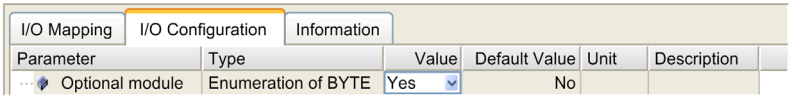
UNINTENDED EQUIPMENT OPERATION

Include in your risk analysis each of the variations of I/O configurations that can be realized marking I/O expansion modules as optional, and in particular the establishment of TM3 Safety modules (TM3S…) as optional I/O modules, and make a determination whether it is acceptable as it relates to your application.

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

Marking an I/O Expansion Module as Optional

To add an expansion module and mark it as optional in the configuration:

Step	Action
1	Add the expansion module to your controller.
2	Double-click the expansion module in the **Devices tree**.
3	Select the **I/O Configuration** tab
4	In the **Optional module** line, select **Yes** in the **Value** column: 

Shared Internal ID Codes

Logic controllers identify expansion modules by a simple internal ID code. This ID code is not specific to each reference, but identifies the structure of the expansion module. Therefore, different references can share the same ID code.

You cannot have two modules with the same internal ID code declared as optional without at least one mandatory module placed between them.

This table groups the module references sharing the same internal ID code:

Modules sharing the same internal ID code
TM2DDI16DT, TM2DDI16DK
TM2DRA16RT, TM2DDO16UK, TM2DDO16TK
TM2DDI8DT, TM2DAI8DT
TM2DRA8RT, TM2DDO8UT, TM2DDO8TT
TM2DDO32TK, TM2DDO32UK
TM3DI16K, TM3DI16/G
TM3DQ16R/G, TM3DQ16T/G, TM3DQ16TK, TM3DQ16U, TM3DQ16UG, TM3DQ16UK
TM3DQ32TK, TM3DQ32UK
TM3DI8/G, TM3DI8A
TM3DQ8R/G, TM3DQ8T/G, TM3DQ8U, TM3DQ8UG
TM3DM8R/G
TM3DM24R/G
TM3SAK6R/G
TM3SAF5R/G
TM3SAC5R/G
TM3SAFL5R/G

Modules sharing the same internal ID code
TM3AI2H/G
TM3AI4/G
TM3AI8/G
TM3AQ2/G
TM3AQ4/G
TM3AM6/G
TM3TM3/G
TM3TI4/G
TM3TI8T/G

Chapter 13
Ethernet Configuration

Introduction

This chapter describes how to configure the Ethernet network interface of the Modicon M241 Logic Controller.

What Is in This Chapter?

This chapter contains the following sections:

Section 13.1
Ethernet Services

What Is in This Section?

This section contains the following topics:

Presentation

Ethernet Services

The controller supports the following services:

- Modbus TCP Server *(see page 116)*
- Modbus TCP Client *(see page 116)*
- Web Server *(see page 118)*
- FTP Server *(see page 132)*
- SNMP *(see page 134)*
- EtherNet/IP Device *(see page 147)*
- Modbus TCP Slave Device *(see page 169)*
- IEC VAR ACCESS *(see page 110)*

Ethernet Protocols

The controller supports the following protocols:

- IP (Internet Protocol)
- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- ARP (Address Resolution Protocol)
- ICMP (Internet Control Messaging Protocol)
- IGMP (Internet Group Management Protocol)

Connections

This table shows the maximum number of connections:

Connection Type	Maximum Number of Connections
Modbus Server	8
Modbus Client	8
EtherNet/IP Target	16
FTP Server	4
Web Server	10
SoMachine Protocol (SoMachine software, trace, Web visualization, HMI devices)	8

Each connection based on TCP manages its own set of connections as follows:

1. When a client tries to open a connection that exceeds the poll size, the controller closes the oldest connection.
2. If all connections are busy (exchange in progress) when a client tries to open a new one, the new connection is denied.
3. All server connections stay open as long as the controller stays in operational states (`RUNNING`, `STOPPED`, `HALT`).
4. All server connections are closed when leaving or entering operational states (`RUNNING`, `STOPPED`, `HALT`), except in case of power outage (because the controller does not have time to close the connections).

Services Available

With an Ethernet communication, the **IEC VAR ACCESS** service is supported by the controller. With the **IEC VAR ACCESS** service, data can be exchanged between the controller and an HMI.

The **NetWork variables** service is also supported by the controller. With the **NetWork variables** service, data can be exchanged between controllers.

NOTE: For more information, refer to the SoMachine Programming Guide.

IP Address Configuration

Introduction

There are different ways to assign the IP address of the controller:

- address assignment by DHCP server
- address assignment by BOOTP server
- fixed IP address
- post configuration file *(see page 197)*. If a post configuration file exists, this assignment method has priority over the others.

The IP address can be changed dynamically:

- via the Controller Selection *(see SoMachine, Programming Guide)* tab in SoMachine.

NOTE: If the attempted addressing method is unsuccessful, the controller will start using a default IP address *(see page 114)* derived from the MAC address.

Carefully manage the IP addresses because each device on the network requires a unique address. Having multiple devices with the same IP address can cause unintended operation of your network and associated equipment.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

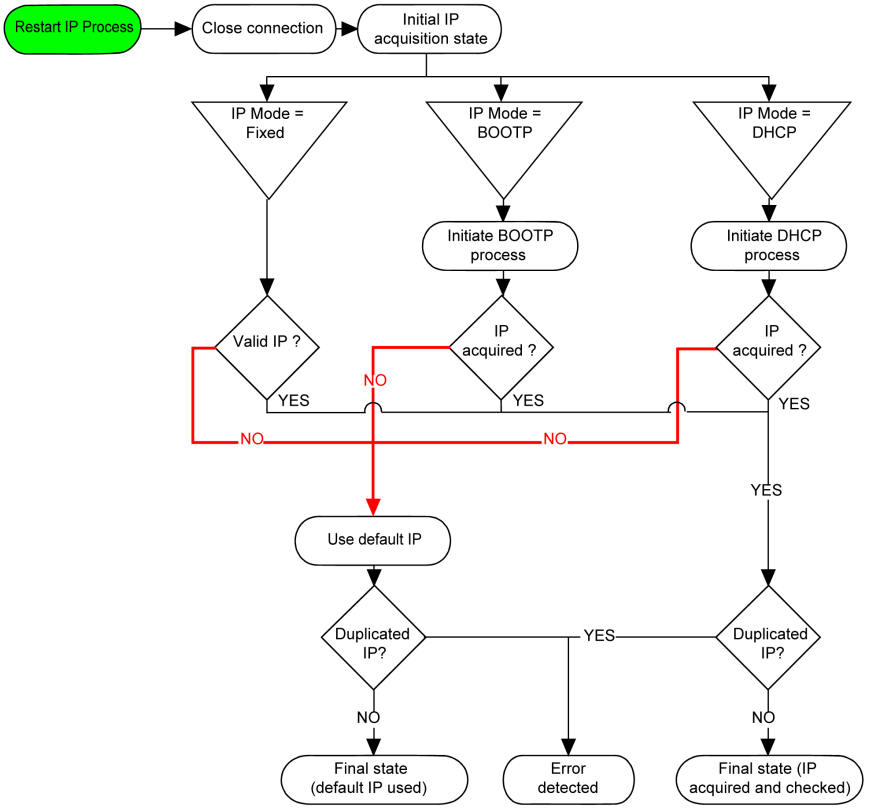
- Verify that there is only one master controller configured on the network or remote link.
- Verify that all devices have unique addresses.
- Obtain your IP address from your system administrator.
- Confirm that the IP address of the device is unique before placing the system into service.
- Do not assign the same IP address to any other equipment on the network.
- Update the IP address after cloning any application that includes Ethernet communications to a unique address.

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

NOTE: Verify that your system administrator maintains a record of all assigned IP addresses on the network and subnetwork, and inform the system administrator of all configuration changes performed.

Address Management

The different types of address systems for the controller are shown in this diagram:



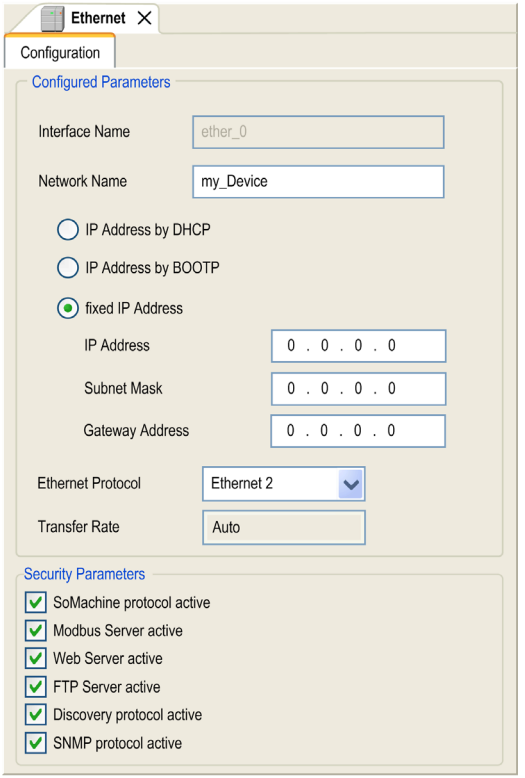
NOTE: If a device programmed to use the DHCP or BOOTP addressing methods is unable to contact its respective server, the controller uses the default IP address. It will, however, constantly repeat its request.

The IP process restarts in the following cases:

- Controller reboot
- Ethernet cable reconnection
- Application download (if IP parameters change)
- DHCP or BOOTP server detected after a prior addressing attempt was unsuccessful.

Ethernet Configuration

In the **Devices tree**, double-click **Ethernet_x**:



The configured parameters are explained as below:

Configured Parameters	Description
Interface Name	Name of the network link.
Network Name	Used as device name to retrieve IP address through DHCP, maximum 16 characters.
IP Address by DHCP	IP address is obtained via DHCP.
IP Address by BOOTP	IP address is obtained via BOOTP.
Fixed IP Address	IP address, Subnet Mask, and Gateway Address are defined by the user.
Ethernet Protocol	Protocol type used (Ethernet2 or IEEE 802.3) **NOTE:** If you change the Ethernet Protocol, a power cycle is required before it will be recognized by the controller.
Transfer Rate	Transfer rate and direction on the bus are automatically configured.

Default IP Address

The IP address by default is 10.10.x.x.

The last 2 fields in the default IP address are composed of the decimal equivalent of the last 2 hexadecimal bytes of the MAC address of the port.

The MAC address of the port can be retrieved on the label placed on the front side of the controller.

The default subnet mask is Default Class A Subnet Mask of 255.0.0.0.

NOTE: A MAC address is always written in hexadecimal format and an IP address in decimal format. Convert the MAC address to decimal format.

Example: If the MAC address is 00.80.F4.01.80.F2, the default IP address is 10.10.128.242.

NOTE: To take into account the new IP address after the download of a project, reboot the controller by doing a power cycle.

Address Classes

The IP address is linked:

- to a device (the host)
- to the network to which the device is connected

An IP address is always coded using 4 bytes.

The distribution of these bytes between the network address and the device address may vary.This distribution is defined by the address classes.

The different IP address classes are defined in this table:

Address Class	Byte1					Byte 2	Byte 3	Byte 4
Class A	0	Network ID				Host ID		
Class B	1	0	Network ID				Host ID	
Class C	1	1	0	Network ID				Host ID
Class D	1	1	1	0	Multicast Address			
Class E	1	1	1	1	0	Address reserved for subsequent use		

Subnet Mask

The subnet mask is used to address several physical networks with a single network address. The mask is used to separate the subnetwork and the device address in the host ID.

The subnet address is obtained by retaining the bits of the IP address that correspond to the positions of the mask containing 1, and replacing the others with 0.

Conversely, the subnet address of the host device is obtained by retaining the bits of the IP address that correspond to the positions of the mask containing 0, and replacing the others with 1.

Example of a subnet address:

IP address	192 (11000000)	1 (00000001)	17 (00010001)	11 (00001011)
Subnet mask	255 (11111111)	255 (11111111)	240 (11110000)	0 (00000000)
Subnet address	192 (11000000)	1 (00000001)	16 (00010000)	0 (00000000)

NOTE: The device does not communicate on its subnetwork when there is no gateway.

Gateway Address

The gateway allows a message to be routed to a device that is not on the current network.

If there is no gateway, the gateway address is 0.0.0.0.

Security Parameters

Security Parameters	Description
SoMachine protocol active	It allows you to deactivate the SoMachine protocol on Ethernet interfaces. When deactivated, every SoMachine request from every device will be rejected, including those from the UDP or TCP connection. Therefore, no connection is possible on Ethernet from a PC with SoMachine, from an HMI target that wants to exchange variables with this controller, from an OPC server, or from Controller Assistant.
Modbus Server active	It allows you to deactivate the Modbus Server of the Logic Controller. Therefore, every Modbus request to the Logic Controller will be ignored.
Web Server active	It allows you to deactivate the Web Server of the Logic Controller. Therefore, every HTTP request to the Logic Controller will be ignored.
FTP Server active	It allows you to deactivate the FTP Server of the Logic Controller. Therefore, every FTP request will be ignored.
Discovery protocol active	It allows you to deactivate Discovery protocol. Therefore, every Discovery request will be ignored.
SNMP protocol active	It allows you to deactivate SNMP server of the Logic Controller. Therefore, every SNMP request will be ignored.

Modbus TCP Client/Server

Introduction

Unlike Modbus serial link, Modbus TCP is not based on a hierarchical structure, but on a client/server model.

The Modicon M241 Logic Controller implements both client and server services so that it can initiate communications to other controllers and I/O devices, and to respond to requests from other controllers, SCADA, HMIs, and other devices.

Without any configuration, the embedded Ethernet port of the controller supports Modbus server.

The Modbus client/server is included in the firmware and does not require any programming action from the user. Due to this feature, it is accessible in RUNNING, STOPPED and EMPTY states.

Modbus TCP Client

The Modbus TCP client supports the following function blocks from the PLCCommunication library without any configuration:

- ADDM
- READ_VAR
- SEND_RECV_MSG
- SINGLE_WRITE
- WRITE_READ_VAR
- WRITE_VAR

For further information, refer to the Function Block Descriptions *(see SoMachine, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.

Modbus TCP Server

The Modbus server supports the Modbus requests:

Function Code Dec (Hex)	Subfunction Dec (Hex)	Function
1 (1)	–	Read digital outputs (%Q)
2 (2)	–	Read digital inputs (%I)
3 (3)	–	Read holding register (%MW)
6 (6)	–	Write single register (%MW)
8 (8)	–	Diagnostic
15 (F)	–	Write multiple digital outputs (%Q)
16 (10)	–	Write multiple registers (%MW)
23 (17)	–	Read/write multiple registers (%MW)
43 (2B)	14 (E)	Read device identification

Diagnostic Request

This table contains the data selection code list:

Data Selection Code (hex)	Description
00	Reserved
01	Basic Network Diagnostics
02	Ethernet Port Diagnostic
03	Modbus TCP/Port 502 Diagnostics
04	Modbus TCP/Port 502 Connection Table
05 - 7E	Reserved for other public codes
7F	Data Structure Offsets

Web Server

Introduction

The controller provides as a standard equipment an embedded Web server with a predefined factory built-in website. You can use the pages of the website for module setup and control as well as application diagnostics and monitoring. These pages are ready to use with a Web browser. No configuration or programming is required.

The Web server can be accessed by the web browsers listed below:

- Google Chrome (version 30.0 or higher)
- Mozilla Firefox (version 1.5 or higher)

The Web server is limited to 10 TCP connections *(see page 110)*.

NOTE: The Web server can be disabled by unchecking the **Web Server active** parameter in the Ethernet Configuration tab *(see page 113)*.

The Web server is a tool for reading and writing data, and controlling the state of the controller, with full access to all data in your application. However, if there are security concerns over these functions, you must at a minimum assign a secure password to the Web Server or disable the Web server to prevent unauthorized access to the application. By enabling the Web server, you enable these functions.

The Web server allows you to monitor a controller and its application remotely, to perform various maintenance activities including modifications to data and configuration parameters, and change the state of the controller. Care must be taken to ensure that the immediate physical environment of the machine and process is in a state that will not present safety risks to people or property before exercising control remotely.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Configure and install the RUN/STOP input for the application, if available for your particular controller, so that local control over the starting or stopping of the controller can be maintained regardless of the remote commands sent to the controller.
- Define a secure password for the Web Server, and do not allow unauthorized or otherwise unqualified personnel to use this feature.
- Ensure that there is a local, competent, and qualified observer present when operating on the controller from a remote location.
- You must have a complete understanding of the application and the machine/process it is controlling before attempting to adjust data, stopping an application that is operating, or starting the controller remotely.
- Take the precautions necessary to assure that you are operating on the intended controller by having clear, identifying documentation within the controller application and its remote connection.

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

NOTE: The Web server must only be used by authorized and qualified personnel. A qualified person is one who has the skills and knowledge related to the construction and operation of the machine and the process controlled by the application and its installation, and has received safety training to recognize and avoid the hazards involved. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this feature.

Web Server Access

Access to the Web server is controlled by User Rights when they are enabled in the controller. For more information, refer to **Users and Groups** Tab Description *(see page 76)*.

If User Rights are not enabled in the controller, you are prompted for a user name and password unique to the FTP/Web server. The default user name is USER and the default password is also USER.

NOTE: You cannot modify the default user name and password. To secure the FTP/Web server functions, you must do so with **Users and Groups**.

⚠ WARNING

UNAUTHORIZED DATA ACCESS

- Secure access to the FTP/Web server using User Rights.
- If you do not enable User Rights, disable the FTP/Web server to prevent any unwanted or unauthorized access to data in your application.

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

In order to change the password, go to **Users and Groups** tab of the device editor. For more information, refer to the SoMachine Programming Guide.

NOTE: The only way to gain access to a controller that has user access-rights enabled and for which you do not have the password(s) is by performing an Update Firmware operation. This clearing of User Rights can only be accomplished by using a SD card or USB key (depending on the support of your particular controller) to update the controller firmware. In addition, you may clear the User Rights in the controller by running a script (for more information, refer to SoMachine Programming Guide). This effectively removes the existing application from the controller memory, but restores the ability to access the controller.

Home Page Access

To access the website home page, type in your navigator the IP address of the controller.

This figure shows the Web Server site login page:

This figure shows the home page of the Web Server site once you have logged in:

NOTE: Schneider Electric follows, and recommends to its customers, industry best practices in the development and implementation of control systems. This recommendation includes a "Defense-in-Depth" approach to secure an Industrial Control System. This approach places the controllers behind one or more firewalls to restrict access to authorized personnel and protocols only.

