Modicon M238 Logic Controller
Programming Guide

10/2016

EIO0000000384.09

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.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

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.

© 2016 Schneider Electric. All Rights Reserved.

Table of Contents

	Safety Information	7
	About the Book	9
Chapter 1	**About the Modicon M238 Logic Controller**	15
	Modicon M238 Logic Controller	15
Chapter 2	**How to Configure the Controller**	17
	How to Configure the Controller	17
Chapter 3	**Libraries**	19
	Libraries	19
Chapter 4	**Supported Standard Data Types**	21
	Supported Standard Data Types	21
Chapter 5	**Memory Mapping**	23
	RAM Memory Organization	24
	Relocation Table	28
Chapter 6	**Tasks**	31
	Maximum Number of Tasks	32
	Task Configuration Screen	33
	Task Types	35
	System and Task Watchdogs	38
	Task Priorities	39
	Default Task Configuration	40
Chapter 7	**Controller States and Behaviors**	41
7.1	Controller State Diagram	42
	Controller State Diagram	43
7.2	Controller States Description	48
	Controller States Description	48
7.3	State Transitions and System Events	52
	Controller States and Output Behavior	53
	Commanding State Transitions	56
	Error Detection, Types, and Management	62
	Remanent Variables	63

Chapter 8 Controller Device Editor 65
Controller Parameters 66
Applications 68
Controller Selection 69
PLC Settings 71
Services 73
Chapter 9 M238 Embedded Functions 75
HSC Embedded Function 76
Embedded I/Os Configuration 78
PTO_PWM Embedded Function 82
Chapter 10 Expansion Modules Configuration 85
General Description 85
Chapter 11 CANopen Configuration 87
CANopen Interface Configuration 87
Chapter 12 AS-Interface Configuration 91
Presentation of the AS-Interface V2 Fieldbus 92
General Functional Description 93
Software Setup Principles 96
Add an AS-Interface Master Module 97
Configure an AS-Interface Master 98
Add an AS-Interface Slave 100
Configure an AS-Interface Slave 103
Automatic Addressing of an AS-Interface V2 Slave 106
Modification of Slave Address 107
System Diagnostic in Online Mode 108
Programming for the AS-Interface V2 Fieldbus 112
Configuration of a Replaced AS-Interface V2 Slave 113
Chapter 13 Serial Line Configuration 115
Serial Lines Configuration 116
ASCII Manager 118
SoMachine Network Manager 120
Modbus IOScanner 121
Modbus Manager 129
Adding a Modem to a Manager 134
Chapter 14 499TWD01100 Ethernet/Modbus Gateway 135
Connection and Configuration of the Ethernet Gateway 135
Chapter 15 Connecting the Modicon M238 Logic Controller to a PC . 141
Connecting the Controller to a PC 141

Chapter 16 Loader Device Accessory 143
16.1 About the Loader Device Accessory 144
Loader Device Accessory Description 145
Physical Description 147
LED Status and Diagnostic 148
Firmware and SoMachine Software Compatibility 150
16.2 Transfer From SoMachine to the USB Memory Key 152
Transfer From SoMachine to the USB Memory Key 152
16.3 File Transfer with a USB Memory Key 153
Upload From the Controller to the USB Memory Key 154
Download From the USB Memory Key to the Controller 156
16.4 Other Functionalities 159
Set the Controller to RUNNING State 160
Change the Firmware of the Loader Device Accessory 161
Chapter 17 Changing the Controller Firmware 163
Changing Through Serial Line 164
Changing Through USB 167
Launching the Exec Loader Wizard 169
Step 1 - Welcome 170
Step 2 - Settings 171
Step 3 - File and Device Exec Properties 173
Step 4 - Transfer Progress 175
Chapter 18 Compatibility 177
Software and Firmware Compatibilities 177
Chapter 19 Modicon M238 Logic Controller - Troubleshooting and FAQ 179
Troubleshooting 180
Frequently Asked Questions 187
Appendices 189
Appendix A AS-Interface Library 191
ASI_CheckSlaveBit 192
ASI_CmdSetAutoAddressing 194
ASI_CmdSetDataExchange 196
ASI_CmdSetOfflineMode 198
ASI_MasterStatusCheck 200

ASI_SlaveAddressChange 202
ASI_SlaveParameterUpdate 205
ASI_SlaveStatusCheck 207
ASI_ReadParameterImage 209
Appendix B Function and Function Block Representation 211
Differences Between a Function and a Function Block 212
How to Use a Function or a Function Block in IL Language 213
How to Use a Function or a Function Block in ST Language. 217
Appendix C Functions to Get/Set Serial Line Configuration in User Program 221
GetSerialConf: Get the Serial Line Configuration 222
SetSerialConf: Change the Serial Line Configuration 223
SERIAL_CONF: Structure of the Serial Line Configuration Data Type 225
Appendix D Controller Performance 227
Processing Performance 227
Glossary 229
Index 239

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 configure your Modicon M238 Logic Controller.

NOTE: Read and understand this document and all related documents before installing, operating, or maintaining your Modicon M238 Logic Controller.

Validity Note

This document has been updated for the release of SoMachine V4.2.

Related Documents

Title of Documentation	Reference Number
SoMachine Programming Guide	EIO0000000067 (ENG); EIO0000000069 (FRE); EIO0000000068 (GER); EIO0000000071 (SPA); EIO0000000070 (ITA); EIO0000000072 (CHS)
Modicon M238 Logic Controller Hardware Guide	EIO0000000016 (ENG); EIO0000000017 (FRE); EIO0000000018 (GER); EIO0000000019 (SPA); EIO0000000020 (ITA); EIO0000000021 (CHS)
Modicon TM2 Expansion Modules Configuration Programming Guide	EIO0000000396 (ENG); EIO0000000397 (FRE); EIO0000000398 (GER); EIO0000000399 (SPA); EIO0000000400 (ITA); EIO0000000401 (CHS)
Modicon M238 Logic Controller System Functions and Variables M238 PLC System Library Guide	EIO0000000364 (ENG); EIO0000000757 (FRE); EIO0000000758 (GER); EIO0000000759 (SPA); EIO0000000760 (ITA); EIO0000000761 (CHS)

Title of Documentation	Reference Number
Modicon M238 Logic Controller High Speed Counting M238 HSC Library Guide	EIO0000000362 (ENG); EIO0000000747 (FRE); EIO0000000748 (GER); EIO0000000749 (SPA); EIO0000000750 (ITA); EIO0000000751 (CHS)
Modicon M238 Logic Controller Pulse Train Output, Pulse Width Modulation M238 PTOPWM Library Guide	EIO0000000363 (ENG); EIO0000000752 (FRE); EIO0000000753 (GER); EIO0000000755 (ITA); EIO0000000754 (SPA); EIO0000000756 (CHS)
SoMachine Modbus and ASCII Read/Write Functions PLC Communication Library Guide	EIO0000000361(ENG); EIO0000000742(FRE); EIO0000000743(GER); EIO0000000745(ITA); EIO0000000744(SPA); EIO0000000746(CHS)
SoMachine Modem Functions Modem Library Guide	EIO0000000552 (ENG); EIO0000000491 (FRE); EIO0000000492 (GER); EIO0000000494 (ITA); EIO0000000493 (SPA); EIO0000000495 (CHS)
SoMachine Compatibility and Migration User Guide	EIO0000001684 (ENG); EIO0000001685 (FRE); EIO0000001686 (GER); EIO0000001687 (ITA); EIO0000001688 (SPA); EIO0000001689 (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 M238 Logic Controller

Modicon M238 Logic Controller

Overview

The Schneider Electric Modicon M238 Logic Controller has a variety of powerful features. This controller can service a wide range of applications.