⚠ WARNING

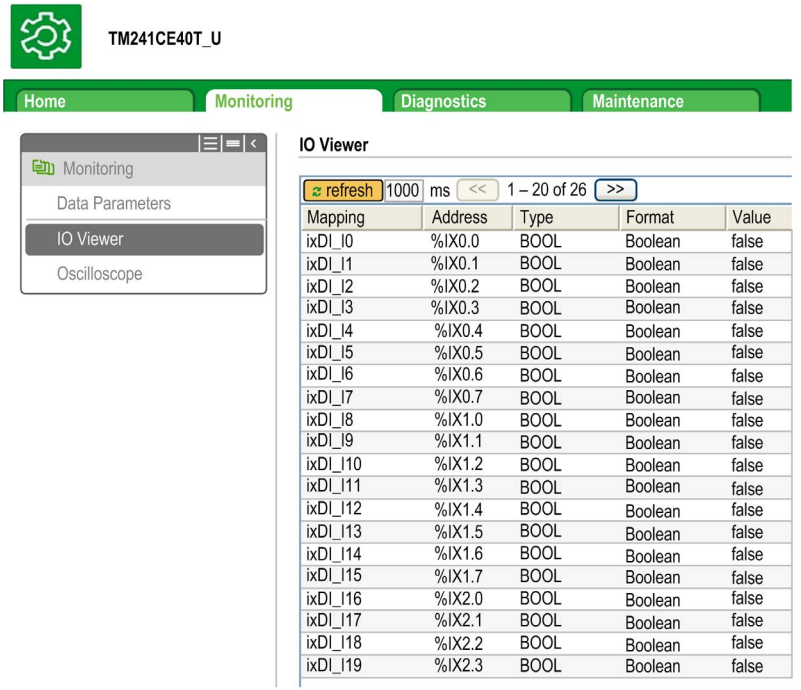
UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

- Evaluate whether your environment or your machines are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Prepare a recovery plan including backup of your system and process information.

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

Monitoring: IO Viewer Submenu

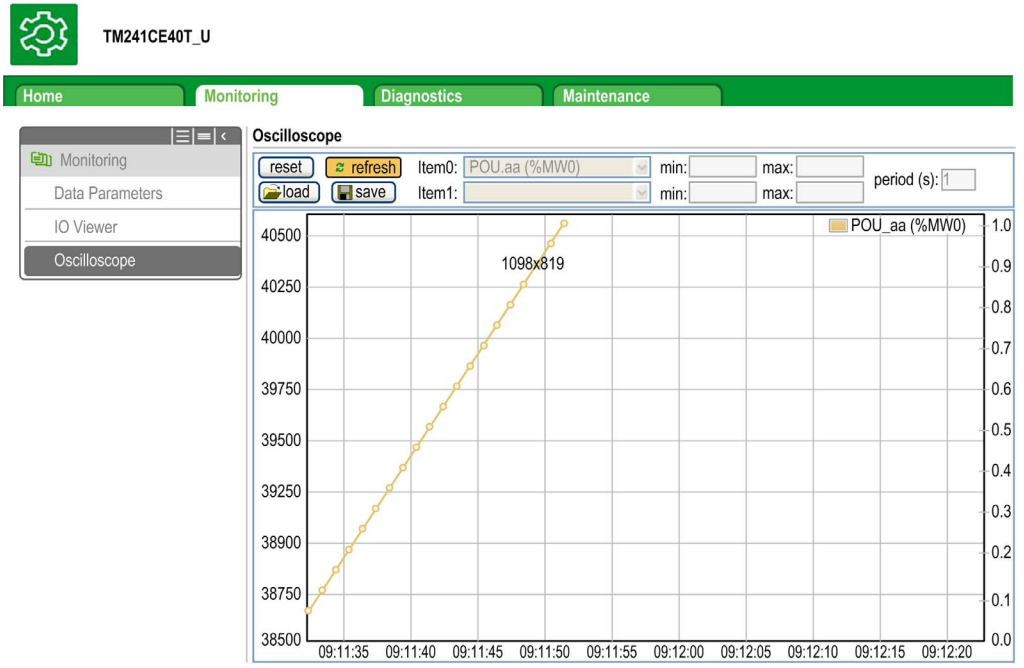
The **IO Viewer** allows you to display and modify the current I/O values:



Element	Description
Refresh	Enables I/O refreshing: ● gray button: refreshing disabled ● orange button: refreshing enabled
1000 ms	I/O refreshing period in ms
<<	Goes to previous I/O list page
>>	Goes to next I/O list page

Monitoring: Oscilloscope Submenu

The **Oscilloscope** page can display up to 2 variables in the form of a recorder time chart:



Element	Description
Reset	Erases the memorization
Refresh	Starts/stops refreshing
Load	Loads parameter configuration of Item0 and Item1
Save	Saves parameter configuration of Item0 and Item1 in the controller
Item0	Variable to be displayed
Item1	Variable to be displayed
Min	Minimum value of the variable axis
Max	Maximum value of the variable axis
Period(s)	Page refresh period in seconds

Monitoring: Data Parameters

Monitoring variables in the Web Server

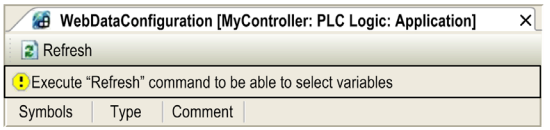
To monitor variables in the web server, you should add a **Web Data Configuration** object to your project. Within this object, you can select all variables you want to monitor.

This table describes how to add a **Web Data Configuration** object:

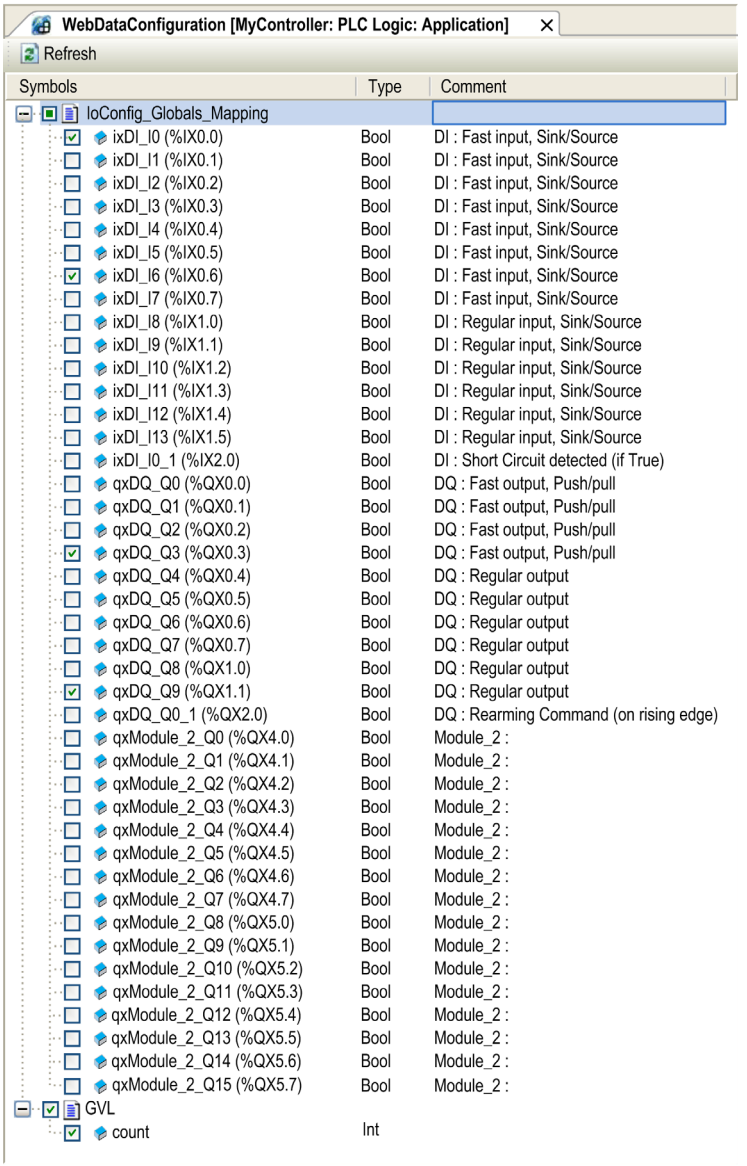
Step	Action
1	Right click the **Application** node in the **Applications tree** tab.
2	Click **Add Object** → **Web Data Configuration...**. **Result:** The **Add Web Data Configuration** window is displayed.
3	Click **Add**. **Result:** The **Web Data Configuration** object is created and the **Web Data Configuration** editor is open. **NOTE:** As a **Web Data Configuration** object is unique for a controller, its name cannot be changed.

Web Data Configuration Editor

Click the **Refresh** button to be able to select variables, this action will display all the variables defined in the application.



Select the variables you want to monitor in the web server:



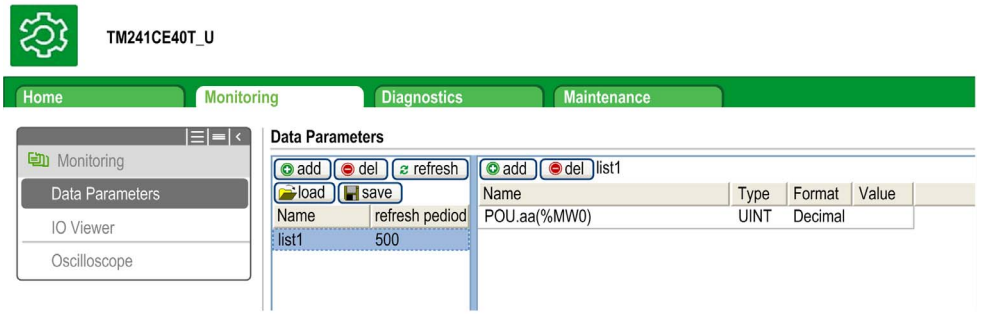
NOTE: The variable selection is possible only in offline mode.

Monitoring: Data Parameters Submenu

The **Data Parameters** page allows you to create and monitor some lists of variables. You can create several lists of variables (maximum 10 lists), each one containing several variables of the controller application (maximum 20 variables per list).

Each list has a name, and a refresh period. The lists are saved in the Flash memory of the controller, so that a created list can be accessed (loaded, modified, saved) from any Web client application accessing this controller.

The **Data Parameters** allows you to display and modify variable values:

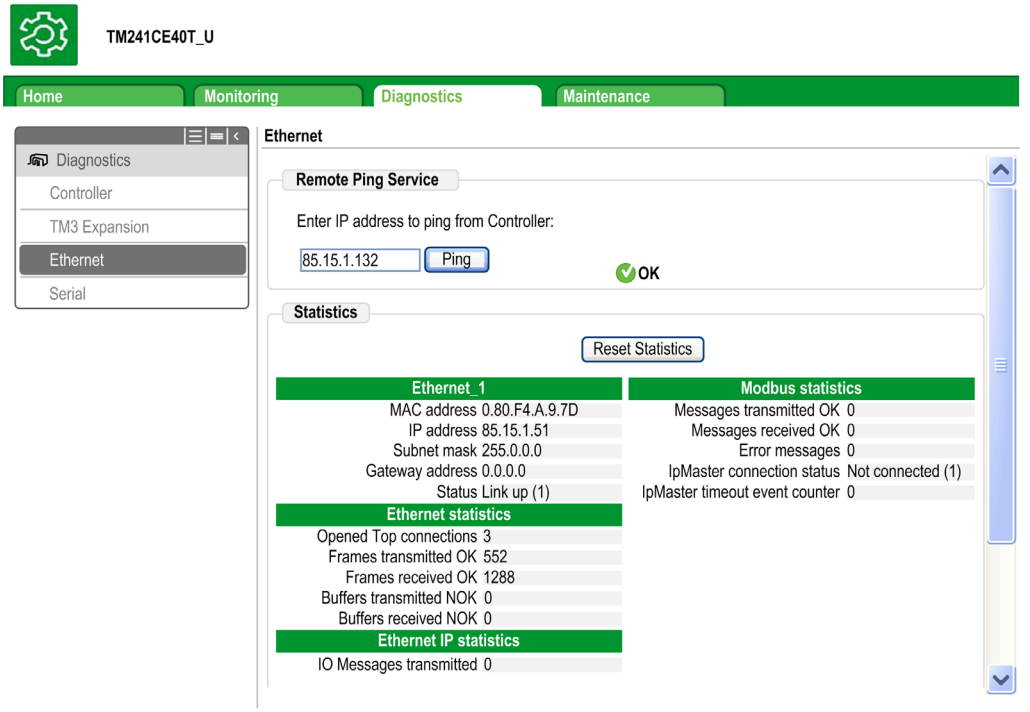


Element	Description
Load	Loads saved lists from the controller internal Flash to the web server page
Save	Saves the selected list description in the controller (*/usr/web* directory)
Add	Adds a list description or a variable
Del	Deletes a list description or a variable
Refresh period	Refreshing period of the variables contained in the list description (in ms)
Refresh	Enables I/O refreshing: ● gray button: refreshing disabled ● orange button: refreshing enabled

NOTE: IEC objects (%IW, %M,...) are not directly accessible. To access IEC objects you must first group their contents in located registers (refer to Relocation Table *(see page 36)*).

Diagnostics: Ethernet Submenu

This figure shows the remote ping service:



Maintenance Tab

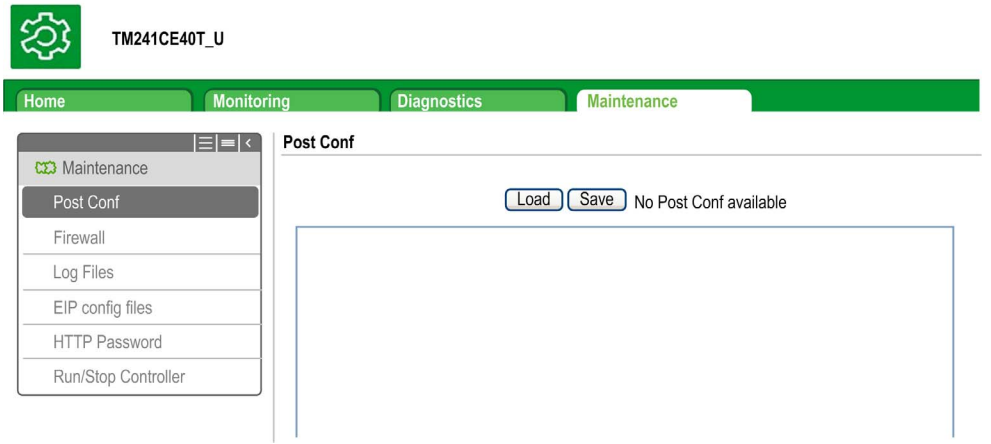
The Maintenance page provides access to the `/usr` and `/sys` folders of the controller flash memory *(see page 32)*:

Index of `/usr`:

Index of `/sys`:

Maintenance: Post Conf Submenu

The **Post Conf** page allows you to update the post configuration file *(see page 197)* saved on the controller:

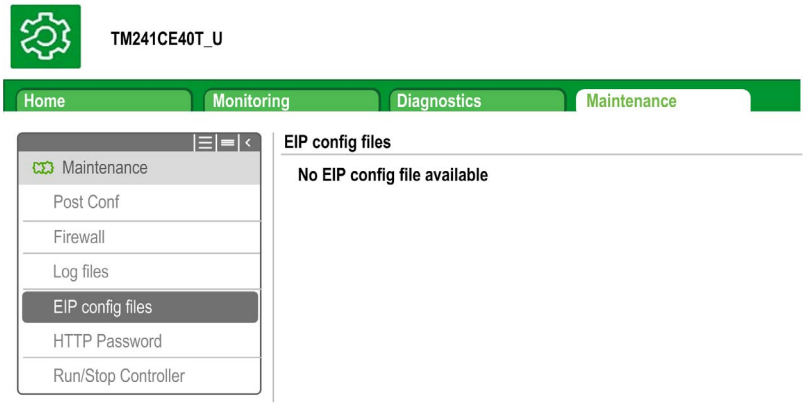


Step	Action
1	Click **Load**.
2	Modify the parameters *(see page 201)*.
3	Click **Save**. **NOTE:** The new parameters will be considered at next Post Configuration file reading *(see page 199)*.

Maintenance: EIP Config Files Submenu

The file tree only appears when the Ethernet IP service is configured on the controller.

Index of `/usr`:



File	Description
My Machine Controller.gz	GZIP file
My Machine Controller.ico	Icon file
My Machine Controller.eds	Electronic Data Sheet file

FTP Server

Introduction

Any FTP client installed on a computer that is connected to the controller (Ethernet port), without SoMachine installed, can be used to transfer files to and from the data storage area of the controller.

NOTE: Schneider Electric follows, and recommends to its customers, industry best practices in the development and implementation of control systems. This recommendation includes a "Defense-in-Depth" approach to secure an Industrial Control System. This approach places the controllers behind one or more firewalls to restrict access to authorized personnel and protocols only.

⚠ WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

- Evaluate whether your environment or your machines are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Prepare a recovery plan including backup of your system and process information.

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

NOTE: Make use of the security-related commands *(see SoMachine, Programming Guide)* which provide a way to add, edit, and remove a user in the online user management of the target device where you are currently logged in.

The FTP server is available even if the controller is empty (no user application and no User Rights are enabled).

FTP Access

Access to the FTP server is controlled by User Rights when they are enabled in the controller. For more information, refer to **Users and Groups** Tab Description *(see page 76)*.

If User Rights are not enabled in the controller, you are prompted for a user name and password unique to the FTP/Web server. The default user name is USER and the default password is also USER.

NOTE: You cannot modify the default user name and password. To secure the FTP/Web server functions, you must do so with **Users and Groups**.

⚠ WARNING

UNAUTHORIZED DATA ACCESS

- Secure access to the FTP/Web server using User Rights.
- If you do not enable User Rights, disable the FTP/Web server to prevent any unwanted or unauthorized access to data in your application.

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

In order to change the password, go to **Users and Groups** tab of the device editor. For more information, refer to the SoMachine Programming Guide.

NOTE: The only way to gain access to a controller that has user access-rights enabled and for which you do not have the password(s) is by performing an Update Firmware operation. This clearing of User Rights can only be accomplished by using a SD card or USB key (depending on the support of your particular controller) to update the controller firmware. In addition, you may clear the User Rights in the controller by running a script (for more information, refer to SoMachine Programming Guide). This effectively removes the existing application from the controller memory, but restores the ability to access the controller.

Files Access

See File Organization *(see page 32)*.

SNMP

Introduction

The SNMP protocol (Simple Network Management Protocol) is used to provide the data and services required for managing a network.

The data is stored in an MIB (Management Information Base). The SNMP protocol is used to read or write MIB data. Implementation of the Ethernet SNMP services is minimal, as only the compulsory objects are handled.

M241 controllers support the standard MIB-2 objects.

SNMP Object Handle

Object	Description	Access	Default Value
sysDescr	Text description of the device	Read	SCHNEIDER M241-51 Fast Ethernet TCP/IP
sysName	Node administrative name	Read/Write	Controller reference

The values written are saved to the controller via SNMP client tool software. The Schneider Electric software for this is ConneXview. ConneXview is not supplied with the controller. For more details, refer to www.schneider-electric.com.

The size of these character strings is limited to 50 characters.

Section 13.2
Firewall Configuration

Introduction

This section describes how to configure the firewall of the Modicon M241 Logic Controller.

What Is in This Section?

This section contains the following topics:

Introduction

Firewall Presentation

In general, firewalls help protect network security zone perimeters by blocking unauthorized access and permitting authorized access. A firewall is a device or set of devices configured to permit, deny, encrypt, decrypt, or proxy traffic between different security zones based upon a set of rules and other criteria.

Process control devices and high-speed manufacturing machines require fast data throughput and often cannot tolerate the latency introduced by an aggressive security strategy inside the control network. Firewalls, therefore, play a significant role in a security strategy by providing levels of protection at the perimeters of the network. Firewalls are important part of an overall, system level strategy.

NOTE: Schneider Electric follows, and recommends to its customers, industry best practices in the development and implementation of control systems. This recommendation includes a "Defense-in-Depth" approach to secure an Industrial Control System. This approach places the controllers behind one or more firewalls to restrict access to authorized personnel and protocols only.

⚠ WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

- Evaluate whether your environment or your machines are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Prepare a recovery plan including backup of your system and process information.

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

Firewall Configuration

There are 3 ways to manage the controller firewall configuration:

- Static configuration,
- Dynamic changes,
- Application settings.

Script files are used in the static configuration and for dynamic changes.

Static Configuration

The static configuration is loaded at the controller boot.

The controller firewall can be statically configured by managing a default script file located in the controller. The path to this file is */Usr/Cfg/FirewallDefault.cmd*.

Dynamic Changes

After the controller boot, the controller firewall configuration can be changed by the use of script files.

There are 2 ways to load these dynamic changes:

- Using a physical SD card *(see page 138)*,
- Using a function block *(see page 138)* in the application.

Application Settings

Refer to Ethernet Configuration *(see page 113)*.

Dynamic Changes Procedure

Using an SD Card

This table describes the procedure to execute a script file from an SD card:

Step	Action
1	Create a valid script file *(see page 141)*. For instance, name the script file *FirewallMaintenance.cmd*.
2	Load the script file on the SD card. For instance, load the script file in the *Usr/cfg* folder.
3	In the file *Sys/Cmd/Script.cmd*, add a code line with the command `Firewall_install "pathname/FileName"` For instance, the code line is `Firewall_install "/sd0/Usr/cfg/FirewallMaintenace.cmd"`
4	Insert the SD card on the controller.

Using a Function Block in the Application

This table describes the procedure to execute a script file from an application:

Step	Action
1	Create a valid script file *(see page 141)*. For instance, name the script file *FirewallMaintenance.cmd*.
2	Load the script file in the controller memory. For instance, load the script file in the *Usr/Syslog* folder with FTP.
3	Use an ExecuteScript *(see Modicon M241 Logic Controller, System Functions and Variables, PLCSystem Library Guide)* function block. For instance, the **[SCmd]** input is `'Firewall_install "/usr/Syslog/FirewallMaintenace.cmd"'`

Firewall Behavior

Introduction

The firewall configuration depends on the action done on the controller and the initial configuration state. There are 5 possible initial states:

- There is no default script file in the controller.
- A correct script file is present.
- An incorrect script file is present.
- There is no default script file and the application has configured the firewall.
- A dynamic script file configuration has been already executed.

No Default Script File

If...	Then ...
Boot of the controller	Firewall is not configured. No protection is activated.
Execute dynamic script file	Firewall is configured according to the dynamic script file.
Execute dynamic incorrect script file	Firewall is not configured. No protection is activated.
Download application	Firewall is configured according to the application settings.

Default Script File Present

If...	Then ...
Boot of the controller	Firewall is configured according to the default script file.
Execute dynamic script file	The whole configuration of the default script file is deleted. Firewall is configured according to the dynamic script file.
Execute dynamic incorrect script file	Firewall is configured according to the default script file. The dynamic script file is not taken into account.
Download application	The whole configuration of the application is ignored. Firewall is configured according to the default script file.

Incorrect Default Script File Present

If...	Then ...
Boot of the controller	Firewall is not configured. No protection is activated
Execute dynamic script file	Firewall is configured according to the dynamic script file.
Execute dynamic incorrect script file	Firewall is not configured. No protection is activated.
Download application	Firewall is configured according to the application settings.

Application Settings with No Default Script File

If...	Then ...
Boot of the controller	Firewall is configured according to the application settings.
Execute dynamic script file	The whole configuration of the application settings is deleted. Firewall is configured according to the dynamic script file.
Execute dynamic incorrect script file	Firewall is configured according to the application settings. The dynamic script file is not taken into account.
Download application	The whole configuration of the previous application is deleted. Firewall is configured according to the new application settings.

Execute Dynamic Script File Already Executed

If...	Then ...
Boot of the controller	Firewall is configured according to the dynamic script file configuration (see note).
Execute dynamic script file	The whole configuration of the previous dynamic script file is deleted. Firewall is configured according to the new dynamic script file.
Execute dynamic incorrect script file	Firewall is configured according to the previous dynamic script file configuration. The dynamic incorrect script file is not taken into account.
Download application	The whole configuration of the application is ignored Firewall is configured according to the dynamic script file.
NOTE: If an SD card containing a cybersecurity script is plugged into the controller, booting is blocked. First remove the SD card to correctly boot the controller.	

Script File Syntax

Overview

This section describes how script files (default script file or dynamic script file) are written so that they can be executed correctly during the booting of the controller or during a specific command triggered by the user.

General Writing Guideline

End every line of a command in the script with a "`;`".

If the line begins with a "`;`", the line is a comment.

The maximum number of lines in a script file is 50.

The syntax is not case-sensitive.

If the syntax is not respected in the script file, the script file is not executed at all. It means that the firewall configuration remains in the previous state.

NOTE: If the script file is not executed, a log file is generated. The log file location in the controller is */usr/Syslog/FWLog.txt*.

Firewall General Commands

Command	Description
`FireWall enable`	Blocks all frames from the Ethernet interfaces. If no IP address is further authorized, it is not possible to communicate on the Ethernet interfaces. **NOTE:** By default, when the Firewall is enabled, all frames are rejected.
`FireWall Disable`	All IP addresses are allowed to access to the controller on all Ethernet interfaces.
`FireWall Eth1 Default Enable`	All frames are accepted by the controller.
`FireWall Eth1 Default Reject`	All frames are rejected by the controller. **NOTE:** By default, if this line is not present, it corresponds to the command `FireWall Eth1 Default Reject`.
NOTE: The number of lines written in a script file must not exceed 50.	

Firewall Specific Commands

Command	Range	Description
`Firewall Eth1 Allow IP •.•.•.•`	• = 0...255	All frames from the mentioned IP address are allowed on all port numbers and port types.
`Firewall Eth1 Reject IP •.•.•.•`	• = 0...255	All frames from the mentioned IP address are rejected on all port numbers and port types.
`Firewall Eth1 Allow IPs •.•.•.• to •.•.•.•`	• = 0...255	All frames from the IP addresses in the mentioned range are allowed for all port numbers and port types.
`Firewall Eth1 Reject IPs •.•.•.• to •.•.•.•`	• = 0...255	All frames from the IP addresses in the mentioned range are rejected for all port numbers and port types.
`Firewall Eth1 Allow port_type port Y`	Y = (destination port numbers *(see page 144)*)	All frames with the mentioned destination port number are allowed.
`Firewall Eth1 Reject port_type port Y`	Y = (destination port numbers *(see page 144)*)	All frames with the mentioned destination port number are allowed.
`Firewall Eth1 Allow port_type ports Y1 to Y2`	Y = (destination port numbers *(see page 144)*)	All frames with a destination port number in the mentioned range are allowed.
`Firewall Eth1 Reject port_type ports Y1 to Y2`	Y = (destination port numbers *(see page 144)*)	All frames with a destination port number in the mentioned range are rejected.
`Firewall Eth1 Allow IP •.•.•.• on port_type port Y`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from the mentioned IP address and with the mentioned destination port number are allowed.
`Firewall Eth1 Reject IP •.•.•.• on port_type port Y`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from the mentioned IP address and with the mentioned destination port number are rejected.
`Firewall Eth1 Allow IP •.•.•.• on port_type ports Y1 to Y2`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from the mentioned IP address and with a destination port number in the mentioned range are allowed.

Command	Range	Description
`Firewall Eth1 Reject IP •.•.•.• on port_type ports Y1 to Y2`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from the mentioned IP address and with a destination port number in the mentioned range are rejected.
`Firewall Eth1 Allow IPs •1.•1.•1.•1 to •2.•2.•2.•2 on port_type port Y`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from an IP address in the mentioned range and with the mentioned destination port number are rejected.
`Firewall Eth1 Reject IPs •1.•1.•1.•1 to •2.•2.•2.•2 on port_type port Y`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from an IP address in the mentioned range and with the mentioned destination port number are rejected.
`Firewall Eth1 Allow IPs •1.•1.•1.•1 to •2.•2.•2.•2 on port_type ports Y1 to Y2`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from an IP address in the mentioned range and with a destination port number in the mentioned range are allowed.
`Firewall Eth1 Reject IPs •1.•1.•1.•1 to •2.•2.•2.•2 on port_type ports Y1 to Y2`	• = 0...255 Y = (destination port numbers *(see page 144)*)	All frames from an IP address in the mentioned range and with a destination port number in the mentioned range are rejected.
Firewall Eth1 Allow MAC ••:••:••:••:••:••	• = 0...F	All frames from the mentioned MAC address ••:••:••:••:•• are allowed.
Firewall Eth1 Reject MAC ••:••:••:••:••:••	• = 0...F	All frames with the mentioned MAC address ••:••:••:••:•• are rejected.

Script File Example

```
; Enable firewall on Ethernet 1. All frames are rejected;
FireWall Eth1 Enable;
; Block all Modbus Requests on all IP address
Firewall Eth1 Reject tcp port 502;
; Allow FTP active connection for IP address 85.16.0.17
Firewall Eth1 Allow IP 85.16.0.17 on tcp port 20 to 21;
```

Used Ports List

Protocol	Destination Port Numbers
SoMachine	UDP 1740, 1741, 1742, 1743 TCP 1105
FTP	TCP 21, 20
HTTP	TCP 80
Modbus	TCP 502
Discovery	UDP 27126, 27127
SNMP	UDP 161, 162
NVL	UDP Default value: 1202
Ethernet/IP	UDP 2222 TCP 44818

Section 13.3
Ethernet Optional Devices

What Is in This Section?

This section contains the following topics:

Ethernet Manager

Adding an Ethernet Manager

The controller supports the following Ethernet managers:

- EthernetIP (for CIP Device)
- ModbusTCP Slave Device

To add an Ethernet manager to your controller, select in the **Hardware Catalog**:

- For an EthernetIP: **EthernetIP**
- For a ModbusTCP: **ModbusTCP Slave Device**

Drag it to the **Devices tree** and drop it on one of the highlighted nodes.

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)*

EtherNet/IP Device

Introduction

This section describes the configuration of the EtherNet/IP Device (CIP) to the controller.

For further information about EtherNet/IP (CIP), refer to the www.odva.org website.

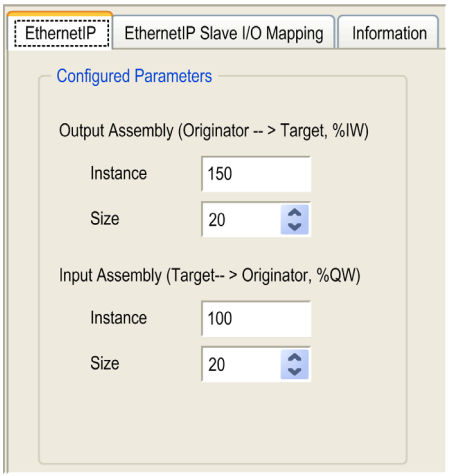
Adding an EtherNet/IP Device

See Adding an EtherNet Manager *(see page 146)*.

EtherNet/IP Device Configuration

To configure the EtherNet/IP device parameters, double-click **Ethernet** → **EthernetIP** in the **Devices tree**.

The following dialog box is displayed:



The EtherNet/IP configuration parameters are defined as:

- **Instance**:
  Number referencing the input or output Assembly.
- **Size**:
  Number of channels of an input or output Assembly.
  The memory size of each channel is 2 bytes that stores the value of an %IWx or %QWx object, where x is the channel number.

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

Element		Admissible Controller Range	SoMachine Default Value
Output Assembly	**Instance**	150...189	150
	Size	2...40	20
Input Assembly	**Instance**	100...149	100
	Size	2...40	20

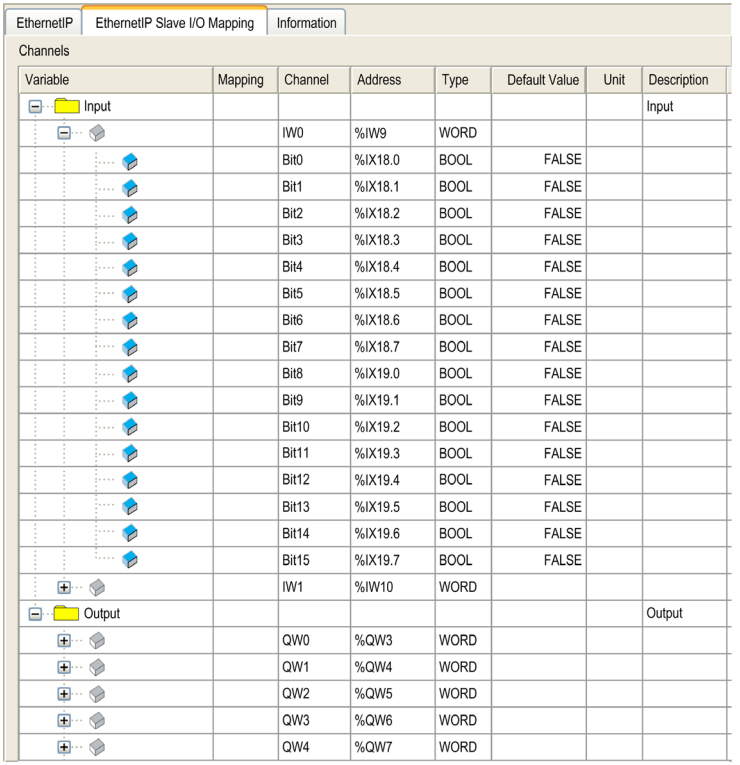
EDS File Generation

The EDS file is generated automatically in the "/usr/Eip" directory within the controller when an application is downloaded, or at start-up when a boot application exists, according to the parameters above.

NOTE: The EDS file is generated when the control network is working correctly on the controller (cable connected and the IP address is acquired).

EthernetIP Slave I/O Mapping Tab

Variables can be defined and named in the **EthernetIP Slave I/O Mapping** tab. Additional information such as topological addressing is also provided in this tab.



Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
Input							Input
		IW0	%IW9	WORD			
		Bit0	%IX18.0	BOOL	FALSE		
		Bit1	%IX18.1	BOOL	FALSE		
		Bit2	%IX18.2	BOOL	FALSE		
		Bit3	%IX18.3	BOOL	FALSE		
		Bit4	%IX18.4	BOOL	FALSE		
		Bit5	%IX18.5	BOOL	FALSE		
		Bit6	%IX18.6	BOOL	FALSE		
		Bit7	%IX18.7	BOOL	FALSE		
		Bit8	%IX19.0	BOOL	FALSE		
		Bit9	%IX19.1	BOOL	FALSE		
		Bit10	%IX19.2	BOOL	FALSE		
		Bit11	%IX19.3	BOOL	FALSE		
		Bit12	%IX19.4	BOOL	FALSE		
		Bit13	%IX19.5	BOOL	FALSE		
		Bit14	%IX19.6	BOOL	FALSE		
		Bit15	%IX19.7	BOOL	FALSE		
		IW1	%IW10	WORD			
Output							Output
		QW0	%QW3	WORD			
		QW1	%QW4	WORD			
		QW2	%QW5	WORD			
		QW3	%QW6	WORD			
		QW4	%QW7	WORD			

The table below describes the **EthernetIP Slave I/O Mapping** configuration:

Channel		Type	Default Value	Description
Input	IW0	WORD	-	Command word of controller outputs (%QW)
	IWxxx			
Output	QW0	WORD	-	State of controller inputs (%IW)
	QWxxx			

The number of word depends on the size parameter configured in EtherNet/IP Device Configuration *(see page 147)*.

Output means OUTPUT from Originator controller (= %IW for the controller).

Input means INPUT from Originator controller (= %QW for the controller).

Connections on EtherNet/IP

To access a target device, open a connection (global name used by EtherNet/IP protocol level) which can include several sessions that send requests.

One explicit connection uses one session (a session is a TCP or UDP connection.

One I/O connection uses 2 sessions.

The following table shows the EtherNet/IP connections limitations:

Characteristic	Maximum
Explicit connections	8 (Class 3)
I/O connections	1 (Class 1)
Connections	8
Sessions	16
Simultaneous requests	32

Profile

The controller supports the following objects:

Object class	Class ID	Cat.	Number of Instances	Effect on Interface Behavior
Identity Object *(see page 151)*	01 hex	1	1	Supports the reset service
Message Router Object *(see page 154)*	02 hex	1	1	Explicit message connection
Assembly Object *(see page 157)*	04 hex	2	2	Defines I/O data format
Connection Manager Object *(see page 159)*	06 hex		1	-
File Object *(see page 162)*	37 hex		2	Allows to exchange EDS file
Modbus Object *(see page 164)*	44 hex		1	-
TCP/IP Interface Object *(see page 165)*	F5 hex	1	1	TCP/IP configuration
Ethernet Link Object *(see page 167)*	F6 hex	1	1	Counter and status information

Identity Object (Class ID = 01 hex)

The following table describes the class attributes of the Identity Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	01h	Implementation revision of the Identity Object
2	Get	Max Instances	UINT	01h	The largest instance number
3	Get	Number of Instances	UINT	01h	The number of object instances
4	Get	Optional Instance Attribute List	UINT, UINT [ ]	00h	The first 2 bytes contain the number of optional instance attributes. Each following pair of bytes represents the number of other optional instance attributes.
6	Get	Max Class Attribute	UINT	07h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	07h	The largest instance attributes value

The following table describes the Class Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
0E	Get Attribute Single	Returns the value of the specified attribute

The following table describes the Instance Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
05	Reset (1)	Initializes EtherNet/IP component (controller reboot)
0E	Get Attribute Single	Returns the value of the specified attribute

(1) Reset Service description:

When the Identity Object receives a Reset request, it:
- determines if it can provide the type of reset requested
- responds to the request
- attempts to perform the type of reset requested

The Reset common service has one specific parameter, Type of Reset (USINT), with the following values:

Value	Type of Reset
0	Emulates as closely as possible cycling power. Simulates Reboot command. **NOTE:** This value is the default value if this parameter is omitted.
1	Emulates as closely as possible the removal and reapplication of supply power to the controller and a restoration of the I/O to initialization values.
2	Returns as closely as possible to the out-of-box configuration, with the exception of communication link parameters, and emulates cycling power as closely as possible. The communication link parameters that are to be preserved are defined by each network type. See the Reset service of the network specific link object(s) for complete information. Simulates Reset origin command.
3...99	Reserved
100...199	Vendor specific
200...255	Reserved

The following table describes the Instance attributes:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Vendor ID	UINT	243h	Schneider Automation ID
2	Get	Device type	UINT	0Eh	PLC
3	Get	Product code	UINT	–	Controller product code
4	Get	Revision	Struct of USINT, USINT	-	Product revision of the controller (1) Equivalent to the 2 low bytes of controller version
5	Get	Status	WORD (1)	-	See definition in the table below
6	Get	Serial number	UDINT	-	Serial number of the controller XX + 3 LSB of MAC address
7	Get	Product name	Struct of USINT, STRING	-	–

(1) Mapped in a WORD:

- MSB: minor revision (second USINT)
- LSB: major revision (first USINT)

Example: 0205h means revision V5.2.