Key Features

The Modicon M238 Logic Controller is supported and programmed with the SoMachine Programming Software, which provides the following IEC61131-3 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 the controller using CFC (Continuous Function Chart) language.

The Modicon M238 Logic Controller can manage up to 7 tasks (1 MAST task and up to 6 other tasks).

The power supply of Modicon M238 Logic Controller is either:

- 24 Vdc
- 100...240 Vac

The Modicon M238 Logic Controller with DC power supply includes the following features:

- 14 digital inputs, including 8 fast inputs
- 10 digital outputs, including 4 fast outputs

The Modicon M238 Logic Controller with AC power supply includes the following features:

- 14 digital inputs, including 8 fast inputs
- 10 digital outputs, including 6 relay outputs

Modicon M238 Logic Controller Range

The following table describes the M238 range and features:

Reference	Power Supply	Serial Ports	CANopen Master	Digital Inputs	Digital Outputs	Memory size
M238 DC Range						
TM238LFDC24DT	24 Vdc	SL1: RS232/RS485 SL2: RS485	Yes	8 fast inputs(1) + 6 regular inputs	4 transistor fast outputs(2) + 6 transistor regular outputs	2 MB
TM238LDD24DT	24 Vdc	SL1: RS232/RS485	No			1 MB
M238 AC Range						
TM238LFAC24DR	100...240 Vac	SL1: RS232/RS485 SL2: RS485	Yes	8 fast inputs(1) + 6 regular inputs	4 transistor outputs + 6 relay outputs	2 MB
TM238LDA24DR	100...240 Vac	SL1: RS232/RS485	No			1 MB

(1) The fast inputs can be used either as regular inputs or as fast inputs for counting or event functions.

(2) The fast outputs can be used either as regular outputs or as fast outputs for PTO (Pulse Train Output), HSC (High Speed Counter), PWM (Pulse Width Modulation), or FG(Frequency Generator) functions.

Chapter 2
How to Configure the Controller

How to Configure the Controller

Introduction

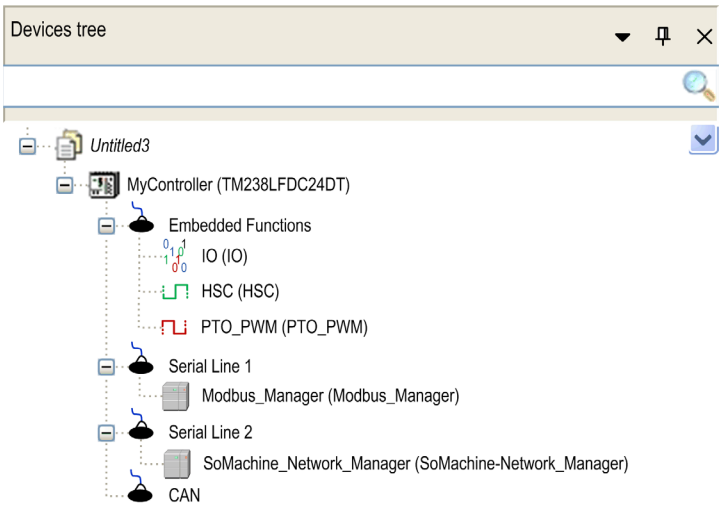
First, create a new project or open an existing project in the SoMachine software.

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	Description
Embedded Functions	Presents the Embedded Functions of the M238.
Serial Line 1 **Serial Line 2** **CAN**	Embedded communications interfaces. **NOTE:** Serial Line 2 and CAN are available only on TM238LFDC24DT and TM238LFAC24DR

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 M238 Logic Controller

When you select a Modicon M238 Logic Controller for your application, SoMachine automatically loads the following libraries:

Library name	Description
IoStandard	**CmpIoMgr** 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.
M238 PLCSystem *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*	Contains functions and variables to get information and send commands to the controller system.
M238 HSC *(see Modicon M238 Logic Controller, High Speed Counting, M238 HSC Library Guide)*	Contains function blocks and variables to get information and send commands to the Fast Inputs/Outputs of the Modicon M238 Logic Controller. These function blocks permit you to implement HSC (High Speed Counting) functions on the Fast Inputs/Outputs of the Modicon M238 Logic Controller.

Library name	Description
M238 PTOPWM *(see Modicon M238 Logic Controller, Pulse Train Output, Pulse Width Modulation, M238 PTOPWM Library Guide)*	Contains function blocks and variables to get information and send commands to the Fast Inputs/Outputs of the Modicon M238 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 M238 Logic Controller.
M238 Relocation Table *(see page 28)*	Allows 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
LREAL	2.2250738585072014e-308	1.7976931348623158e+308	64 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 M238 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:

Topic	Page
RAM Memory Organization	24
Relocation Table	28

RAM Memory Organization

Introduction

This section describes the RAM (Random Access Memory) size for different areas of the Modicon M238 Logic Controller.

TM238LFDC24DT and TM238LFAC24DR Memory

The RAM size is 2 Mbytes composed of 2 areas:

- 1048 kbytes System Area for Operating System memory
- 1000 kbytes Customer Area for dedicated application memory

Memory containing Persistent and Retain variables is preserved and protected by an external battery during power outages.

This table shows the different types of memory areas with their sizes in the TM238LFDC24DT and TM238LFAC24DR memory:

Area	Element	Size (Bytes)
System Area 1048 kbytes	System Area Mappable Addresses %MW0...%MW59999	120000
	System and Diagnostic variables (%MW60000...%MW60199) This memory is accessible through Modbus requests only. These must be read-only requests.	400
	Dynamic Memory Area: Read Relocation Table *(see page 28)* (%MW60200...%MW61999) This memory is accessible through Modbus requests only. These must be read-only requests.	3600
	Reserved	400
	Dynamic Memory Area: Write Relocation Table *(see page 28)* (%MW62200...%MW63999) This memory is accessible through Modbus requests only. These can be read or write requests.	3600
	Reserved	945152
Customer Area 1000 kbytes	Variables (including Retain and Persistent variables, see table below)	1024000[1]
	Application	
	Libraries *(see page 26)*	
	Symbols	

(1) Size checked at build time and should not exceed the value indicated in the table.

10568 bytes Battery Saved RAM	
8168 bytes	Retain Variables [2]
400 bytes	Persistent Retain Variables
2000 bytes	%MW0...%MW999
(2) Not all the 8168 bytes are available for the customer application because some libraries may use Retain Variables.	

TM238LDD24DT and TM238LDA24DR Memory

The RAM size is 1 Mbytes composed of 2 areas:

- 524 kbytes System Area for Operating System memory
- 500 kbytes Customer Area for dedicated application memory

Memory containing Persistent and Retain variables is preserved and protected by an external battery during power outages.

This table shows the different types of areas with their sizes for the TM238LDD24DT and TM238LDA24DR memory:

Area	Element	Size (Bytes)
System Area 524 kbytes	System Area Mappable Addresses %MW0...%MW59999	120000
	System and Diagnostic variables (%MW60000...%MW60199) This memory is accessible through Modbus requests only. These must be read-only requests.	400
	Dynamic Memory Area: Read Relocation Table *(see page 28)* (%MW60200...%MW61999) This memory is accessible through Modbus requests only. These must be read-only requests.	3600
	Reserved	400
	Dynamic Memory Area: Write Relocation Table *(see page 28)* (%MW62200...%MW63999) This memory is accessible through Modbus requests only. These can be read or write requests.	3600
	Reserved	408576
Customer Area 500 kbytes	Variables (including Retain and Persistent variables, see table below)	512000[1]
	Application	
	Libraries *(see page 26)*	
	Symbols	
(1) Size checked at build time and should not exceed the value indicated in the table.		