Status Description (Attribute 5):

Bit	Name	Description
0	Owned	Unused
1	Reserved	-
2	Configured	TRUE indicates the device application has been reconfigured.
3	Reserved	-
4...7	Extended Device Status	● 0: self-testing or undetermined ● 1: firmware update in progress ● 2: at least one invalid I/O connection error detected ● 3: no I/O connections established ● 4: non-volatile configuration invalid ● 5: non recoverable error detected ● 6: at least one I/O connection in RUNNING state ● 7: at least one I/O connection established, all in idle mode ● 8: reserved ● 9...15: unused
8	Minor Recoverable Error	TRUE indicates the device detected an error, which, under most circumstances, is recoverable. This type of event does not lead to a change in the device state.
9	Minor Unrecoverable Error	TRUE indicates the device detected an error, which, under most circumstances, is not recoverable. This type of event does not lead to a change in the device state.
10	Major Recoverable Error	TRUE indicates the device detected an error, which requires the device to report an exception and enter into the HALT state. This type of event leads to a change in the device state, but, under most circumstances, is recoverable.
11	Major Unrecoverable Error	TRUE indicates the device detected an error, which requires the device to report an exception and enter into the HALT state. This type of event leads to a change in the device state, but, under most circumstances, is not recoverable.
12...15	Reserved	-

Message Router Object (Class ID = 02 hex)

The following table describes the class attributes of the Message Router Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	01h	Implementation revision of the Message Router Object
2	Get	Max Instances	UINT	01h	The largest instance number
3	Get	Number of Instance	UINT	01h	The number of object instances
4	Get	Optional Instance Attribute List	Struct of UINT, UINT [ ]	20	The first 2 bytes contain the number of optional instance attributes. Each following pair of bytes represents the number of other optional instance attributes (from 100 to 119).
5	Get	Optional Service List	UINT	00h	The number and list of any implemented optional services attribute (0: no optional services implemented)
6	Get	Max Class Attribute	UINT	07h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	119	The largest instance attributes value

The following table describes the Class Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
0E	Get Attribute Single	Returns the value of the specified attribute

The following table describes the Instance Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
0E	Get Attribute Single	Returns the value of the specified attribute

The following table describes the Instance attributes:

Attribute ID	Access	Name	Data Type	Value	Description
1	Get	Implemented Object List	Struct of UINT, UINT [ ]	-	Implemented Object list. The first 2 bytes contain the number of implemented objects. Each two bytes that follow represent another implemented class number. This list contains the following objects: ● Identity ● Message Router ● Assembly ● Connection Manager ● Parameter ● File Object ● Modbus ● Port ● TCP/IP ● Ethernet Link
2	Get	Number available	UINT	20h	Maximum number of concurrent CIP (Class1 or Class 3) connections supported
100	Get	Total incoming Class1 packets received during the last second	UINT	-	Total number of incoming packets received for all implicit (Class1) connections during the last second
101	Get	Total outgoing Class1 packets sent during the last second	UINT	-	Total number of outgoing packets sent for all implicit (Class1) connections during the last second
102	Get	Total incoming Class3 packets received during the last second	UINT	-	Total number of incoming packets received for all explicit (Class 3) connections during the last second
103	Get	Total outgoing Class3 packets sent during the last second	UDINT	-	Total number of outgoing packets sent for all explicit (Class 3) connections during the last second
104	Get	Total incoming unconnected packets received during the last second	UINT	-	Total number of incoming unconnected packets received during the last second

Attribute ID	Access	Name	Data Type	Value	Description
105	Get	Total outgoing unconnected packets sent during the last second	UINT	-	Total number of outgoing unconnected packets sent during the last second
106	Get	Total incoming EtherNet/IP packets received during the last second	UINT	-	Total unconnected Class1 or Class 3 packets received during the last second
107	Get	Total outgoing EtherNet/IP packets sent during the last second	UINT	-	Total unconnected Class1 or Class 3 packets sent during the last second
108	Get	Total incoming Class1 packets received	UINT	-	Total number of incoming packets received for all implicit (Class1) connections
109	Get	Total outgoing Class1 packets sent	UINT	-	Total number of outgoing packets sent for all implicit (Class1) connections
110	Get	Total incoming Class3 packets received	UINT	-	Total number of incoming packets received for all explicit (Class 3) connections. This number includes the packets that would be returned if an error had been detected (listed in the next two rows).
111	Get	Total incoming Class3 packets Invalid Parameter Value	UINT	-	Total number of incoming Class 3 packets that targeted a non-supported service/class/instance/attribute/member
112	Get	Total incoming Class3 packets Invalid Format	UINT	-	Total number of incoming Class 3 packets that had an invalid format
113	Get	Total outgoing Class3 packets sent	UINT	-	Total number of packets sent for all explicit (Class 3) connections
114	Get	Total incoming unconnected packets received	UINT	-	Total number of incoming unconnected packets. This number includes the packets that would be returned if an error had been detected (listed in the next two rows).

Attribute ID	Access	Name	Data Type	Value	Description
115	Get	Total incoming unconnected packets Invalid Parameter Value	UINT	-	Total number of incoming unconnected packets that targeted a non-supported service/class/instance/attribute/member
116	Get	Total incoming unconnected packets Invalid Format	UINT	-	Total number of incoming unconnected packets that had an invalid format
117	Get	Total outgoing unconnected packets sent	UINT	-	Total number of all unconnected packets sent
118	Get	Total incoming EtherNet/IP packets	UINT	-	Total unconnected, Class 1, or Class 3 packets received
119	Get	Total outgoing EtherNet/IP packets	UINT	-	Total unconnected, Class 1, or Class 3 packets sent

Assembly Object (Class ID = 04 hex)

The following table describes the class attributes of the Assembly Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	2	Implementation revision of the Assembly Object
2	Get	Max Instances	UINT	189	The largest instance number
3	Get	Number of Instances	UINT	2	The number of object instances
4	Get	Optional Instance Attribute List	Struct of: UINT UINT [ ]	 1 4	The first 2 bytes contain the number of optional instance attributes. Each following pair of bytes represents the number of other optional instance attributes.
5	Get	Optional Service List	UINT	00h	The number and list of any implemented optional services attribute (0: no optional services implemented)
6	Get	Max Class Attribute	UINT	07h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	04h	The largest instance attributes value

The following table describes the Class Services:

Service Code (hex)	Name	Description
0E	Get Attribute Single	Returns the value of the specified attribute

The following table describes the Instance Services:

Service Code (hex)	Name	Description
0E	Get Attribute Single	Returns the value of the specified attribute
10	Set Attribute Single	Modifies the value of the specified attribute
18	Get Member	Reads a member of an Assembly object instance
19	Set Member	Modifies a member of an Assembly object instance

Instances Supported

Output means OUTPUT from Originator controller (= %IW for the controller).

Input means INPUT from Originator controller (= %QW for the controller).

The controller supports 2 Assemblies:

Name	Instance	Data Size
Controller Output (%IW)	Configurable: must be between 100 and 149	2...40 words
Controller Input (%QW)	Configurable: must be between 150 and 189	2...40 words

NOTE: The Assembly object binds together the attributes of multiple objects so that information to or from each object can be communicated over a single connection. Assembly objects are static. The Assemblies in use can be modified through the parameter access of the network configuration tool (RSNetWorx). The controller needs to recycle power to register a new Assembly assignment.

The following table describes the Instance attributes:

Attribute ID	Access	Name	Data Type	Value	Description
1	Get	Number of Member Object List	UINT	2...40	Always 1 member for the controller
2	Get	Member List	ARRAY of Struct	-	Array of 1 structure where each structure represents one member
3	Get/Set	Instance Data	ARRAY of Byte	-	Data Set service only available for Controller output
4	Get	Instance Data Size	UINT	4...80	Size of data in byte

Member list content:

Name	Data Type	Value	Type of Reset
Member data size	UINT	4...40	Member data size in bits
Member path size	UINT	6	Size of the EPATH (see table below)
Member path	EPATH	-	EPATH to the Member

EPATH is:

Word	Value	Semantic
0	2004 hex	Class 4
1	24xx hex	Instance xx where xx is the instance value (example: 2464 hex = instance 100).
2	30 hex	Attribute 3

Connection Manager Object (Class ID = 06 hex)

The following table describes the class attributes of the Assembly Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	2	Implementation revision of the Connection Manager Object
2	Get	Max Instances	UINT	189	The largest instance number
3	Get	Number of Instances	UINT	2	The number of object instances

Attribute ID	Access	Name	Data Type	Value	Details
4	Get	Optional Instance Attribute List	Struct of: UINT UINT [ ]	-	The number and list of the optional attributes. The first word contains the number of attributes to follow and each following word contains another attribute code. Following optional attributes include: ● total number of incoming connection open requests ● the number of requests rejected because of the non-conforming format of the Forward Open ● the number of requests rejected because of insufficient resources ● the number of requests rejected because of the parameter value sent with the Forward Open ● the number of Forward Close requests received ● the number of Forward Close requests that had an invalid format ● the number of Forward Close requests that could not be matched to an active connection ● the number of connections that have timed out because the other side stopped producing, or a network disconnection occurred
6	Get	Max Class Attribute	UINT	07h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	08h	The largest instance attributes value

The following table describes the Class Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
0E	Get Attribute Single	Returns the value of the specified attribute

The following table describes the Instance Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all instance attributes
0E	Get Attribute Single	Returns the value of the specified attribute
4E	Forward Close	Closes an existing connection
52	Unconnected Send	Sends a multi-hop unconnected request
54	Forward Open	Opens a new connection

The following table describes the Instance attributes:

Attribute ID	Access	Name	Data Type	Value	Description
1	Get	Open Requests	UINT	-	Number of Forward Open service requests received
2	Get	Open Format Rejects	UINT	-	Number of Forward Open service requests which were rejected due to invalid format
3	Get	Open Resource Rejects	ARRAY of Byte	-	Number of Forward Open service requests which were rejected due to lack of resources
4	Get	Open Other Rejects	UINT	-	Number of Forward Open service requests which were rejected for reasons other than invalid format or lack of resources
5	Get	Close Requests	UINT	-	Number of Forward Close service requests received
6	Get	Close Format Requests	UINT	-	Number of Forward Close service requests which were rejected due to invalid format
7	Get	Close Other Requests	UINT	-	Number of Forward Close service requests which were rejected for reasons other than invalid format
8	Get	Connection Timeouts	UINT	-	Total number of connection timeouts that have occurred in connections controlled by this Connection Manager

File Object (Class ID = 37 hex)

The following table describes the class attributes of the File Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	1	Implementation revision of the File Object
2	Get	Max Instances	UINT	C9h	The largest instance number
3	Get	Number of Instances	UINT	2	The number of object instances
6	Get	Max Class Attribute	UINT	20h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	0Bh	The largest instance attributes value
32	Get	Instance List	-	-	Returns information on all configured instances including Instance Number, Instance Name and Instance File Name

The following table describes the Class Services:

Service Code (hex)	Name	Description
0E	Get Attribute Single	Returns the value of the specified attribute

Instance Code

The following table describes the Instance Services:

Service Code (hex)	Name	Description
0E	Get Attribute Single	Returns the value of the specified instance attribute
4B	Initiate Upload	Start uploading process. Request contains the Maximum File Size the Client can accept on Upload. Response contains the actual File Size, which will never be more than the Maximum File Size and the Transfer Size, which is the number of bytes transferred with each Upload Transfer request.
4F	Upload Transfer	Upload another section of file data. Request contains the Transfer Number, which is incremented with each subsequent transfer. Response contains the matching Transfer Number, Transfer Type, File Data, and for the last transfer, the Checksum word. Transfer Type indicates if this is the first, intermediate or last packet, if it is the only packet, or if the transfer should be aborted.

The following table describes the Instance Attributes:

Attribute ID	Access	Name	Data Type	Value	Description
1	Get	State	USINT	-	One of the following: ● 0: non existent ● 1: file empty - file should not have any content until it is downloaded from the remote client. When set, File name, Revision, Checksum and File Save Flag values have no meaning, and File Size is 0. ● 2: file loaded - file contents are pre-loaded by the application (file size > 0) or file data has been downloaded and stored into the non-volatile memory area ● 3: upload initiated ● 4: download initiated ● 5: upload in progress ● 6: download in progress ● 7: storing to non-volatile area is in progress
2	Get	Instance Name	STRING	-	Unique Name assigned to the File Object Instance. For the C8 hex instance, it is "EDS and Icon Files". For the C9 hex instance, it is "Related EDS and Icon Files".
3	Get	Instance Format Revision	UINT	-	Revision number assigned for this instance by the application, to differentiate between different file formats.
4	Get	File Name	STRING	-	Unique name for File Storage
5	Get	File Revision	USINT	Major Minor	File Revision is updated every time file content is changed.
6	Get	File Size	UDINT	-	File Size in bytes
7	Get	File Checksum	UINT	-	Two’s complement of the 16 bit sum of all bytes
8	Get	Invocation Method	USINT	-	Defines what should happen after the file is downloaded. Possible options include: ● 0: No Action ● 2: Power Cycle, etc.
9	Get	File Save Parameters	BYTE	-	If bit 1 is set, the file should be explicitly saved to non-volatile storage after download is complete.
10	Get	File Type	USINT	-	● 0: Read/Write access ● 1: Read Only access

Attribute ID	Access	Name	Data Type	Value	Description
11	Get	File Encoding Format	UINT	-	• 0: no encoding • 1: encoded using ZLIB

Modbus Object (Class ID = 44 hex)

The Modbus object provides an additional method to access the Modbus table data. A single explicit request will either read or write 1 or more contiguous registers. An additional Pass-through service allows the user to specify an actual Modbus message data.

The following table describes the class attributes of the Modbus Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	1	Implementation revision of the Modbus Object

The following table describes the Class Services:

Service Code (hex)	Name	Description
0E	Get Attribute Single	Returns the value of the specified attribute

Instance Codes

Only instance 1 is supported.

The following table describes the Instance Services:

Service Code (hex)	Name	Description
4B	Read Digital Inputs	Returns the value of one or several contiguous Digital Input registers
4C	Read Coils	Returns the value of one or several contiguous Coils
4E	Read Holding Registers	Returns the value of one or several contiguous Holding Registers
4F	Write Coils	Updates the value of one or several contiguous Coils
50	Write Holding Registers	Updates the value of one or several contiguous Holding Registers

NOTE: The Read Register service requires 4 bytes of data: the first word contains the starting register address and the second word contains the number of registers to read. The Write service request requires the same 4 bytes, followed by the actual data.
The Modbus Pass-through service indicates a specific Modbus function. The translation function will not perform any Indian conversion on the request or response data. Both request and response contain 1 byte of the Modbus Function code followed by the Modbus message data, including a sub-function code if present.

TCP/IP Interface Object (Class ID = F5 hex)

This object maintains link specific counters and status information for an Ethernet 802.3 communications interface.

The following table describes the class attributes of the TCP/IP Interface Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	1	Implementation revision of the TCP/IP Interface Object
2	Get	Max Instances	UINT	1	The largest instance number
3	Get	Number of Instance	UINT	1	The number of object instances
6	Get	Max Class Attribute	UINT	07h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	06h	The largest instance attributes value

The following table describes the Class Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
0E	Get Attribute Single	Returns the value of the specified attribute

Instance Codes

Only instance 1 is supported.

The following table describes the Instance Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all instance attributes
0E	Get Attribute Single	Returns the value of the specified instance attribute

The following table describes the Instance Attributes:

Attribute ID	Access	Name	Data Type	Value	Description
1	Get	Status	DWORD	Bit level	• 0: The interface configuration attribute has not been configured. • 1: The interface configuration contains a valid configuration. • 2...15: Reserved for future use.

Attribute ID	Access	Name	Data Type	Value	Description
2	Get	Configuration Capability	DWORD	Bit level	● 0: BOOTP Client ● 1: DNS Client ● 2: DHCP Client ● 3: DHCP-DNS capable ● 4: interface configuration set table All other bits are reserved and set to 0.
3	Get	Configuration	DWORD	Bit level	● 0: The interface configuration is valid. ● 1: The interface configuration is obtained with BOOTP. ● 2: The interface configuration is obtained with DHCP. ● 3: reserved ● 4: DNS Enable All other bits are reserved and set to 0.
4	Get	Physical Link	UINT	Path size	Number of 16 bits word in the element Path
			Padded EPATH	Path	Logical segments identifying the physical link object. The path is restricted to one logical class segment and one logical instance segment. The maximum size is 12 bytes.
5	Get	Interface configuration	UDINT	IP Address	-
			UDINT	Network Mask	-
			UDINT	Gateway Address	-
			UDINT	Primary Name	-
			UDINT	Secondary Name	0: no secondary name server address has been configured. Otherwise, the name server address shall be set to a valid Class A, B, or C address.
			STRING	Default Domain Name	ASCII characters. Maximum length is 48 characters. Padded to an even number of characters (pad not included in length). 0: no Domain Name is configured
6	Get	Host Name	STRING	-	ASCII characters. Maximum length is 64 characters. Shall be padded to an even number of characters (pad not included in length). 0: no Host Name is configured

Ethernet Link Object (Class ID = F6 hex)

This object provides the mechanism to configure a TCP/IP network interface device.

The following table describes the class attributes of the Ethernet Link Object:

Attribute ID	Access	Name	Data Type	Value	Details
1	Get	Revision	UINT	2	Implementation revision of the Ethernet Link Object
2	Get	Max Instances	UINT	1	The largest instance number
3	Get	Number of Instances	UINT	1	The number of object instances
6	Get	Max Class Attribute	UINT	07h	The largest class attributes value
7	Get	Max Instance Attribute	UINT	03h	The largest instance attribute value

The following table describes the Class Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all class attributes
0E	Get Attribute Single	Returns the value of the specified attribute

Instance Codes

Only instance 1 is supported.

The following table describes the Instance Services:

Service Code (hex)	Name	Description
01	Get Attribute All	Returns the value of all instance attributes
10	Set Attribute Single	Modifies the value of the specified attribute
0E	Get Attribute Single	Returns the value of the specified instance attribute

The following table describes the Instance Attributes:

Attribute ID	Access	Name	Data Type	Value	Description
1	Get	Interface Speed	UDINT	-	Speed in Mbps (10 or 100)
2	Get	Interface Flags	DWORD	Bit level	• 0: link status • 1: half/full duplex • 2...4: negotiation status • 5: manual setting / requires reset • 6: local hardware error detected All other bits are reserved and set to 0.
3	Get	Physical Address	ARRAY of 6 USINT	-	This array contains the MAC address of the product. Format: XX-XX-XX-XX-XX-XX

Modbus TCP Slave Device

Overview

This section describes the configuration of Modbus TCP slave device to the controller.

The Modbus TCP slave device creates a specific I/O area in the controller, accessible through the Modbus TCP protocol. It is used when an external I/O scanner (master) needs to access the `%IW` and `%QW` objects of the controller. The main advantage of using a Modbus TCP slave device is that the controller objects are gathered, and can be accessed through a single Modbus request.

The Modbus slave device adds another Modbus server function to the controller. This server is accessible by the Modbus client application by using the configured Unit ID (not 255). The regular Modbus server of the slave controller needs no configuration, and is addressed through the Unit ID = 255.

Inputs/outputs are seen from the slave controller: inputs are written by the master, and outputs are read by the master.

The TCP slave device also has the capability to define a privileged Modbus client application, whose connection is never forcefully closed (regular Modbus connections may be closed when more than 8 connections are needed).

The timeout duration associated to the privileged connection allows you to verify whether the controller is being polled by the privileged master. If no Modbus request is received within the timeout duration, the diagnostic information `i_byMasterIpLost` is set to 1 (TRUE). For more information, refer to the Ethernet Port Read-Only System Variables *(see Modicon M241 Logic Controller, System Functions and Variables, PLCSystem Library Guide)*.

For further information about Modbus TCP, refer to the www.modbus.org website.

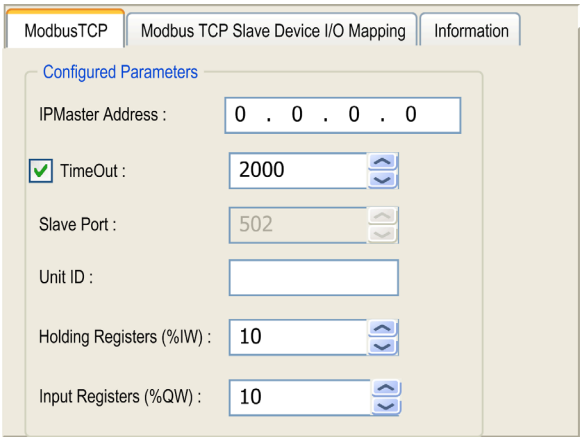
Adding a Modbus TCP Slave Device

See Adding an Ethernet Manager *(see page 146)*.

Modbus TCP Configuration

To configure the Modbus TCP slave device, double-click **Modbus → Modbus TCP** in the **Devices tree**.

This dialog box appears:



Element	Description
IP Master Address	IP address of the Modbus master The connections are not closed on this address.
TimeOut	Timeout in 500 ms increments **NOTE:** The timeout applies to the IP master Address unless the address is 0.0.0.0.
Slave Port	Modbus communication port (502)
Unit ID	Sends the requests to the Modbus TCP slave device (1...247), instead of sending requests to the regular Modbus server (255).
Holding Registers (%IW)	Number of registers in the input registers (2...40) (each register is 2 bytes)
Input Registers (%QW)	Number of registers in the output registers (2...40) (each register is 2 bytes)

Modbus TCP Slave Device I/O Mapping Tab

The I/Os are mapped to Modbus registers from master point of view as follows:

- %IWs are mapped from register 0 to n-1 and are R/W (n = Holding register quantity, each holding register is 2 bytes).
- %QWs are mapped from register n to n+m -1 and are read only (m = Input registers quantity, each input register is 2 bytes).

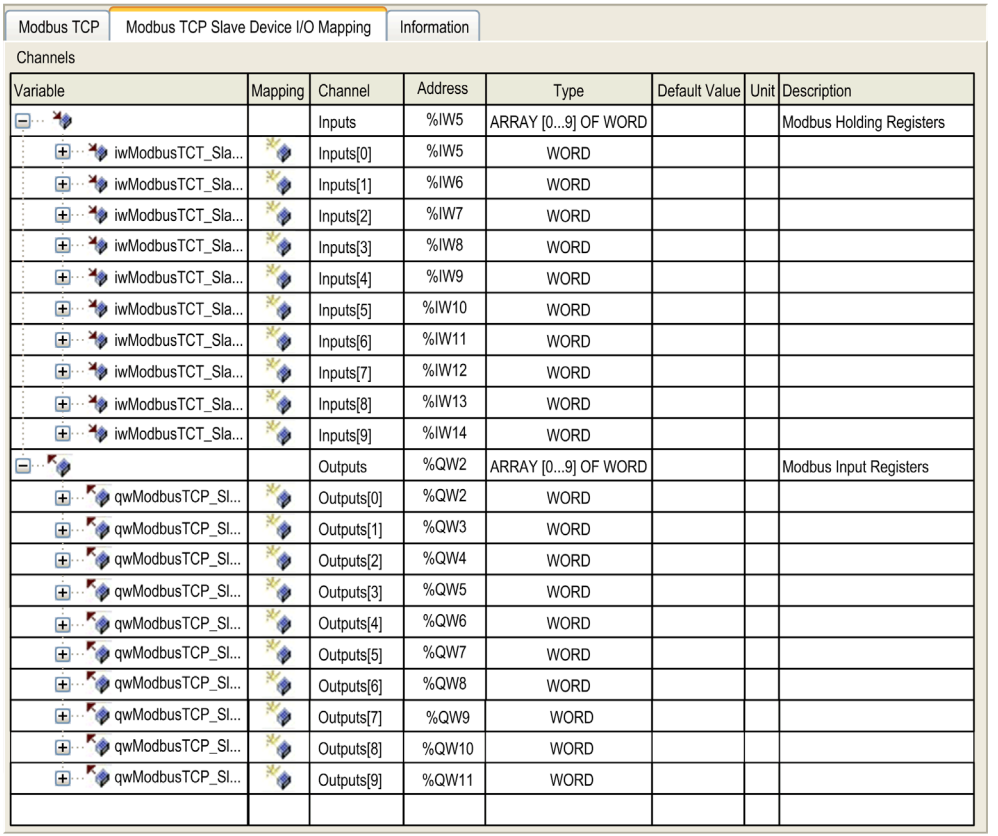
Once a Modbus TCP slave device has been configured, Modbus commands sent to its Unit ID (Modbus address) access the `%IW` and `%QW` objects of the controller instead of the regular Modbus words (accessed when the Unit ID is 255). This facilitates read/write operations by a Modbus TCP IOScanner application.

The Modbus TCP slave device responds to a subset of the Modbus commands with the purpose of exchanging data with the external I/O scanner. The following Modbus commands are supported by the Modbus TCP slave device:

Function Code Dec (Hex)	Function	Comment
3 (3)	Read holding register	Allows the master to read %IW and %QW of the device
6 (6)	Write single register	Allows the master to write %IW of the device
16 (10)	Write multiple registers	Allows the master to write %IW of the device
23 (17)	Read/write multiple registers	Allows the master to read %IW and %QW of the device and write %IW of the device
Other	Not supported	–

NOTE: Modbus requests that attempt to access registers above n+m-1 are answered by the 02 - ILLEGAL DATA ADDRESS exception code.

To link I/O to variables, select the **Modbus TCP Slave Device I/O Mapping** tab:



Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
		Inputs	%IW5	ARRAY [0...9] OF WORD			Modbus Holding Registers
iwModbusTCT_Sla...		Inputs[0]	%IW5	WORD			
iwModbusTCT_Sla...		Inputs[1]	%IW6	WORD			
iwModbusTCT_Sla...		Inputs[2]	%IW7	WORD			
iwModbusTCT_Sla...		Inputs[3]	%IW8	WORD			
iwModbusTCT_Sla...		Inputs[4]	%IW9	WORD			
iwModbusTCT_Sla...		Inputs[5]	%IW10	WORD			
iwModbusTCT_Sla...		Inputs[6]	%IW11	WORD			
iwModbusTCT_Sla...		Inputs[7]	%IW12	WORD			
iwModbusTCT_Sla...		Inputs[8]	%IW13	WORD			
iwModbusTCT_Sla...		Inputs[9]	%IW14	WORD			
		Outputs	%QW2	ARRAY [0...9] OF WORD			Modbus Input Registers
qwModbusTCP_Sl...		Outputs[0]	%QW2	WORD			
qwModbusTCP_Sl...		Outputs[1]	%QW3	WORD			
qwModbusTCP_Sl...		Outputs[2]	%QW4	WORD			
qwModbusTCP_Sl...		Outputs[3]	%QW5	WORD			
qwModbusTCP_Sl...		Outputs[4]	%QW6	WORD			
qwModbusTCP_Sl...		Outputs[5]	%QW7	WORD			
qwModbusTCP_Sl...		Outputs[6]	%QW8	WORD			
qwModbusTCP_Sl...		Outputs[7]	%QW9	WORD			
qwModbusTCP_Sl...		Outputs[8]	%QW10	WORD			
qwModbusTCP_Sl...		Outputs[9]	%QW11	WORD			

Channel		Type	Description
Input	IW0	WORD	Holding register 0
	...	...	...
	IWx	WORD	Holding register x
Output	IW0	WORD	Input register 0
	...	...	...
	IWy	WORD	Input register y

The number of words depends on the **Holding Registers (%IW)** and **Input Registers (%QW)** parameters of the **ModbusTCP** tab.

NOTE: Output means OUTPUT from Originator controller (= %IW for the controller).
Input means INPUT from Originator controller (= %QW for the controller).

Chapter 14
Serial Line Configuration

Introduction

This chapter describes how to configure the serial line communication of the Modicon M241 Logic Controller.

The Modicon M241 Logic Controller has 2 Serial Line ports. These ports are configured to use the following protocols when new or after a controller firmware update:

- Serial Line 1: SoMachine Network Manager.
- Serial Line 2: Modbus Manager.

What Is in This Chapter?

This chapter contains the following topics:

Serial Line Configuration

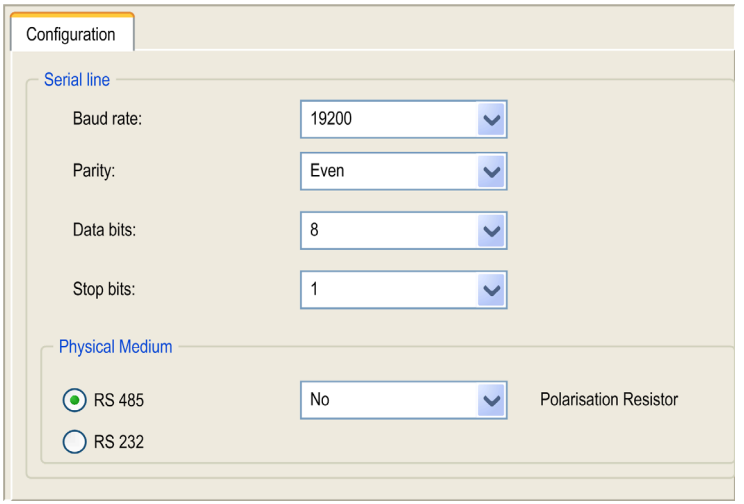
Introduction

The Serial Line configuration window allows you to configure the physical parameters of a serial line (baud rate, parity, and so on).

Serial Line Configuration

To configure a Serial Line, double-click **Serial line** in the **Devices tree**.

The **Configuration** window is displayed as below:



The following parameters must be identical for each serial device connected to the port.

Element	Description
Baud rate	Transmission speed in bits/s
Parity	Used for error detection
Data bits	Number of bits for transmitting data
Stop bits	Number of stop bits
Physical Medium	Specify the medium to use: ● RS485 (using polarisation resistor or not) ● RS232 (only available on Serial Line 1)
Polarization Resistor	Polarization resistors are integrated in the controller. They are switched on or off by this parameter.

The SoMachine protocol is incompatible with other protocols such as Modbus Serial Line. Connecting a new controller or updating the firmware of a controller connected to an active Modbus configured serial line can cause the other devices on the serial line to stop communicating. Ensure that the controller is not connected to an active Modbus serial line network before downloading a valid application having the concerned port or ports properly configured for the intended protocol.

NOTICE

INTERRUPTION OF SERIAL LINE COMMUNICATIONS

Be sure that your application has the serial line ports properly configured for Modbus before physically connecting the controller to an operational Modbus Serial Line network.

Failure to follow these instructions can result in equipment damage.

This table indicates the maximum baud rate value of the managers:

Manager	Maximum Baud Rate (Bits/S)
SoMachine Network Manager	115200
Modbus Manager	
ASCII Manager	
Modbus IOScanner	

SoMachine Network Manager

Introduction

Use the SoMachine Network Manager to exchange variables with a XBTGT/XBTGK Advanced Panel with SoMachine software protocol, or when the Serial Line is used for SoMachine programming.

Adding the Manager

To add a SoMachine Network Manager to your controller, select the **SoMachine-Network Manager** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

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)*

Configuring the Manager

There is no configuration for SoMachine Network Manager.

Adding a Modem

To add a modem to the SoMachine Network Manager, refer to Adding a Modem to a Manager *(see page 191)*.

Modbus Manager

Introduction

The Modbus Manager is used for Modbus RTU or ASCII protocol in master or slave mode.

Adding the Manager

To add a Modbus manager to your controller, select the **Modbus Manager** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

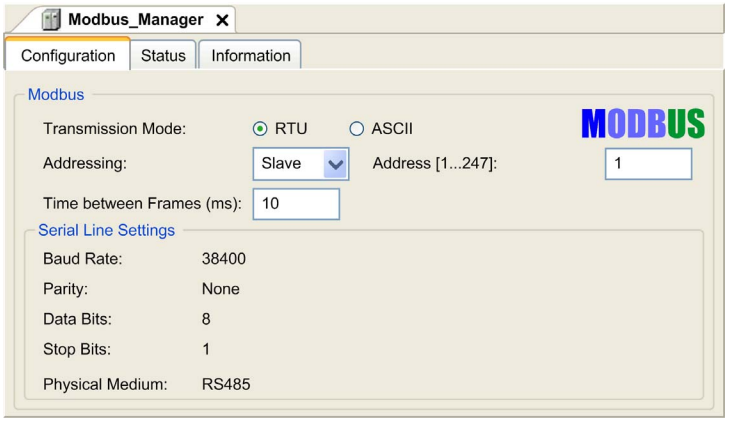
For more information on adding a device to your project, refer to:

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

Modbus Manager Configuration

To configure the Modbus Manager of your controller, double-click **Modbus Manager** in the **Devices tree**.

The Modbus Manager configuration window is displayed as below:



Set the parameters as described in this table:

Element	Description
Transmission Mode	Specify the transmission mode to use: ● RTU: uses binary coding and CRC error-checking (8 data bits) ● ASCII: messages are in ASCII format, LRC error-checking (7 data bits) Set this parameter identical for each Modbus device on the link.
Addressing	Specify the device type: ● Master ● Slave
Address	Modbus address of the device, when slave is selected.
Time between Frames (ms)	Time to avoid bus-collision. Set this parameter identical for each Modbus device on the link.
Serial Line Settings	Parameters specified in the Serial Line configuration window.

Modbus Master

When the controller is configured as a Modbus Master, the following function blocks are supported from the PLCCommunication Library:

- ADDM
- READ_VAR
- SEND_RECV_MSG
- SINGLE_WRITE
- WRITE_READ_VAR
- WRITE_VAR

For further information, see Function Block Descriptions *(see SoMachine, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)* of the PLCCommunication Library.

Modbus Slave

When the controller is configured as Modbus Slave, the following Modbus requests are supported:

Function Code Dec (Hex)	Sub-Function Dec (Hex)	Function
1 (1 hex)	–	Read digital outputs (%Q)
2 (2 hex)	–	Read digital inputs (%I)
3 (3 hex)	–	Read multiple register (%MW)
6 (6 hex)	–	Write single register (%MW)
8 (8 hex)	–	Diagnostic
15 (F hex)	–	Write multiple digital outputs (%Q)
16 (10 hex)	–	Write multiple registers (%MW)

Function Code Dec (Hex)	Sub-Function Dec (Hex)	Function
23 (17 hex)	–	Read/write multiple registers (%MW)
43 (2B hex)	14 (E hex)	Read device identification

This table contains the sub-function codes supported by the diagnostic Modbus request 08:

Sub-Function Code		Function
Dec	**Hex**	
10	0A	Clears Counters and Diagnostic Register
11	0B	Returns Bus Message Count
12	0C	Returns Bus Communication Error Count
13	0D	Returns Bus Exception Error Count
14	0E	Returns Slave Message Count
15	0F	Returns Slave No Response Count
16	10	Returns Slave NAK Count
17	11	Returns Slave Busy Count
18	12	Returns Bus Character Overrun Count

This table lists the objects that can be read with a read device identification request (basic identification level):

Object ID	Object Name	Type	Value
00 hex	Vendor code	ASCII String	Schneider Electric
01 hex	Product code	ASCII String	Controller reference eg: TM241CE24T
02 hex	Major / Minor revision	ASCII String	aa.bb.cc.dd (same as device descriptor)

The following section describes the differences between the Modbus memory mapping of the controller and HMI Modbus mapping. If you do not program your application to recognize these differences in mapping, your controller and HMI will not communicate correctly. Thus it will be possible for incorrect values to be written to memory areas responsible for output operations.

⚠ WARNING

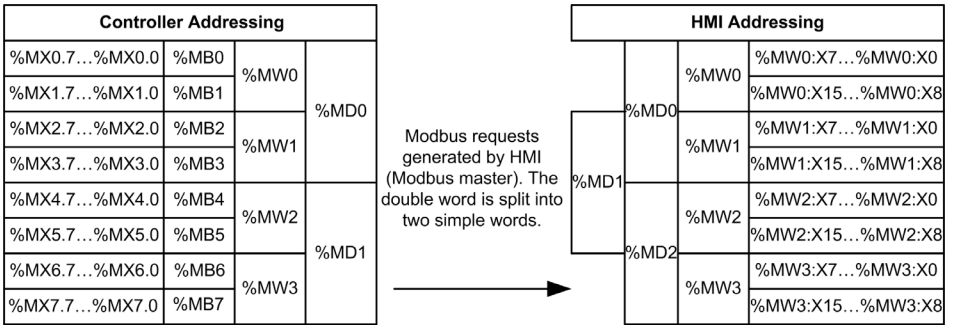
UNINTENDED EQUIPMENT OPERATION

Program your application to translate between the Modbus memory mapping used by the controller and that used by any attached HMI devices.

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

When the controller and the Magelis HMI are connected via Modbus (HMI is master of Modbus requests), the data exchange uses simple word requests.

There is an overlap on simple words of the HMI memory while using double words but not for the controller memory (see following diagram). In order to have a match between the HMI memory area and the controller memory area, the ratio between double words of HMI memory and the double words of controller memory has to be 2.



The following gives examples of memory match for the double words:

- %MD2 memory area of the HMI corresponds to %MD1 memory area of the controller because the same simple words are used by the Modbus request.
- %MD20 memory area of the HMI corresponds to %MD10 memory area of the controller because the same simple words are used by the Modbus request.

The following gives examples of memory match for the bits:

- %MW0:X9 memory area of the HMI corresponds to %MX1.1 memory area of the controller because the simple words are split in 2 distinct bytes in the controller memory.

Adding a Modem

To add a Modem to the Modbus Manager, refer to Adding a Modem to a Manager *(see page 191)*.

ASCII Manager

Introduction

The ASCII manager is used to transmit and/or receive data with a simple device.

Adding the Manager

To add an ASCII manager to your controller, select the **ASCII Manager** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

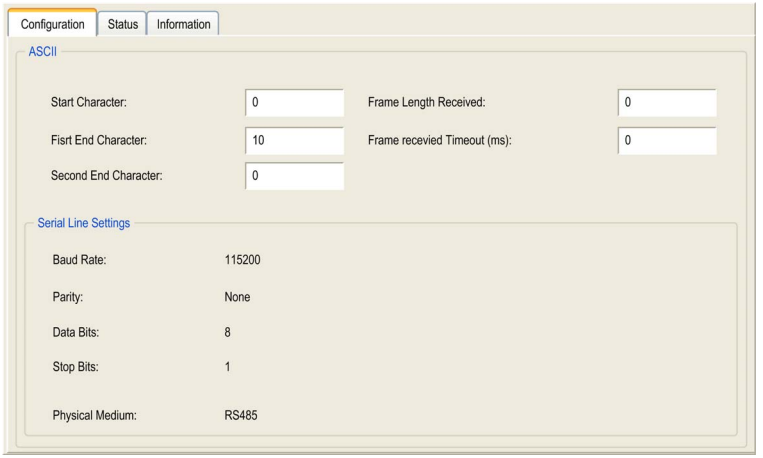
For more information on adding a device to your project, refer to:

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

ASCII Manager Configuration

To configure the ASCII manager of your controller, double-click **ASCII Manager** in the **Devices tree**.