10568 bytes Battery Saved RAM	
8168 bytes	Retain Variables [2]
400 bytes	Persistent Retain Variables
2000 bytes	%MW0...%MW999
(2) Not all the 8168 bytes are available for the customer application because some libraries may use Retain Variables.	

System Variables

For more information on system variables, refer to the *M238 PLCSystem Library Guide*.

Library Sizes

Library Name	Average Size	Comment
M238 HSC *(see Modicon M238 Logic Controller, High Speed Counting, M238 HSC Library Guide)*	10 kbytes	Depends on the functions used.
M238 PLCSystem *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*	25 kbytes	Always embedded in the application. The use of the functions does not consume additional memory.
M238 PTOPWM *(see Modicon M238 Logic Controller, Pulse Train Output, Pulse Width Modulation, M238 PTOPWM Library Guide)*	10 kbytes	Depends on the functions used.
PLC Communication	20 kbytes	Depends on the functions used.
CANopen Stack	115 kbytes	Depends on the functions used. Each CANopen Slave consumes approximately an additional 10 kbytes of memory.

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.

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 M238 PLCSystem Library Guide.

Adding a Relocation Table

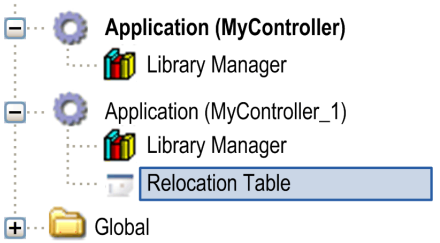
This table describes how to add a **Relocation Table** to your project:

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.

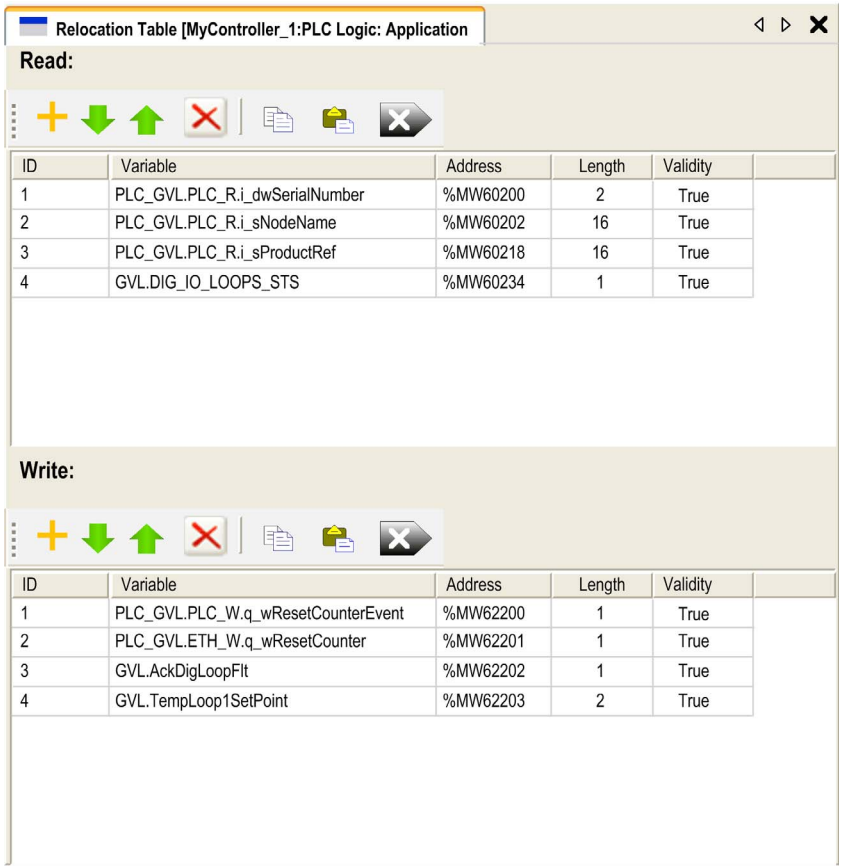
Relocation Table Editor

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:



ID	Variable	Address	Length	Validity
1	PLC_GVL.PLC_R.i_dwSerialNumber	%MW60200	2	True
2	PLC_GVL.PLC_R.i_sNodeName	%MW60202	16	True
3	PLC_GVL.PLC_R.i_sProductRef	%MW60218	16	True
4	GVL.DIG_IO_LOOPS_STS	%MW60234	1	True

ID	Variable	Address	Length	Validity
1	PLC_GVL.PLC_W.q_wResetCounterEvent	%MW62200	1	True
2	PLC_GVL.ETH_W.q_wResetCounter	%MW62201	1	True
3	GVL.AckDigLoopFlt	%MW62202	1	True
4	GVL.TempLoop1SetPoint	%MW62203	2	True

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:

Topic	Page
Maximum Number of Tasks	32
Task Configuration Screen	33
Task Types	35
System and Task Watchdogs	38
Task Priorities	39
Default Task Configuration	40

Maximum Number of Tasks

Maximum Number of Tasks

The maximum number of tasks you can define for the Modicon M238 Logic Controller are:

- Total number of tasks = 7
- Cyclic tasks = 3
- Freewheeling tasks = 1
- Event tasks = 2
- External Event tasks = 4

NOTE: The total number of Freewheeling task, Cyclic tasks, and Event tasks must not be greater than 3.

Special Considerations for Freewheeling

A Freewheeling task *(see page 36)* 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 38)*.

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.

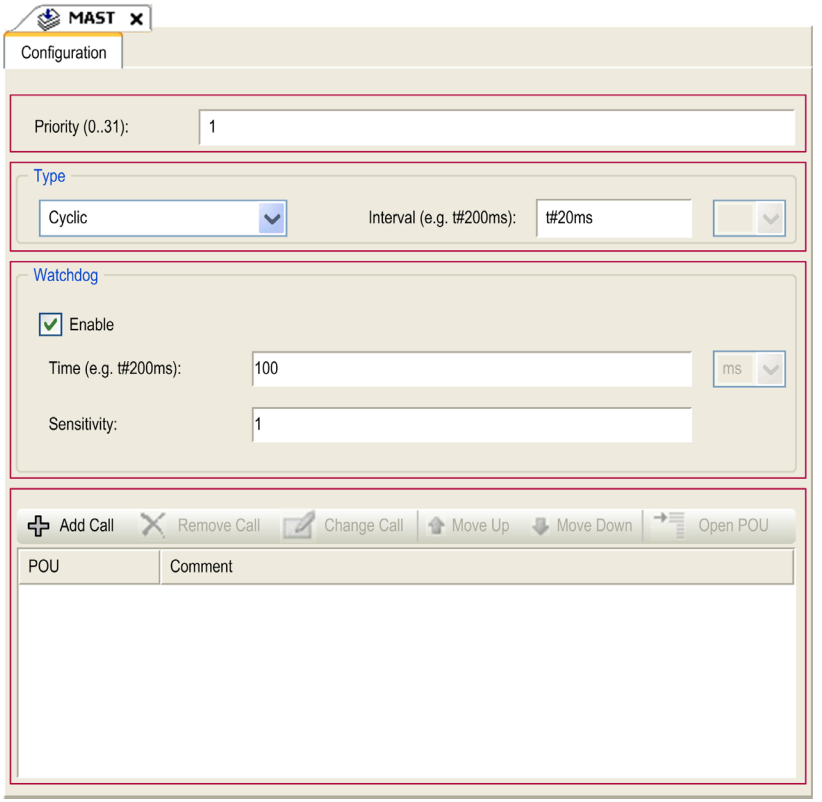
Task Configuration Screen

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 39)*.
Type	These task types are available: ● Cyclic *(see page 35)* ● Event *(see page 37)* ● External *(see page 37)* ● Freewheeling *(see page 36)*
Watchdog	To configure the watchdog *(see page 38)*, 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 *(see page 43)*.
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.