The ASCII Manager configuration window is displayed as below:



Set the parameters as described in this table:

Parameter	Description
Start Character	If 0, no start character is used in the frame. Otherwise, in **Receiving Mode**, the corresponding character in ASCII is used to detect the beginning of a frame. In **Sending Mode**, this character is added at the beginning of the frame.
First End Character	If 0, no first end character is used in the frame. Otherwise, in **Receiving Mode**, the corresponding character in ASCII is used to detect the end of a frame. In **Sending Mode**, this character is added at the end of the frame.
Second End Character	If 0, no second end character is used in the frame. Otherwise, in **Receiving Mode**, the corresponding character in ASCII is used to detect the end of a frame. In **Sending Mode**, this character is added at the end of the frame.
Frame Length Received	If 0, this parameter is not used. This parameter allows the system to conclude an end of frame at reception when the controller received the specified number of characters. **Note:** This parameter cannot be used simultaneously with **Frame Received Timeout (ms)**.
Frame Received Timeout (ms)	If 0, this parameter is not used. This parameter allows the system to conclude the end of frame at reception after a silence of the specified number of ms.
Serial Line Settings	Parameters specified in the Serial Line configuration window *(see page 174)*.

NOTE: In the case of using several frame termination conditions, the first condition to be TRUE will terminate the exchange.

Adding a Modem

To add a Modem to the ASCII manager, refer to Adding a Modem to a Manager *(see page 191)*.

Modbus IOScanner

Introduction

The Modbus IOScanner is used to simplify exchanges with Modbus slave devices.

Add a Modbus IOScanner

To add a Modbus IOScanner on a Serial Line, select the **Modbus_IOScanner** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes.

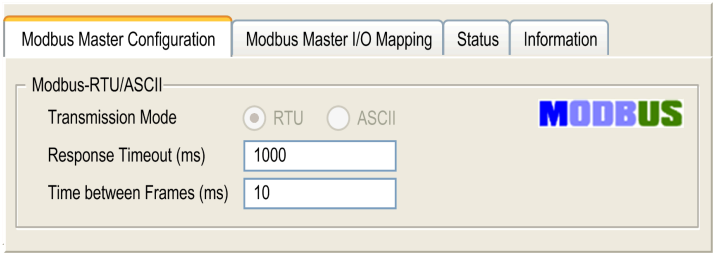
For more information on adding a device to your project, refer to:

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

Modbus IOScanner Configuration

To configure a Modbus IOScanner on a Serial Line, double-click **Modbus IOScanner** in the **Devices tree**.

The configuration window is displayed as below:



Set the parameters as described in this table:

Element	Description
Transmission Mode	Specifies the transmission mode to use: ● RTU: uses binary coding and CRC error-checking (8 data bits) ● ASCII: messages are in ASCII format, LRC error-checking (7 data bits) Set this parameter identical for each Modbus device on the network.
Response Timeout (ms)	Timeout used in the exchanges.
Time between Frames (ms)	Delay to reduce data collision on the bus. Set this parameter identical for each Modbus device on the network.

Adding a Device on the Modbus IOScanner

Introduction

This section describes how to add a device on the Modbus IOScanner.

Add a Device on the Modbus IOScanner

To add a device on the Modbus IOScanner, select the **Generic Modbus Slave** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the **Modbus_IOScanner** node of the **Devices tree**.

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

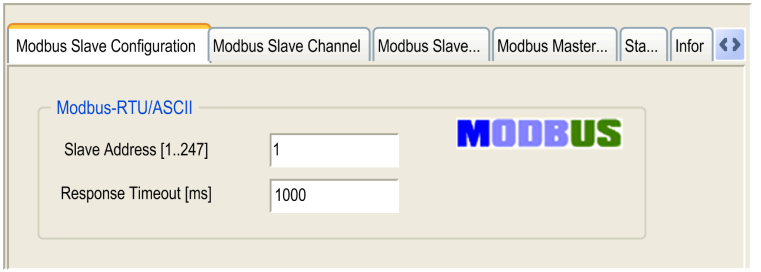
• Using the Drag-and-drop Method *(see SoMachine, Programming Guide)*

• Using the Contextual Menu or Plus Button *(see SoMachine, Programming Guide)*

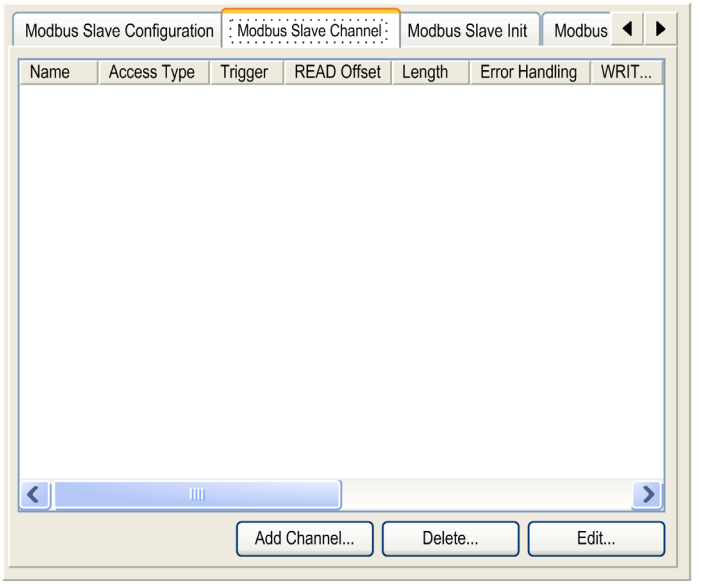
NOTE: The variable for the exchange is automatically created in the `%IWx` and `%QWx` of the **Modbus Serial Master I/O Mapping** tab.

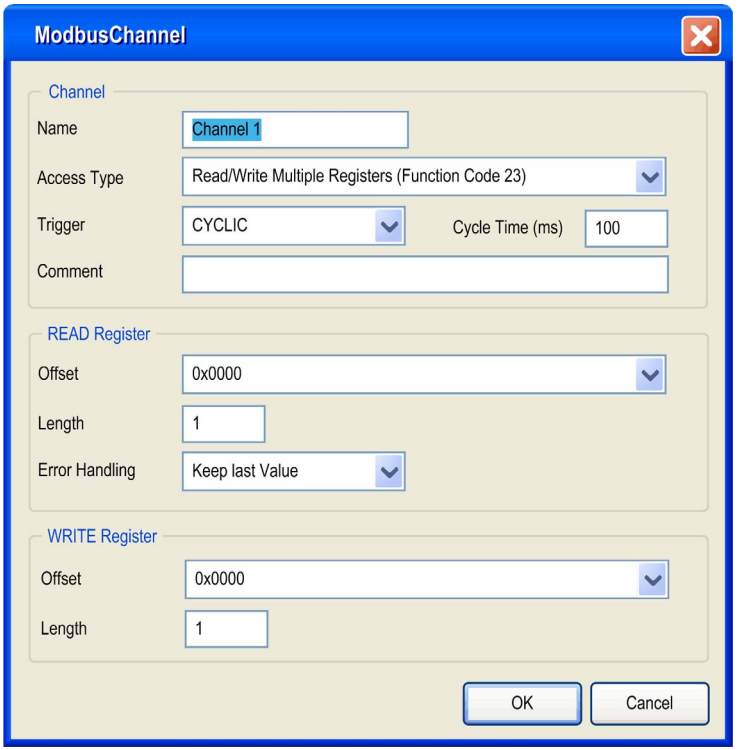
Configure a Device Added on the Modbus IOScanner

To configure the device added on the Modbus IOScanner, proceed as follow:

Step	Action
1	In the **Devices tree**, double-click **Generic Modbus Slave**. **Result:** The configuration window is displayed. 
2	Enter a **Slave Address** value for your device (choose a value from 1 to 247).
3	Choose a value for the **Response Timeout** (in ms).

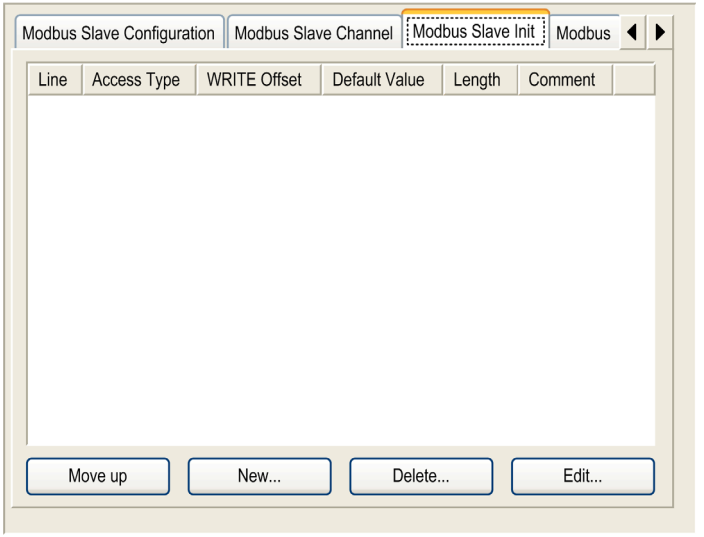
To configure the **Modbus Channels**, proceed as follow:

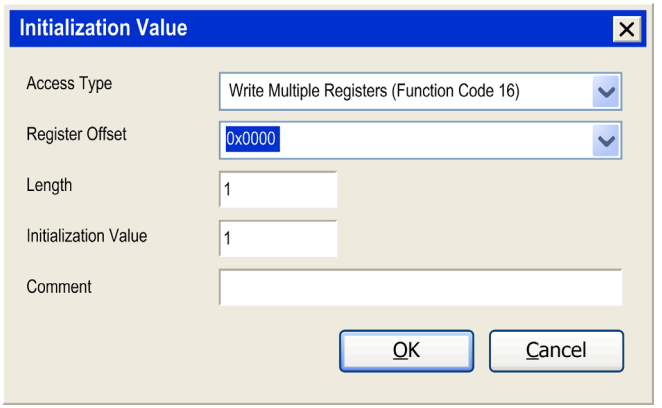
Step	Action
1	Click the **Modbus Slave Channel** tab: 

Step	Action
2	Click the **Add Channel** button: 

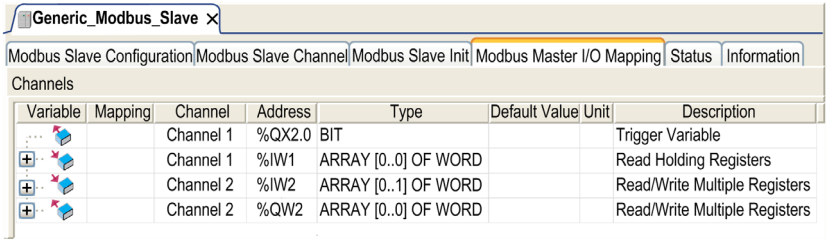
Step	Action
3	Configure an exchange: In the field **Channel**, you can add the following values: ● **Channel**: Enter a name for your channel. ● **Access Type**: Choose the exchange type: Read or Write or Read/Write multiple registers (i.e. %MW) *(see page 190)*. ● **Trigger**: Choose the trigger of the exchange. It can be either **CYCLIC** with the period defined in **Cycle Time (ms)** field or started by a **RISING EDGE** on a boolean variable (this boolean variable is then created in the **Modbus Master I/O Mapping** tab). ● **Comment**: Add a comment about this channel. In the field **READ Register** (if your channel is Read or Read/Write one), you can configure the %MW to be read on the Modbus slave. Those will be mapped on %IW (see Modbus Master I/O Mapping tab): ● **Offset**: Offset of the %MW to read. 0 means that the first object that will be read will be %MW0. ● **Length**: Number of %MW to be read. For example, if 'Offset' = 2 and 'Length' = 3, the channel will read %MW2, %MW3 and %MW4. ● **Error Handling**: choose the behavior of the related %IW in case of loss of communication. In the field**WRITE Register** (if your channel is Write or Read/Write one), you can configure the %MW to be written to the Modbus slave. Those will be mapped on %QW (see Modbus Master I/O Mapping tab): ● **Offset**: Offset of the %MW to write. 0 means that the first object that will be written will be %MW0. ● **Length**: Number of %MW to be written. For example, if 'Offset' = 2 and 'Length' = 3, the channel will write %MW2, %MW3 and %MW4.
4	Click the **Delete** button to remove a channel. Click the **Edit** button to change the parameters of a channel.
5	Click **OK** to validate the configuration of this channel.

To configure your **Modbus Initialization Value**, proceed as follow:

Step	Action
1	Click the **Modbus Slave Init** tab: 

Step	Action
2	Click **New** to create a new initialization value:  The **Initialization Value** window contains the following parameters: ● **Access Type**: Choose the exchange type: Read or Write or Read/Write multiple registers (that is, %MW) *(see page 190)*. ● **Register Offset**: Register number of register to be initialized. ● **Length**: Number of %MW to be read. For example, if 'Offset' = 2 and 'Length' = 3, the channel will read %MW2, %MW3 and %MW4. ● **Initialization Value**: Value the registers are initialized with. ● **Comment**: Add a comment about this channel.
3	Click **Move up** to change the position of a value in the list. Click **Delete** to remove a value in the list. Click **Edit** to change the parameters of a value.
4	Click **OK** to create a new **Initialization Value**.

To configure your **Modbus Master I/O Mapping**, proceed as follow:

Step	Action
1	Click the **Modbus Master I/O Mapping** tab: 
2	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**.
3	For more information on I/O mapping, refer to SoMachine Programming Guide.

Access Types

This table describes the different access types available:

Function	Function Code	Availability
`Read Coils`	1	**ModbusChannel**
`Read Discrete Inputs`	2	**ModbusChannel**
`Read Holding Registers` (default setting for the channel configuration)	3	**ModbusChannel**
`Read Input Registers`	4	**ModbusChannel**
`Write Single Coil`	5	**ModbusChannel Initialization Value**
`Write Single Register`	6	**ModbusChannel Initialization Value**
`Write Multiple Coils`	15	**ModbusChannel Initialization Value**
`Write Multiple Registers` (default setting for the slave initialization)	16	**ModbusChannel Initialization Value**
`Read/Write Multiple Registers`	23	**ModbusChannel**

Adding a Modem to a Manager

Introduction

A modem can be added to the following managers:

- ASCII Manager
- Modbus Manager
- SoMachine Network Manager

NOTE: Use Modem TDW-33 (which implements AT & A1 commands) if you need a modem connexion with SoMachine Network Manager.

Adding a Modem to a Manager

To add a modem to your controller, select the modem you want in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the manager node.

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)*

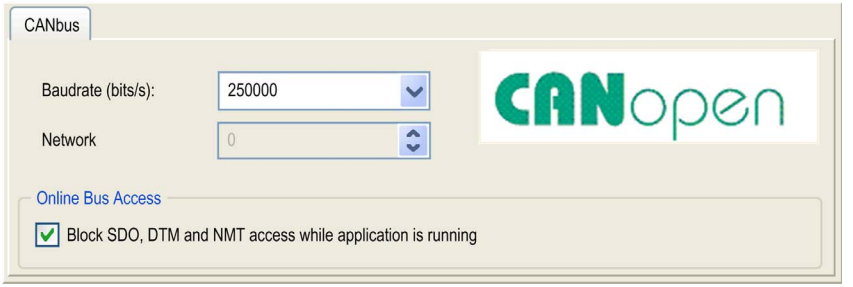
For further information, refer to Modem Library *(see SoMachine, Modem Functions, Modem Library Guide)*.

Chapter 15
CANopen Configuration

CANopen Interface Configuration

CAN Bus Configuration

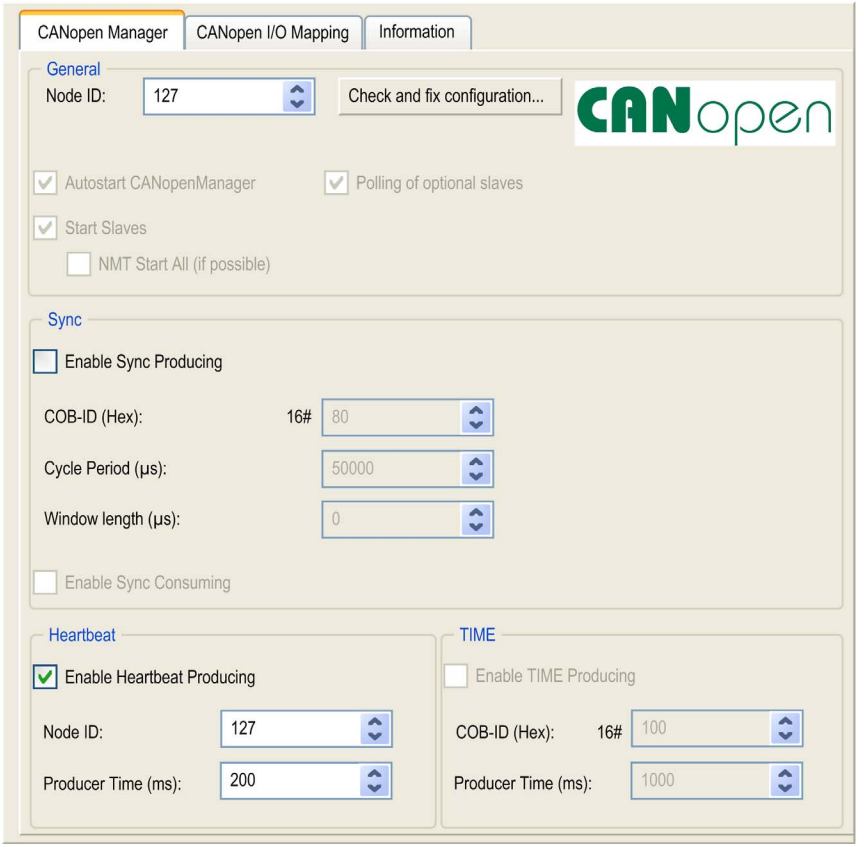
To configure the **CAN** bus of your controller, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **CAN_1**.
2	Configure the baudrate (by default: 250000 bits/s):  **NOTE:** The **Online Bus Access** option allows you to block SDO, DTM, and NMT sending through the status screen.

CANopen Manager Creation and Configuration

If the **CANopen Manager** is not already present below the **CAN** node, proceed as follows to create and configure it:

Step	Action
1	Select **CANopen Performance** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on one of the highlighted nodes. 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)*

Step	Action
2	Double-click **CANopen_Performance**. **Result:** The **CANopen Manager** configuration window appears: 

NOTE: If **Enable Sync Producing** is checked, the **CAN_x_Sync** task is added to the **Application → Task Configuration** node in the **Applications tree** tab.
Do not delete or change the **Type** or **External event** attributes of **CAN_x_Sync** tasks. If you do so, SoMachine will detect an error when you attempt to build the application, and you will not be able to download it to the controller.
If you uncheck the **Enable Sync Producing** option on the **CANopen Manager** sub-tab of the **CANopen_Performance** tab, the **CAN0_Sync** task is automatically deleted from your program.

Adding a CANopen Device

Refer to the SoMachine Programming Guide for more information on Adding Communication Managers and Adding Slave Devices to a Communication Manager.

CANopen Operating Limits

The Modicon M241 Logic Controller CANopen master has the following operating limits:

Maximum number of slave devices	63
Maximum number of Received PDO (RPDO)	252
Maximum number of Transmitted PDO (TPDO)	252

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Do not connect more than 63 CANopen slave devices to the controller.
- Program your application to use 252 or fewer Transmit PDO (TPDO).
- Program your application to use 252 or fewer Receive PDO (RPDO).

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

Chapter 16
Post Configuration

Introduction

This chapter describes how to generate and configure the post configuration file of the Modicon M241 Logic Controller.

What Is in This Chapter?

This chapter contains the following topics:

Post Configuration Presentation

Introduction

Post configuration is an option that allows you to modify some parameters of the application without changing the application. Post configuration parameters are defined in a file called **Machine.cfg**, which is stored in the controller.

By default, all parameters are set in the application.The parameters defined in the Post Configuration file are used instead of the corresponding parameters defined in the application. Not all parameters have to be specified in the Post Configuration file (for example: one parameter can change the IP address without changing the Gateway Address).