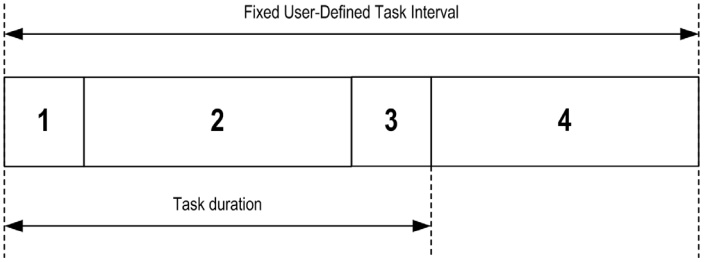
Task Types

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:



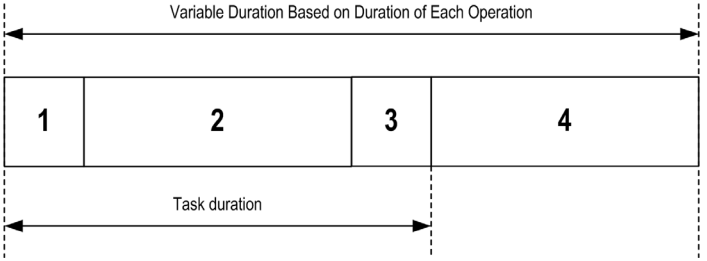
1. **Read Inputs:** The physical input states are written to the %I input memory variables and other system operations are executed.
2. **Task Processing:** The user code (POU, and so on) defined in the task is processed. The %Q output memory variables are updated according to your application program 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 M238 Logic Controller Settings *(see page 71)*.
   For more information on I/O behavior, refer to Controller States Detailed Description *(see page 48)*.
   NOTE: Expansion I/Os are always physically updated by the MAST task.
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: 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 M238 Logic Controller Settings *(see page 71)*.
   For more information on I/O behavior, refer to Controller States Detailed Description *(see page 48)*.
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.

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 Threshold cross event. To associate the **HSC4_TH3** event to an External Event task, select it from the External event drop-down list on the **Configuration** tab.

Depending on the related product, there are up to 2 types of events that can be associated with an External Event task:

- Rising edge on Fast input (`%IX0.0 ...%IX0.7` inputs)
- HSC thresholds

System and Task Watchdogs

Introduction

Two types of watchdog functionality are implemented for the Modicon M238 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

Two system watchdogs are defined for the Modicon M238 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 2 system watchdogs are defined as follows:

- If all of the tasks require more than 80% of the processor resources for more than 3 seconds, a system error is detected. The controller enters the HALT state.
- If the lowest priority task of the system is not executed during an interval of 20 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.

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 39)* for more information on priority settings. Refer to System and Task Watchdogs *(see page 38)* 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. If you do so, 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:

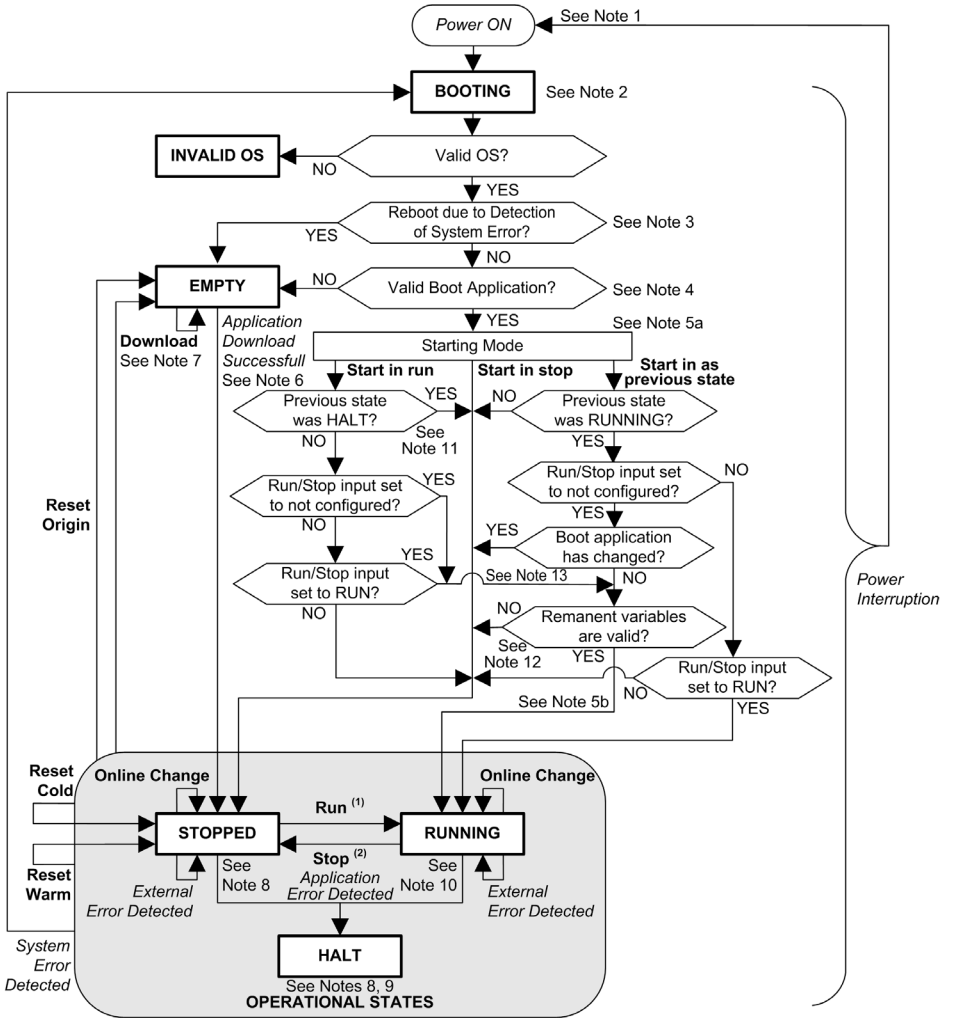
Section	Topic	Page
7.1	Controller State Diagram	42
7.2	Controller States Description	48
7.3	State Transitions and System Events	52

Section 7.1
Controller State Diagram

Controller State Diagram

Controller State Diagram

The following 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 56)*.

(2) For details on RUNNING to STOPPED state transition, refer to Stop Command *(see page 56)*.

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 53)* for further details.

Note 2

The boot process can take up to 10 seconds under normal conditions. The outputs will assume their initialization states.

Note 3

In some cases, when a system error is detected, it will cause the controller to automatically reboot into the EMPTY state as if no Boot application were present in the Flash memory. However, the Boot application is not actually deleted from the Flash memory.

Note 4

The application is loaded into RAM after verification of a valid Boot application.

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 59)*.

Note 5a

The **Starting Mode** is set in the **PLC settings** tab of the Controller Device Editor.

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 or the last controller state before the download.