Parameters

The Post Configuration file allows you to change network parameters.

Ethernet parameters:
- IP Address
- Subnet Mask
- Gateway Address
- Transfer Rate
- IP Config Mode
- Device Name
- IPMaster Address *(see page 169)*

Serial Line parameters, for each serial line in the application (embedded port or PCI module):
- Baud rate
- Parity
- Data bits
- Stop bit

Profibus parameters, for each Profibus in the application (TM4 module):
- Station address
- Baud rate

NOTE: Parameter updates with a Post Configuration file that impacts parameters used by other devices via a communication port are not updated in the other device.
For example, if the IP address used by an HMI is updated in the configuration with a Post Configuration file, the HMI will still use the previous address. You must update the address used by the HMI independently.

Operating Mode

The Post Configuration file is read:

- after a Reset Warm command *(see page 66)*
- after a Reset Cold command *(see page 67)*
- after a reboot *(see page 68)*
- after an application download *(see page 70)*

Refer to Controller States and Behaviors *(see page 51)* for further details on controller states and transitions.

Post Configuration File Management

Introduction

The file **Machine.cfg** is located in the directory */usr/cfg*.

Each parameter specified by a variable type, variable ID, and value. The format is:

```
id[moduleType].param[paramId].paramField=value
```

where:

- `moduleType` is a numerical value, for example 111.
- `paramId` is a numerical value specifying the parameter to be modified, for example 10000.
- `paramField` is a string value that must be used in addition to the `paramId` to specify serial line parameters, for example, “Bauds”.
- `value` is the value assigned to the parameter. Its type depends on the parameter data type.

Each parameter will be defined on 3 lines in the Post Configuration file:

- The first line describes the internal 'path' for this parameter.
- The second line is a comment describing the parameter in a comprehensive way.
- The third line is the definition of the parameter (as described above) with its value.

Post Configuration File Generation

The Post Configuration file (**Machine.cfg**) is generated by SoMachine.

To generate the file, proceed as follows:

Step	Action
1	In the menu bar, choose **Build** → **Generate Post Configuration** **Result:** an explorer window appears.
2	Select the destination folder of the Post Configuration file.
3	Click **OK**.

NOTE: When you use SoMachine to create a Post Configuration file, it reads the value of each parameter currently assigned in your application program and then writes the new files using these values. This automatically generated a file explicitly assigns a value to every parameter that can be specified via Post configuration. After generating a Post configuration file, review the file and remove any parameter assignments that you wish to remain under the control of your application. Retain only those parameters assignments that you wish changed by the Post configuration function that are necessary to make you application portable.

Post Configuration File Transfer

After creating and modifying your Post Configuration file, transfer it to the `/usr/cfg` directory of the controller. The controller will not read the **Machine.cfg** file unless it is in this directory.

You can transfer the Post Configuration file by the following methods:
- SD card (with the proper script)
- download through the FTP server *(see page 132)*
- download with SoMachine controller device editor *(see page 76)*

Modifying a Post Configuration File

If the Post Configuration file is located in the PC, use a text editor to modify it.

NOTE: Do not change the text file encoding. The default encoding is ANSI.

To modify the Post Configuration file directly in the controller, use the **Setup** menu of the Web server *(see page 118)*.

Deleting the Post Configuration File

You can delete the Post Configuration file by the following methods:
- SD card (with the delete script)
- through the FTP server *(see page 132)*
- online with SoMachine controller device editor *(see page 76)*, **Files** tab

For more information on **Files** tab of the Device Editor, refer to SoMachine Programming Guide.

NOTE:
The parameters defined in the application will be used instead of the corresponding parameters defined in the Post Configuration file:
- after a Reset Warm command *(see page 66)*
- after a Reset Cold command *(see page 67)*
- after a reboot *(see page 68)*
- after an application download *(see page 70)*

Post Configuration Example

Post Configuration File Example

```
# TM241CEC24T / Ethernet_1 / IPAddress
# Ethernet IP address
id[45000].pos[7].id[111].param[0] = [172, 30, 3, 99]]

# TM241CEC24T / Ethernet_1 / SubnetMask
# Ethernet IP mask
id[45000].pos[7].id[111].param[1] = [255, 255, 0, 0]]

# TM241CEC24T / Ethernet_1 / GatewayAddress
# Ethernet IP gateway address
id[45000].pos[7].id[111].param[2] = [0, 0, 0, 0]]

# TM241CEC24T / Ethernet_1 / IPConfigMode
# IP configuration mode: 0:FIXED 1:BOOTP 2:DHCP
id[45000].pos[7].id[111].param[4] = 0

# TM241CEC24T / Ethernet_1 / DeviceName
# Name of the device on the Ethernet network
id[45000].pos[7].id[111].param[5] = 'my_Device'

# TM241CEC24T / Serial_Line_1 / Serial Line Configuration / Baudrate
# Serial Line Baud Rate in bit/s
id[45000].pos[8].id[40101].param[10000].Bauds = 115200

# TM241CEC24T / Serial_Line_1 / Serial Line Configuration / Parity
# Serial Line Parity (0=None, 1=Odd, 2=Even)
id[45000].pos[8].id[40101].param[10000].Parity = 0

# TM241CEC24T / Serial_Line_1 / Serial Line Configuration / DataBits
# Serial Line Data bits (7 or 8)
id[45000].pos[8].id[40101].param[10000].DataFormat = 8
```

```
# TM241CEC24T / Serial_Line_1 / Serial Line Configuration / StopBits
# Serial Line Stop bits (1 or 2)
id[45000].pos[8].id[40101].param[10000].StopBit = 1

# TM241CEC24T / Serial_Line_2 / Serial Line Configuration / Baudrate
# Serial Line Baud Rate in bit/s
id[45000].pos[9].id[40102].param[10000].Bauds = 19200

# TM241CEC24T / Serial_Line_2 / Serial Line Configuration / Parity
# Serial Line Parity (0=None, 1=Odd, 2=Even)
id[45000].pos[9].id[40102].param[10000].Parity = 2

# TM241CEC24T / Serial_Line_2 / Serial Line Configuration / DataBits
# Serial Line Data bits (7 or 8)
id[45000].pos[9].id[40102].param[10000].DataFormat = 8

# TM241CEC24T / Serial_Line_2 / Serial Line Configuration / StopBits
# Serial Line Stop bits (1 or 2)
id[45000].pos[9].id[40102].param[10000].StopBit = 1
```

Chapter 17
Connecting a Modicon M241 Logic Controller to a PC

Connecting the Controller to a PC

Overview

To transfer, run, and monitor the applications, connect the controller to a computer, that has SoMachine installed, using either a USB cable or an Ethernet connection (for those references that support an Ethernet port).

NOTICE
INOPERABLE EQUIPMENT
Always connect the communication cable to the PC before connecting it to the controller.
Failure to follow these instructions can result in equipment damage.

USB Powered Download

In order to execute limited operations, the M241 Logic Controller has the capability to be powered through the USB Mini-B port. A diode mechanism avoids having the logic controller both powered by USB and by the normal power supply, or to supply voltage on the USB port.

When powered only by USB, the logic controller executes the firmware and the boot project (if any) and the I/O board is not powered during boot (same duration as a normal boot). USB powered download initializes the internal flash memory with some firmware or some application and parameters when the controller is powered by USB. The preferred tool to connect to the controller is the **Controller Assistant**. Refer to the *SoMachine Controller Assistant User Guide*.

The controller packaging allows easy access to USB Mini-B port with minimum opening of the packaging. You can connect the controller to the PC with a USB cable. Long cables are not suitable for the USB powered download.

⚠ WARNING
INSUFFICENT POWER FOR USB DOWNLOAD
Do not use a USB cable longer than 3m (9.8 ft) for USB powered download.
Failure to follow these instructions can result in death, serious injury, or equipment damage.

NOTE: It is not intended that you use the USB Powered Download on an installed controller. Depending on the number of I/O expansion modules in the physical configuration of the installed controller, there may be insufficient power from your PC USB port to accomplish the download.

USB Mini-B Port Connection

TCSXCNAMUM3P: This USB cable is suitable for short duration connections such as quick updates or retrieving data values.

BMXXCAUSBH018: Grounded and shielded, this USB cable is suitable for long duration connections.

NOTE: You can only connect 1 controller or any other device associated with SoMachine and its component to the PC at any one time.

The USB Mini-B Port is the programming port you can use to connect a PC with a USB host port using SoMachine software. Using a typical USB cable, this connection is suitable for quick updates of the program or short duration connections to perform maintenance and inspect data values. It is not suitable for long-term connections such as commissioning or monitoring without the use of specially adapted cables to help minimize electromagnetic interference.

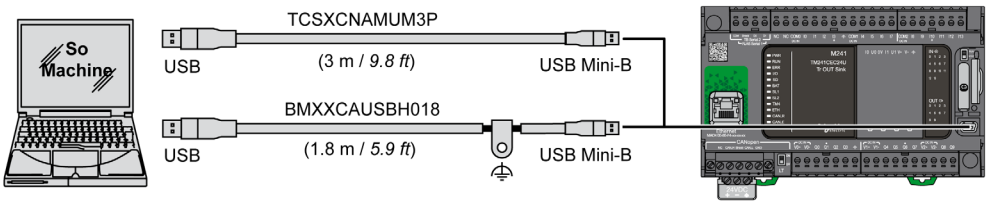
⚠ WARNING

UNINTENDED EQUIPMENT OPERATION OR INOPERABLE EQUIPMENT

- You must use a shielded USB cable such as a BMX XCAUSBH0•• secured to the functional ground (FE) of the system for any long-term connection.
- Do not connect more than one controller at a time using USB connections.

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

The communication cable should be connected to the PC first to minimize the possibility of electrostatic discharge affecting the controller.

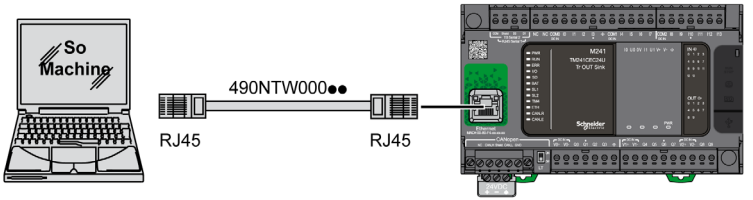


To connect the USB cable to your controller, follow the steps below:

Step	Action
1	**1a** If making a long-term connection using the cable BMXXCAUSBH018, or other cable with a ground shield connection, be sure to securely connect the shield connector to the functional ground (FE) or protective ground (PE) of your system before connecting the cable to your controller and your PC. **1b** If making a short-term connection using the cable TCSXCNAMUM3P or other non-grounded USB cable, proceed to step 2.
2	Connect your USB cable to the computer.
3	Open the hinged access cover.
4	Connect the Mini connector of your USB cable to the controller USB connector.

Ethernet Port Connection

You can also connect the controller to a PC using an Ethernet cable.



To connect the controller to the PC, do the following:

Step	Action
1	Connect your Ethernet cable to the PC.
2	Connect your Ethernet cable to the Ethernet port on the controller.

Chapter 18
SD Card

Introduction

This chapter describes how to transfer firmware, application, using an SD card to the Modicon M241 Logic Controller.

What Is in This Chapter?

This chapter contains the following topics:

Updating Modicon M241 Logic Controller Firmware

Introduction

The firmware updates for Modicon M241 Logic Controller are available on the http://www.schneider-electric.com website (in .zip format).

Updating the firmware is possible by:
- Using an SD card with a compatible script file
- Using the **Controller Assistant**

Performing a firmware change will delete the current application program in the device, including the Boot Application in Flash memory.

NOTICE
LOSS OF APPLICATION DATA ● Perform a backup of the application program to the hard disk of the PC before attempting a firmware change. ● Restore the application program to the device after a successful firmware change. **Failure to follow these instructions can result in equipment damage.**

If there is a power outage or communication interruption during the transfer of the application program or a firmware change, your device may become inoperative. If a communication interruption or a power outage occurs, reattempt the transfer.

NOTICE
INOPERABLE EQUIPMENT ● Do not interrupt the transfer of the application program or a firmware change once the transfer has begun. ● Do not place the device into service until the transfer has completed successfully. **Failure to follow these instructions can result in equipment damage.**

The serial line ports of your controller are configured for the SoMachine protocol by default when new or when you update the controller firmware. The SoMachine protocol is incompatible with that of other protocols such as Modbus Serial Line. Connecting a new controller to, or updating the firmware of a controller connected to, an active Modbus configured serial line can cause the other devices on the serial line to stop communicating. Make sure that the controller is not connected to an active Modbus serial line network before first downloading a valid application having the concerned port or ports properly configured for the intended protocol.

NOTICE
INTERRUPTION OF SERIAL LINE COMMUNICATIONS Be sure that your application has the serial line ports properly configured for Modbus before physically connecting the controller to an operational Modbus Serial Line network. **Failure to follow these instructions can result in equipment damage.**

Updating Firmware by SD Card

Follow these steps to update the firmware by an SD card:

Step	Action
1	Extract the .zip file to the root of the SD card. **NOTE:** The SD card folder \sys\cmd\ contains the download script file.
2	Remove power from the controller.
3	Insert the SD card into the controller.
4	Restore power to the controller. **NOTE:** The SD LED (green) is flashing during the operation.
5	Wait until the end of the download (theSD LED is solid green): ● In case of a detected error, the SD LED is turned off, and the ERR (red) and I/O (red) LEDs begin flashing. ● If the download ended successfully, the ERR LED (red) is flashing regularly, and the SD LED (green) is steady ON.
6	Remove the SD card from the controller. **Result**: The controller restarts automatically with new firmware if the download ended successfully.

Updating Firmware by Controller Assistant

Launch **SoMachine Central** and click **Maintenance** → **Controller Assistant** to open the **Controller Assistant** window.

To execute a complete firmware update of a controller without replacing the Boot application and data, proceed as follows:

Step	Action
1	On the **Home** dialog, click the **Read from....** controller button. **Result**: The **Controller selection** dialog opens.
2	Select the required connection type and controller and click the **Reading** button. **Result**: The image is transmitted from the controller to the computer. After this has been accomplished successfully, you are automatically redirected to the **Home** dialog.
3	Click the button **New / Process...** and then **Update firmware...**. **Result**: The dialog for updating the firmware opens.
4	Execute individual steps for updating the firmware in the current image (Changes are only effected in the image on your computer). In the final step, you can decide whether you want to create a backup copy of the image read by the controller. **Result**: Following the update of the firmware, you are automatically returned to the **Home** dialog.
5	On the **Home** dialog, click the **Write on....** controller button. **Result**: The **Controller selection** dialog opens.
6	Select the required connection type and controller and click the **Write** button. **Result**: The image is transmitted from your computer to the controller. After the transmission, you are automatically returned to the **Home** dialog.

For more information about the firmware update and creating a new flash disk with firmware, refer to Project Settings - Firmware Update and Flash Memory Organization *(see page 32)*.

File Transfer with SD Card

Introduction

The Modicon M241 Logic Controller allows file transfers with an SD card.

To upload or download files to the controller with an SD card, use one of the following methods:

- The clone function (use of an empty SD card)
- A script stored in the SD card

When an SD card is inserted into the SD card slot of the controller, the firmware searches and executes the script contained in the SD card (/sys/cmd/Script.cmd).

NOTE: The controller operation is not modified during file transfer.

The **Mass Storage (USB or SDCard)** editor lets you generate and copy the script and all necessary files into the SD card.

NOTE: The Modicon M241 Logic Controller accepts only SD card formatted in FAT or FAT32.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- You must have operational knowledge of your machine or process before connecting this device to your controller.
- Ensure that guards are in place so that any potential unintended equipment operation will not cause injury to personnel or damage to equipment.

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

If there is a power outage or communication interruption during the transfer of the application program or a firmware change, your device may become inoperative. If a communication interruption or a power outage occurs, reattempt the transfer.

NOTICE

INOPERABLE EQUIPMENT

- Do not interrupt the transfer of the application program or a firmware change once the transfer has begun.
- Do not place the device into service until the transfer has completed successfully.

Failure to follow these instructions can result in equipment damage.

Clone Function

The clone function allows you to upload the application from one controller and to download it only to a same controller reference.

This function clones every parameter of the controller (for example applications, firmware, data file, post configuration). Refer to Memory Mapping *(see page 27)*. However, for security reasons, it does not duplicate the Web Server/FTP password, nor any user access-rights, on any targeted machine.

NOTE: Ensure access-rights are disabled in the source controller before doing a clone operation. For more details about Access Rights, refer to the SoMachine Programming Guide.

This procedure describes how to upload to your SD card the current application stored in the controller:

Step	Action
1	Erase an SD card and set the card label as follows: **CLONExxx** **NOTE:** The label must begin with '**CLONE**' (not case sensitive), followed by any normal character.
2	Remove power from the controller.
3	Insert the prepared SD card in the controller.
4	Restore power to the controller. **Result:** The clone operation starts automatically. During the clone operation, the following LEDs are ON: **PWR**, **I/O**, and **SD**. **NOTE:** The clone operation lasts 2 or 3 minutes.
5	Wait until the clone operation is completed (the SD LED turns OFF). **Result**: The controller starts in normal application mode.
6	Remove the SD card from the controller.

This procedure describes how to download to your controller the current application stored in the SD card:

Step	Action
1	Remove power from the controller.
2	Insert the SD card into the controller.
3	Restore power to the controller. **Result:** The clone operation is in progress. **NOTE:** The SD LED (green) is flashing during the operation.

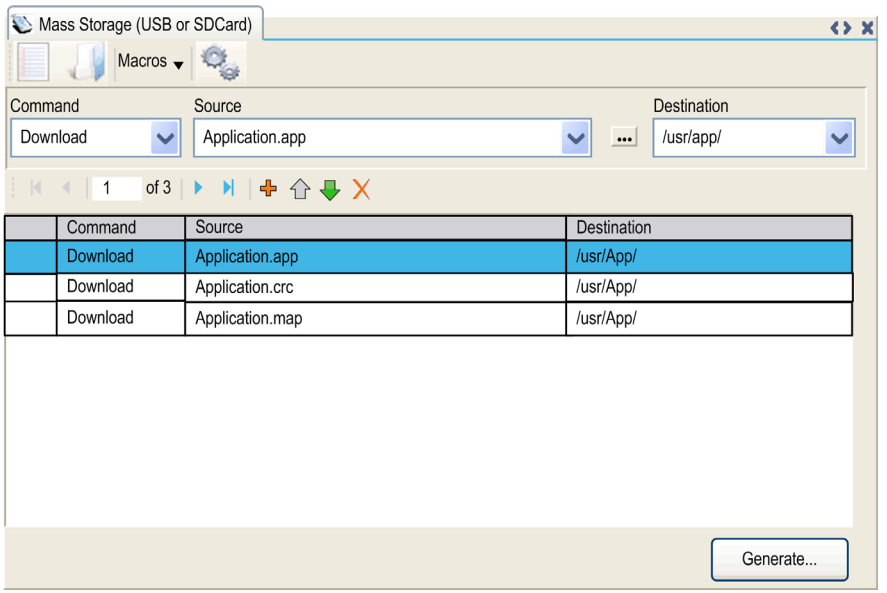
Step	Action
4	Wait until the end of the download (the SD LED is solid green): ● In case of a detected error, the SD LED is turned off, and the ERR (red) and I/O (red) LEDs begin flashing. ● If the download ended successfully, the ERR LED (red) is flashing regularly, and the SD LED (green) is steady ON.
5	Remove the SD card to restart the controller.