However, there are two important 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. 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 48)* 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 (system watchdog or internal detected error), a cycle power is mandatory.

Note 10

The RUNNING state has two exception conditions.

They are:

- RUNNING with External Detected Error: this exception condition is indicated by the Err Status LED, which displays 1 red flash. You may exit this state by clearing the external detected error. No controller commands are required but may however include a power cycle to the controller. For more information, refer to Expansion Module Configuration, Introduction *(see page 85)*.
- RUNNING with Breakpoint: this exception condition is indicated by the RUN Status LED, which displays 1 green flash. Refer to Controller States Description *(see page 48)* for further details.

Note 11

When Starting Mode is set to Start in run and if the Run/Stop input is not configured, the controller will reboot in STOPPED state. A second reboot will be necessary to set the controller in RUNNING state.

Note 12

Remanent variables can be invalid if battery is not present for example.

Note 13

The boot application can be different from the application loaded. It can happen when the boot application was downloaded through USB Key, 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 M238 PLCSystem Library *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*

Controller States Table

The following table describes the controller states:

Controller State	Description	RUN LED	Err LED
BOOTING	The controller executes the boot firmware and its own internal self-tests. It then checks the checksum of the firmware and user applications. It does not execute the application nor does it communicate.	Off	Red flashing
INVALID_OS	There is not a valid firmware file present In the Flash memory. The controller does not execute the application. Communication is only possible through the USB host port, and then only for uploading a valid OS. Refer to Upgrading an M238 Firmware *(see page 163)*.	Off	Red flashing
EMPTY	There is no application present or an invalid application.	Off	3 red flashes

Controller State	Description	RUN LED	Err LED
EMPTY after detection of a *System Error*	This state is the same as the normal EMPTY state except that a flag is set to make it appear as if no Boot Application is present (no Application is loaded) and the LED indications are different.	Off	Rapid red flashing
RUNNING	The controller is executing a valid application.	Green	Off
RUNNING with Breakpoint	This state is the 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 the SoMachine Menu Commands Online Help.	Single green flash	Off
RUNNING with detection of an *External Error*	This state is the same as the normal RUNNING state except the LED indications are different.	Green	Single red flash
STOPPED	The controller has a valid application that is stopped. See Details of the STOPPED State *(see page 50)* for an explanation of the behavior of outputs and field buses in this state.	Green Flashing	Off
STOPPED with detection of an *External Error*	This state is the same as the normal STOPPED state except the LED indications are different.	Green flashing	Single red flash
HALT	The controller stops executing the application because it has detected an application Error. This description is the same as for the STOPPED state with the following exceptions: ● The task responsible for the Application Detected Error always behaves as if the **Update IO while in stop** option was not selected. All other tasks follow the actual setting. ● The LED indications are different	Green flashing	Red

Details of the STOPPED State

The following statements are true for the STOPPED state:

- Serial (Modbus, ASCII, etc.), 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. 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: If Q0, Q1, Q2 or Q3 outputs are configured for PTO, PWM, FG, or HSC operation, they fallback to a value of 0 irrespective of the configured fallback setting. For PTO operation, outputs Q0, Q1, Q2, and Q3 execute a fast stop deceleration. Outputs configured for PWM, FG, and HSC go immediately to 0.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Design and program your system so that controlled equipment assumes a safe state when the controller enters fallback mode if you use outputs Q0, Q1, Q2, or Q3 for PTO, PWM, FG, or HSC operation.

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

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 Serial, and USB connections. However, the physical outputs are unaffected and retain the state specified by the configuration options.
  NOTE: if Q0, Q1, Q2 or Q3 outputs are configured for PTO, PWM, FG, or HSC operation, they fallback to a value of 0 irrespective of the configured fallback setting. For PTO operation, outputs Q0, Q1, Q2, and Q3 execute a fast stop deceleration. Outputs configured for PWM, FG, and HSC go immediately to 0.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Design and program your system so that controlled equipment assumes a safe state when the controller enters fallback mode if you use outputs Q0, Q1, Q2, or Q3 for PTO, PWM, FG, or HSC operation.

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

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:

Topic	Page
Controller States and Output Behavior	53
Commanding State Transitions	56
Error Detection, Types, and Management	62
Remanent Variables	63

Controller States and Output Behavior

Introduction

The Modicon M238 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.

NOTE: An exception to this is if the RUNNING with External Error Detected state has been provoked by a I/O expansion bus error. For more information, refer to Expansion Module Configuration, Introduction *(see page 85)*.

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 and HALT controller states. 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 48)* 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. 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 48)* for more details on these variations.

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.

⚠ 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 *(see page 80)* for more information.
- SoMachine Online Menu: Select the **Start** command.
- By an external call via Modbus request using the PLC_W.q_wPLCControl and PLC_W.q_uiOpenPLCControl system variables of the M238 PLCSystem Library *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*.
- **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 *(see page 43)* 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 *(see page 80)* for more information.
- SoMachine Online Menu: Select the **Stop** 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 M238 PLCSystem Library *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*.
- **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.
- The controller is restarted into the STOPPED state automatically under certain conditions.

Refer to Controller State Diagram *(see page 43)* 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 M238 PLCSystem Library *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*.

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.

For details on variables, refer to Remanent Variables *(see page 63)*.

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 M238 PLCSystem Library *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)*.

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.

For details on variables, refer to Remanent Variables *(see page 63)*.

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. The Boot application file is 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. All I/O are reset to their initialization values.

For details on variables, refer to Remanent Variables *(see page 63)*.

Reboot

Effect: Commands a reboot of the controller.

Starting Conditions: Any state.

Methods for Issuing the Reboot Command:

- Power cycle

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 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 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 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.

For details on variables, refer to Remanent Variables *(see page 63)*.

NOTE: The Check context test *(see page 43)* concludes that the context is valid when the application and the remanent variables are the same as defined in the Boot application. Remanent variables are only maintained when there is sufficient battery *(see M238 Logic Controller, Hardware Guide)*.

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 *(see page 43)*.

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. All I/O are reset to their initialization values and then set to the new user-configured default values after the download is complete.

For details on variables, refer to Remanent Variables *(see page 63)*.

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: ● The controller is configured for an expansion module that is not present or not detected(1). ● 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

(1) Expansion modules may appear to be absent for any number of reasons, even if the absent I/O module is physically present on the bus. For more information, refer to Expansion Module Configuration, Introduction *(see page 85)*.

NOTE: Refer to the M238 PLCSystem Library Guide *(see Modicon M238 Logic Controller, System Functions and Variables, M238 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".

Remanent variables are retained only if the battery *(see M238 Logic Controller, Hardware Guide)* charge is sufficient.

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?

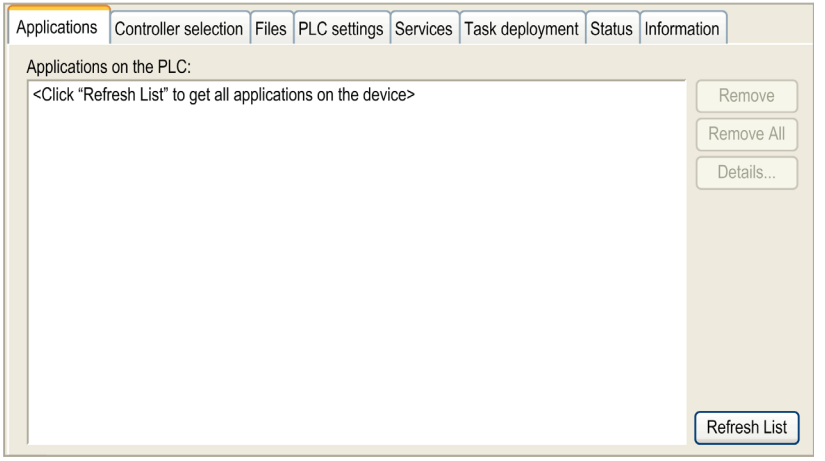
This chapter contains the following topics:

Topic	Page
Controller Parameters	66
Applications	68
Controller Selection	69
PLC Settings	71
Services	73

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 69)*	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.	Online mode only
Files	File management between the PC and the controller.	Online mode only
PLC settings *(see page 71)*	Configuration of: ● application name ● I/O behavior in stop ● bus cycle options	–
Services *(see page 73)*	Lets you configure the online services of the controller (RTC, device identification).	Online mode only

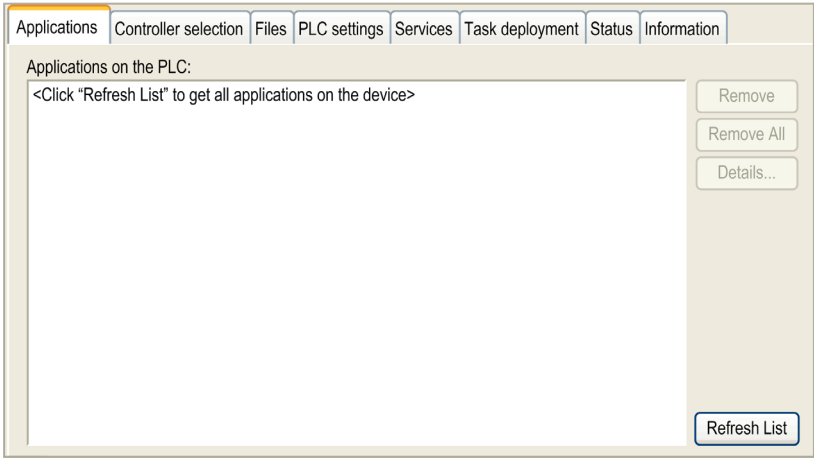
Tab	Description	Restriction
Task deployment	Displays a list of I/Os and their assignments to tasks.	After compilation only
Status	Displays device-specific status and diagnostic messages.	–
Information	Displays general information about the device (name, description, provider, version, image).	–

Applications

Overview

The **Applications** view of the device editor serves to scan and to remove applications on the controller. Information on the content of the application can be available as well as some details on the application properties.

Applications view of the device editor:



The **Applications** view provides the following elements:

Element		Description
Applications on the PLC		List of the names of applications, which have been found on the controller during the last scan.
Buttons	**Remove**	The application currently selected in the list will be removed from the controller.
	Remove all	All applications are removed from the controller.
	Refresh List	The controller is scanned for applications and the list is updated.

Controller Selection

Introduction

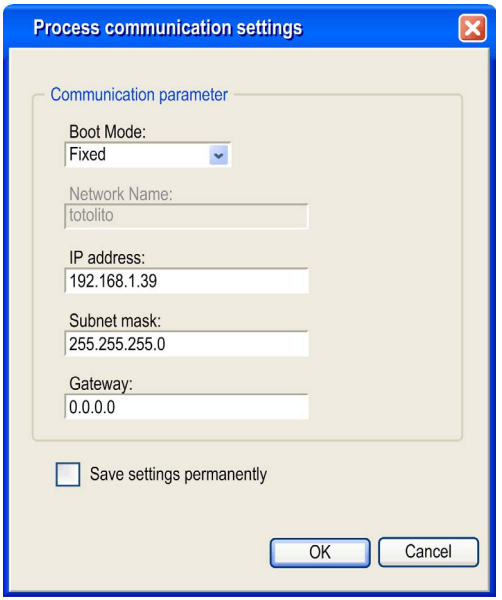
This tab allows you to manage the connection from the PC to the controller:

- 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.

The **Process communication settings** window appears as shown below:



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

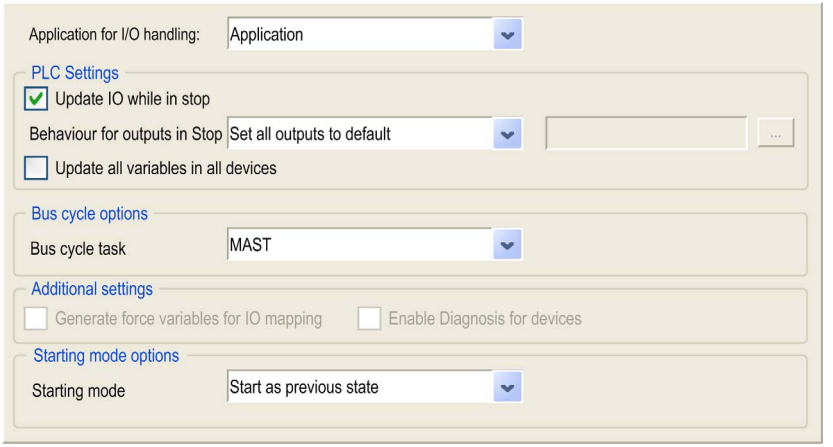
- Without the **Save settings permanently** option:
  Configure the communication parameters and click **OK**. These settings are immediately taken into account and are not kept if the controller is reset. For the next resets, the communication parameters configured into the application are taken into 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**: The current values will not be modified. • **Set all outputs to default**: The default (fallback) values resulting from the mapping will be assigned. **NOTE:** This option is not taken into account for the outputs used by the HSC, PTO, PWM, or Frequency Generator.
	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.

Element		Description
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.
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 43)*. Select with this option one of these starting modes: ● Start as previous state ● Start in stop ● Start in run

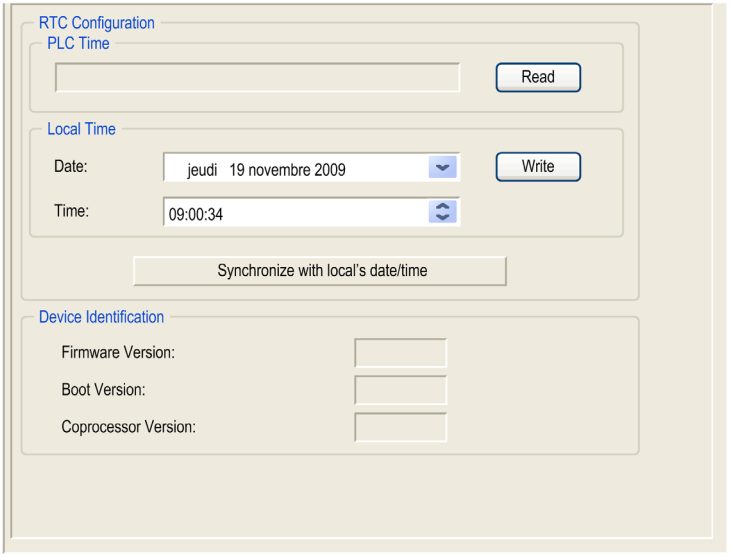
Services

Services Tab

The **Services** tab is divided in 2 parts:

- RTC Configuration
- Device Identification

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.

Chapter 9
M238 Embedded Functions

Overview

This chapter describes the embedded functions of the Modicon M238 Logic Controller.

Each embedded function uses inputs and outputs.