NOTE: If you wish to control access to the cloned application in the target controller, you will need to enable and establish user access-rights, and any Web Server/FTP passwords, which are controller-specific. For more details about Access Rights, refer to the SoMachine Programming Guide.

NOTE: Downloading a cloned application to the controller will first remove the existing application from controller memory, regardless of any user access-rights that may be enabled in the target controller.

Script and Files Generation with Mass Storage

Click **Project** → **Mass Storage (USB or SDCard)** in the main menu:



Element	Description
New	Create a new script.
Open	Open a script.
Macros	Insert a Macro. A macro is a sequence of unitary commands. A macro helps to perform many common operations such as upload application, download application, and so on.
Generate	Generate the script and all necessary files on the SD card.
Command	Basic instructions.
Source	Source file path on the PC or the controller.
Destination	Destination directory on the PC or the controller.
Add New	Add a script command.
Move Up/Down	Change the script commands order.
Delete	Delete a script command.

Commands descriptions:

Command	Description	Source	Destination	Syntax
Download	Download a file from the SD card to the controller.	Select the file to download.	Select the controller destination directory.	`'Download "/usr/Cfg/*"'`
SetNodeName	Sets the node name of the controller.	New node name.	Controller node name	`'SetNodeName "Name_PLC"'`
Upload	Upload files contained in a controller directory to the SD card.	Select the directory.	-	`'Upload "/usr/*"'`
Delete	Delete files contained in a controller directory. **NOTE:** Delete **"*"** does not delete system files.	Select the directory and enter a specific file name **Important:** by default, all directory files are selected.	-	`'Delete "/usr/SysLog/*"'`
	Removes the UserRights from the controller.	-	-	`'Delete "/usr/*"'`

Command	Description	Source	Destination	Syntax
Reboot	Restart the controller (only available at the end of the script).	-	-	`'Reboot'`

NOTE: When UserRights are activated on a controller and if the user is not allowed to read/write/delete file system, scripts used to **Upload**/**Download**/**Delete** files are disabled. It includes the clone operation. For more details about UserRights, refer to the SoMachine Programming Guide.

Macros description

Macros	Description	Directory/Files

Download App	Download the application from the SD card to the controller.	`/usr/App/*.app`
		`/usr/App/*.crc`
		`/usr/App/*.map`

Upload App	Upload the application from the controller to the SD card.	
Download Sources	Download the project archive from the SD card to the controller.	`/usr/App/*.prj`
Upload Sources	Upload the project archive from the controller to the SD card.	
Download Multi-files	Download multiple files from the SD card to a controller directory.	Defined by user
Upload Log	Upload the log files from the controller to the SD card.	`/usr/Log/*.log`

Transfer Procedure

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- You must have operational knowledge of your machine or process before connecting this device to your controller.
- Ensure that guards are in place so that any potential unintended equipment operation will not cause injury to personnel or damage to equipment.

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

Step	Action
1	Create the script with the **Mass Storage (USB or SDCard)** editor.
2	Click **Generate...** and select the SD card root directory. **Result**: The script and files are transferred on the SD card.

Step	Action
3	Insert the SD card into the controller. **NOTE:** The SD LED blinks green during transfer. For more information on the LED behavior during the transfer, refer to the Updating Firmware by SD Card *(see page 211)*.
4	Remove the SD card from the controller. **NOTE:** Changes will be applied after next restart.

When the controller has executed the script, the result is logged on the SD card (file `/sys/cmd/Cmd.log`).

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Consult the controller state and behavior diagram in this document to understand the state that will be assumed by the controller after you cycle power.

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

Appendices

Overview

This appendix lists the documents necessary for technical understanding of the Modicon M241 Logic Controller Programming Guide.

What Is in This Appendix?

The appendix contains the following chapters:

Appendix A
Functions to Get/Set Serial Line Configuration in User Program

Overview

This section describes the functions to get/set the serial line configuration in your program.

To use these functions, add the **M2xx Communication** library.

For further information on adding a library, refer to the SoMachine Programming Guide.

What Is in This Chapter?

This chapter contains the following topics:

GetSerialConf: Get the Serial Line Configuration

Function Description

`GetSerialConf` returns the configuration parameters for a specific serial line communication port.

Graphical Representation

Parameter Description

Input	Type	Comment
`Link`	LinkNumber *(see SoMachine, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*	`Link` is the communication port number.
`PointerToSerialConf`	POINTER TO `SERIAL_CONF` *(see page 225)*	`PointerToSerialConf` is the address of the configuration structure (variable of `SERIAL_CONF` type) in which the configuration parameters are stored. The `ADR` standard function must be used to define the associated pointer. (See the example below.)

Output	Type	Comment
`GetSerialConf`	`WORD`	This function returns: ● 0: The configuration parameters are returned ● 255: The configuration parameters are not returned because: ❍ the function was not successful ❍ the function is in progress

Example

Refer to the `SetSerialConf` *(see page 224)* example.

SetSerialConf: Change the Serial Line Configuration

Function Description

`SetSerialConf` is used to change the serial line configuration.

Graphical Representation

NOTE: Changing the configuration of the Serial Line(s) port(s) during programming execution can interrupt ongoing communications with other connected devices.

⚠ WARNING

LOSS OF CONTROL DUE TO UNEXPECTED CONFIGURATION CHANGE

Validate and test all the parameters of the `SetSerialConf` function before putting your program into service.

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

Parameter Description

Input	Type	Comment
`Link`	`LinkNumber` *(see SoMachine, Modbus and ASCII Read/Write Functions, PLC-Communication Library Guide)*	`LinkNumber` is the communication port number.
`PointerToSerialConf`	POINTER TO `SERIAL_CONF` *(see page 225)*	`PointerToSerialConf` is the address of the configuration structure (variable of `SERIAL_CONF` type) in which the new configuration parameters are stored. The `ADR` standard function must be used to define the associated pointer. (See the example below.) If 0, set the application default configuration to the serial line.

Output	Type	Comment
SetSerialConf	WORD	This function returns: ● 0: The new configuration is set ● 255: The new configuration is refused because: ❍ the function is in progress ❍ the input parameters are not valid

Example

```
VAR
    MySerialConf: SERIAL_CONF
    result: WORD;
END_VAR

(*Get current configuration of serial line 1*)
GetSerialConf(1, ADR(MySerialConf));

(*Change to modbus RTU slave address 9*)
MySerialConf.Protocol := 0;          (*Modbus RTU/Somachine protocol (in
this case CodesysCompliant selects the protocol)*)
MySerialConf.CodesysCompliant := 0; (*Modbus RTU*)
MySerialConf.address := 9;           (*Set modbus address to 9*)

(*Reconfigure the serial line 1*)
result := SetSerialConf(1, ADR(MySerialConf));
```

SERIAL_CONF: Structure of the Serial Line Configuration Data Type

Structure Description

The SERIAL_CONF structure contains configuration information about the serial line port. It contains these variables:

Variable	Type	Description
Bauds	DWORD	baud rate
InterframeDelay	WORD	minimum time (in ms) between 2 frames in Modbus (RTU, ASCII)
FrameReceivedTimeout	WORD	In the ASCII protocol, FrameReceivedTimeout allows the system to conclude the end of a frame at reception after a silence of the specified number of ms. If 0 this parameter is not used.
FrameLengthReceived	WORD	In the ASCII protocol, FrameLengthReceived allows the system to conclude the end of a frame at reception, when the controller received the specified number of characters. If 0, this parameter is not used.
Protocol	BYTE	0: Modbus RTU or SoMachine (see CodesysCompliant)
		1: Modbus ASCII
		2: ASCII
Address	BYTE	Modbus address 0 to 255 (0 for Master)
Parity	BYTE	0: none
		1: odd
		2: even
Rs485	BYTE	0: RS232
		1: RS485
ModPol (polarizartion resistor)	BYTE	0: no
		1: yes
DataFormat	BYTE	7 bits or 8 bits
StopBit	BYTE	1: 1 stop bit
		2: 2 stop bits
CharFrameStart	BYTE	In the ASCII protocol, 0 means there is no start character in the frame. Otherwise, the corresponding ASCII character is used to detect the beginning of a frame in receiving mode. In sending mode, this character is added at the beginning of the user frame.
CharFrameEnd1	BYTE	In the ASCII protocol, 0 means there is no second end character in the frame. Otherwise, the corresponding ASCII character is used to detect the end of a frame in receiving mode. In sending mode, this character is added at the end of the user frame.

Variable	Type	Description
CharFrameEnd2	BYTE	In the ASCII protocol, 0 means there is no second end character in the frame. Otherwise, the corresponding ASCII character is used (along with CharFrameEnd1) to detect the end of a frame in receiving mode. In sending mode, this character is added at the end of the user frame.
CodesysCompliant	BYTE	0: Modbus RTU
		1: SoMachine (when Protocol = 0)
CodesysNetType	BYTE	not used

Appendix B
Controller Performance

Processing Performance

Introduction

This chapter provides information about the M241 processing performance.

Logic Processing

This table presents logic processing performance for various logical instructions:

IL Instruction Type	Duration for 1000 Instructions
Addition/subtraction/multiplication of INT	42 µs
Addition/subtraction/multiplication of DINT	41 µs
Addition/subtraction/multiplication of REAL	336 µs
Division of REAL	678 µs
Operation on BOOLEAN, for example, Status:= Status and value	75 µs
LD INT + ST INT	64 µs
LD DINT + ST DINT	49 µs
LD REAL + ST REAL	50 µs

Communication and System Processing Time

The communication processing time varies, depending on the number of sent/received requests.

Response Time on Event

The response time presented in the following table represents the time between a signal rising edge on an input triggering an external task and the edge of an output set by this task. The event task also process 100 IL instructions before setting the output:

Minimum	Typical	Maximum
120 µs	200 µs	500 µs

Glossary

A

analog output
Converts numerical values within the logic controller and sends out proportional voltage or current levels.

application
A program including configuration data, symbols, and documentation.

application source
The collection of human-readable controller instructions, configuration data, HMI instructions, symbols, and other program documentation. The application source file is saved on the PC and you can download the application source file to most logic controllers. The application source file is used to build the executable program that runs in the logic controller.

ARP
(*address resolution protocol*) An IP network layer protocol for Ethernet that maps an IP address to a MAC (hardware) address.

B

BCD
(*binary coded decimal*) The format that represents decimal numbers between 0 and 9 with a set of 4 bits (a nybble/nibble, also titled as half byte). In this format, the 4 bits used to encode decimal numbers have an unused range of combinations.

For example, the number 2,450 is encoded as `0010 0100 0101 0000`.

BOOL
(*boolean*) A basic data type in computing. A `BOOL` variable can have one of these values: 0 (`FALSE`), 1 (`TRUE`). A bit that is extracted from a word is of type `BOOL`; for example, `%MW10.4` is a fifth bit of memory word number 10.

Boot application
(*boot application*) The binary file that contains the application. Usually, it is stored in the controller and allows the controller to boot on the application that the user has generated.

BOOTP

(*bootstrap protocol*) A UDP network protocol that can be used by a network client to automatically obtain an IP address (and possibly other data) from a server. The client identifies itself to the server using the client MAC address. The server, which maintains a pre-configured table of client device MAC addresses and associated IP addresses, sends the client its pre-configured IP address. BOOTP was originally used as a method that enabled diskless hosts to be remotely booted over a network. The BOOTP process assigns an infinite lease of an IP address. The BOOTP service utilizes UDP ports 67 and 68.

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.

continuous function chart language

A graphical programming language (an extension of the IEC61131-3 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 inputs of other blocks to create complex expressions.

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.

CRC

(*cyclical redundancy check*) A method used to determine the validity of a communication transmission. The transmission contains a bit field that constitutes a checksum. The message is used to calculate the checksum by the transmitter according to the content of the message. Receiving nodes, then recalculate the field in the same manner. Any discrepancy in the value of the 2 CRC calculations indicates that the transmitted message and the received message are different.

D

data log

The controller logs events relative to the user application in a *data log*.

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.)

DINT

(*double integer type*) Encoded in 32-bit format.

DNS

(*domain name system*) The naming system for computers and devices connected to a LAN or the Internet.

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.

DWORD

(*double word*) Encoded in 32-bit format.

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.

encoder

A device for length or angular measurement (linear or rotary encoders).

equipment

A part of a machine including sub-assemblies such as conveyors, turntables, and so on.

Ethernet

A physical and data link layer technology for LANs, also known as IEEE 802.3.

EtherNet/IP

(*Ethernet industrial protocol*) An open communications protocol for manufacturing automation solutions in industrial systems. EtherNet/IP is in a family of networks that implement the common industrial protocol at its upper layers. The supporting organization (ODVA) specifies EtherNet/IP to accomplish global adaptability and media independence.

expansion bus

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

F

FBD

(*function block diagram*) One of 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, which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

FE

(functional Earth) A common grounding connection to enhance or otherwise allow normal operation of electrically sensitive equipment (also referred to as functional ground in North America).

In contrast to a protective Earth (protective ground), a functional earth connection serves a purpose other than shock protection, and may normally carry current. Examples of devices that use functional earth connections include surge suppressors and electromagnetic interference filters, certain antennas, and measurement instruments.

FG

(*frequency generator*) A function that generates a square wave signal with programmable frequency.

firmware

Represents the BIOS, data parameters, and programming instructions that constitute the operating system on a controller. The firmware is stored in non-volatile memory within the controller.

flash memory

A non-volatile memory that can be overwritten. It is stored on a special EEPROM that can be erased and reprogrammed.

freewheeling

When a logic controller is in freewheeling scan mode, a new task scan starts as soon as the previous scan has been completed. Contrast with *periodic scan mode*.

FTP

(*file transfer protocol*) A standard network protocol built on a client-server architecture to exchange and manipulate files over TCP/IP based networks regardless of their size.

H

HE10

Rectangular connector for electrical signals with frequencies below 3 MHz, complying with IEC 60807-2.

I

I/O

(*input/output*)

ICMP

(*Internet control message protocol*) Reports errors detected and provides information related to datagram processing.

IEC

(*international electrotechnical commission*) A non-profit and non-governmental international standards organization that prepares and publishes international standards for electrical, electronic, and related technologies.

IEC 61131-3

Part 3 of a 3-part IEC standard for industrial automation equipment. IEC 61131-3 is concerned with controller programming languages and defines 2 graphical and 2 textual programming language standards. The graphical programming languages are ladder diagram and function block diagram. The textual programming languages include structured text and instruction list.

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).

instruction list language

A program written in the instruction list 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 (see IEC 61131-3).

INT

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

IP

(*Internet protocol* Part of the TCP/IP protocol family that tracks the Internet addresses of devices, routes outgoing messages, and recognizes incoming messages.

L

ladder diagram language
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 (see IEC 61131-3).

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).

LINT
(*long integer*) A whole number encoded in a 64-bit format (4 times `INT` or 2 times `DINT`).

LRC
(*longitudinal redundancy checking*) An error-detection method for determining the correctness of transmitted and stored data.

LWORD
(*long word*) A data type encoded in a 64-bit format.

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.

MAST
A processor task that is run through its programming software. The MAST task has 2 sections:
- **IN:** Inputs are copied to the IN section before execution of the MAST task.
- **OUT:** Outputs are copied to the OUT section after execution of the MAST task.

MIB
(*management information base*) An object database that is monitored by a network management system like SNMP. SNMP monitors devices are defined by their MIBs. Schneider Electric has obtained a private MIB, groupeschneider (3833).

ms
(*millisecond*)

MSB
(*most significant bit/byte* The part of a number, address, or field that is written as the left-most single value in conventional hexadecimal or binary notation.

N

network

A system of interconnected devices that share a common data path and protocol for communications.

NMT

(*network management*) CANopen protocols that provide services for network initialization, detected error control, and device status control.

node

An addressable device on a communication network.

O

open loop

Open loop control refers to a motion control system with no external sensors to provide position or velocity correction signals.

See also: *closed loop*.

P

PDO

(*process data object*) An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

PE

(*Protective Earth*) A common grounding connection to help avoid the hazard of electric shock by keeping any exposed conductive surface of a device at earth potential. To avoid possible voltage drop, no current is allowed to flow in this conductor (also referred to as *protective ground* in North America or as an equipment grounding conductor in the US national electrical code).

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.

program

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

protocol

A convention or standard definition that controls or enables the connection, communication, and data transfer between 2 computing system and devices.

PTO

(*pulse train outputs*) a fast output that oscillates between off and on in a fixed 50-50 duty cycle, producing a square wave form. The PTO is especially well suited for applications such as stepper motors, frequency converters, and servo motor control, among others.

PWM

(*pulse width modulation*) A fast output that oscillates between off and on in an adjustable duty cycle, producing a rectangular wave form (though you can adjust it to produce a square wave). The PTO is well adapted to simulate or approximate an analog output in that it regulates the voltage of the output over its period making it useful in light dimming or speed control applications, among others.

R

REAL

A data type that is defined as a floating-point number encoded in a 32-bit format.

RJ45

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

RPDO

(*receive process data object* An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

RTC

(*real-time clock*) A battery-backed time-of-day and calender clock that operates continuously, even when the controller is not powered for the life of the battery.

run

A command that causes the controller to scan the application program, read the physical inputs, and write to the physical outputs according to solution of the logic of the program.

S

scan

A function that includes:
- reading inputs and placing the values in memory
- executing the application program 1 instruction at a time and storing the results in memory
- using the results to update outputs

SDO

(*service data object*) A message used by the field bus master to access (read/write) the object directories of network nodes in CAN-based networks. SDO types include service SDOs (SSDOs) and client SDOs (CSDOs).

SFC

(*sequential function chart*) A language that is composed of steps with associated actions, transitions with associated logic condition, and directed links between steps and transitions. (The SFC standard is defined in IEC 848. It is IEC 61131-3 compliant.)

SINT

(*signed integer*) A 15-bit value plus sign.

SNMP

(*simple network management protocol*) A protocol that can control a network remotely by polling the devices for their status and viewing information related to data transmission. You can also use it to manage software and databases remotely. The protocol also permits active management tasks, such as modifying and applying a new configuration.

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.

STOP

A command that causes the controller to stop running an application program.

string

A variable that is a series of ASCII characters.

T

task

A group of sections and subroutines, executed cyclically or periodically for the MAST task or periodically for the FAST task.

A task possesses a level of priority and is linked to inputs and outputs of the controller. These I/O are refreshed in relation to the task.

A controller can have several tasks.

TCP

(*transmission control protocol*) A connection-based transport layer protocol that provides a simultaneous bi-directional transmission of data. TCP is part of the TCP/IP protocol suite.

terminal block

(*terminal block*) The component that mounts in an electronic module and provides electrical connections between the controller and the field devices.

TPDO

(*transmit process data object*) An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

U

UDINT

(*unsigned double integer*) Encoded in 32 bits.

UDP

(*user datagram protocol*) A connectionless mode protocol (defined by IETF RFC 768) in which messages are delivered in a datagram (data telegram) to a destination computer on an IP network. The UDP protocol is typically bundled with the Internet protocol. UDP/IP messages do not expect a response, and are therefore ideal for applications in which dropped packets do not require retransmission (such as streaming video and networks that demand real-time performance).

UINT

(*unsigned integer*) Encoded in 16 bits.

V

variable

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

W

watchdog

A watchdog is a special timer used to ensure that programs do not overrun their allocated scan time. The watchdog timer is usually set to a higher value than the scan time and reset to 0 at the end of each scan cycle. If the watchdog timer reaches the preset value, for example, because the program is caught in an endless loop, an error is declared and the program stopped.

WORD

A type encoded in a 16-bit format.

Index

R

S

T

W