The Modicon M238 Logic Controller with DC power supply has:
- 14 digital inputs, including 8 fast inputs *(see M238 Logic Controller, Hardware Guide)*
- 10 digital outputs, including 4 fast outputs *(see M238 Logic Controller, Hardware Guide)*

The Modicon M238 Logic Controller with AC power supply has:
- 14 digital inputs, including 8 fast inputs *(see M238 Logic Controller, Hardware Guide)*
- 10 digital outputs, including 6 relay outputs *(see M238 Logic Controller, Hardware Guide)*

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
HSC Embedded Function	76
Embedded I/Os Configuration	78
PTO_PWM Embedded Function	82

HSC Embedded Function

Overview

The HSC Embedded 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:

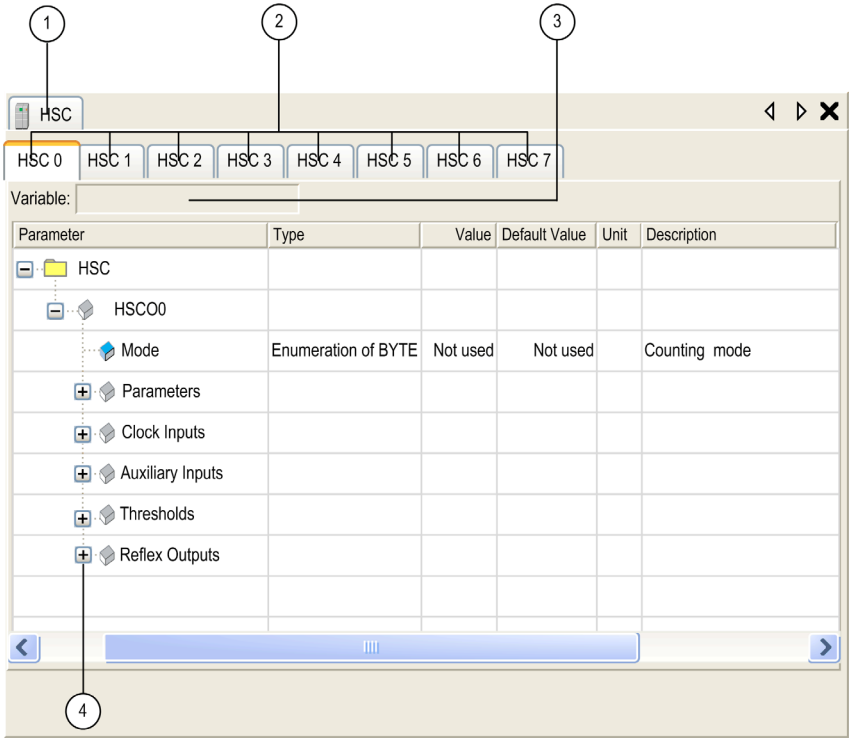
- **Simple** type: a single input counter *(see M238 Logic Controller, Hardware Guide)*.
- **Main** type: a counter that uses up to 4 fast inputs and 2 reflex outputs *(see M238 Logic Controller, Hardware Guide)*.

Accessing the HSC Configuration Window

To access the embedded HSC configuration window, double-click **Embedded Functions → HSC** in the **Devices tree**.

HSC Configuration Window

The following figure shows a sample of the HSC configuration window used to configure the HSC:



The following table describes the areas of the HSC configuration window and the associated actions:

Number	Action
1	Select the **HSC** tab to access each one of the HSC configuration windows.
2	Select a specific HSC tab to access the HSC channel you need to configure.
3	After choosing the type of HSC (**Simple** or **Main**) you want, use the field **Variable** to change the instance.
4	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 M238 HSC choice matrix *(see Modicon M238 Logic Controller, High Speed Counting, M238 HSC Library Guide)*.

Embedded I/Os Configuration

Overview

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

The embedded inputs are composed of 8 fast inputs and 6 standard inputs.

The 8 fast inputs are named I0 to I7 and the 6 standard inputs are named I8 to I13.

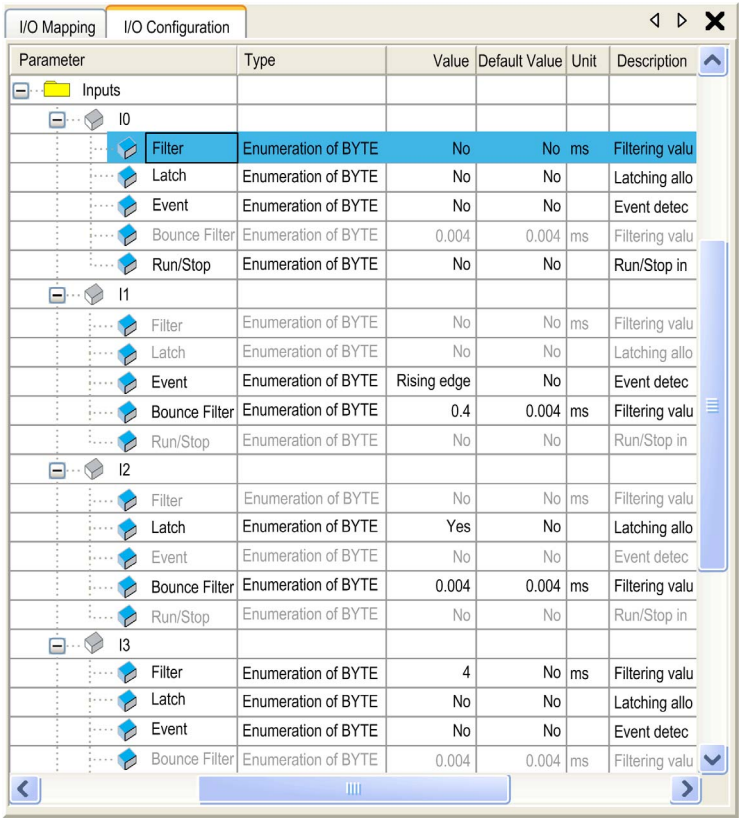
Accessing the I/O Configuration Window

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

Step	Description
1	In the **Devices tree**, double-click **Embedded Functions → IO**.
2	Select the **I/O Configuration** tab.

I/O Configuration Window

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



Parameter	Type	Value	Default Value	Unit	Description
Inputs					
I0					
Filter	Enumeration of BYTE	No	No	ms	Filtering valu
Latch	Enumeration of BYTE	No	No		Latching allo
Event	Enumeration of BYTE	No	No		Event detec
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering valu
Run/Stop	Enumeration of BYTE	No	No		Run/Stop in
I1					
Filter	Enumeration of BYTE	No	No	ms	Filtering valu
Latch	Enumeration of BYTE	No	No		Latching allo
Event	Enumeration of BYTE	Rising edge	No		Event detec
Bounce Filter	Enumeration of BYTE	0.4	0.004	ms	Filtering valu
Run/Stop	Enumeration of BYTE	No	No		Run/Stop in
I2					
Filter	Enumeration of BYTE	No	No	ms	Filtering valu
Latch	Enumeration of BYTE	Yes	No		Latching allo
Event	Enumeration of BYTE	No	No		Event detec
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering valu
Run/Stop	Enumeration of BYTE	No	No		Run/Stop in
I3					
Filter	Enumeration of BYTE	4	No	ms	Filtering valu
Latch	Enumeration of BYTE	No	No		Latching allo
Event	Enumeration of BYTE	No	No		Event detec
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering valu

I/O Mapping Window Description

The **I/O Mapping** window contains the following columns:

Column	Description
Variable	Lets you map the channel on a variable. Double-click the icon to enter the variable name. If it is a new variable, the variable is created. It is also possible to map an existing variable with the variables **Input Assistant** by clicking the **...** button. **NOTE:** New variables are automatically created on each channel according to the "Automatic I/O mapping" project option settings.
Mapping	Indicates if the channel is mapped on a new variable or an existing variable.
Channel	Name of the channel of the device.
Address	Address of the channel.
Type	Data type of the channel.
Default Value	Value taken by the Output when the controller is in a STOPPED state. Double-click to change the default value.
Unit	Unit of the channel value.
Description	Description of the channel.

Digital Input Configuration Parameters

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

Parameter	Value	Description	Constraint
Filter	No* 1.5 ms 4 ms 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 **Run/Stop** disabled.
Event	No* Rising edge Falling edge Both edges	Event detection	This parameter is only available for the fast inputs I0 to I7. Available if: **Latch** disabled AND **Run/Stop** disabled.

Parameter	Value	Description	Constraint
Bounce Filter	0.004 ms* 0.4 ms 1.2 ms 4 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 0.004.
Run/Stop	No* Yes	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.

PTO_PWM Embedded Function

Overview

The PTO embedded function can provide 3 different functions:

- **PTO**:
  The PTO (Pulse Train Output) *(see M238 Logic Controller, Hardware Guide)* implements digital technology that provides precise positioning for open loop control of motor drives.
- **PWM**:
  The PWM (Pulse Width Modulation) *(see M238 Logic Controller, Hardware Guide)* function generates a programmable square wave signal on a dedicated output with adjustable duty cycle and frequency.
- **FG**:
  The FG (Frequency Generator) *(see M238 Logic Controller, Hardware Guide)* function generates a square wave signal on dedicated output channels with a fixed duty cycle (50%).

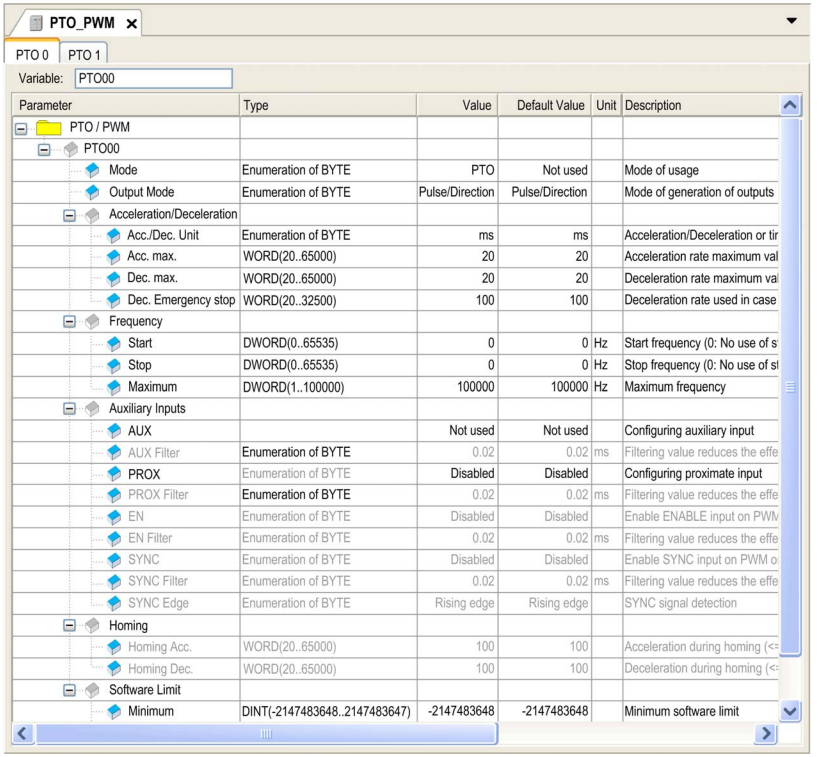
Accessing the PTO_PWM Configuration Tab

Follow these steps to access the PTO_PWM embedded function configuration tab:

Step	Description
1	In the **Devices tree**, double-click **MyController → Embedded Functions → PTO_PWM**.
2	Select the **PTO x** tab corresponding to the channel to be configured.

PTO_PWM Configuration Tab

This following figure shows a sample PTO_PWM configuration window used to configure a PTO, PWM or FG:



Parameter	Type	Value	Default Value	Unit	Description
PTO / PWM					
PTO00					
Mode	Enumeration of BYTE	PTO	Not used		Mode of usage
Output Mode	Enumeration of BYTE	Pulse/Direction	Pulse/Direction		Mode of generation of outputs
Acceleration/Deceleration					
Acc./Dec. Unit	Enumeration of BYTE	ms	ms		Acceleration/Deceleration or tir
Acc. max.	WORD(20..65000)	20	20		Acceleration rate maximum val
Dec. max.	WORD(20..65000)	20	20		Deceleration rate maximum va
Dec. Emergency stop	WORD(20..32500)	100	100		Deceleration rate used in case
Frequency					
Start	DWORD(0..65535)	0	0	Hz	Start frequency (0: No use of s
Stop	DWORD(0..65535)	0	0	Hz	Stop frequency (0: No use of st
Maximum	DWORD(1..100000)	100000	100000	Hz	Maximum frequency
Auxiliary Inputs					
AUX		Not used	Not used		Configuring auxiliary input
AUX Filter	Enumeration of BYTE	0.02	0.02	ms	Filtering value reduces the effe
PROX	Enumeration of BYTE	Disabled	Disabled		Configuring proximate input
PROX Filter	Enumeration of BYTE	0.02	0.02	ms	Filtering value reduces the effe
EN	Enumeration of BYTE	Disabled	Disabled		Enable ENABLE input on PWM
EN Filter	Enumeration of BYTE	0.02	0.02	ms	Filtering value reduces the effe
SYNC	Enumeration of BYTE	Disabled	Disabled		Enable SYNC input on PWM o
SYNC Filter	Enumeration of BYTE	0.02	0.02	ms	Filtering value reduces the effe
SYNC Edge	Enumeration of BYTE	Rising edge	Rising edge		SYNC signal detection
Homing					
Homing Acc.	WORD(20..65000)	100	100		Acceleration during homing (<=
Homing Dec.	WORD(20..65000)	100	100		Deceleration during homing (<=
Software Limit					
Minimum	DINT(-2147483648..2147483647)	-2147483648	-2147483648		Minimum software limit

For further information on configuration parameters, refer to:

- PTO configuration. *(see Modicon M238 Logic Controller, Pulse Train Output, Pulse Width Modulation, M238 PTOPWM Library Guide)*
- PWM and FG configuration. *(see Modicon M238 Logic Controller, Pulse Train Output, Pulse Width Modulation, M238 PTOPWM Library Guide)*

Chapter 10
Expansion Modules Configuration

General Description

Introduction

In your project, you can add the following types of expansion modules to your controller:

- digital
- analog
- expert (e.g. HSC, PTO)

The TM2 expansion bus of the Modicon M238 Logic Controller is created when you assemble the I/O expansion modules to the controller. I/O expansion modules are considered external devices in the controller architecture and are treated, as such, differently than the embedded I/Os that are native to the controller (embedded I/O).

If the Modicon M238 Logic Controller cannot communicate to one or more I/O expansion modules that is contained in the program configuration, the controller considers it an exception, and assumes the controller state of RUNNING with External Error Detected *(see page 62)*.

The unsuccessful communication with one or more I/O modules may be detected during the start-up of the logic controller or during runtime, and there may be any number of causes. Causes of communication exception on the TM2 expansion bus include, among others, disconnection of or physically missing I/O modules, electromagnetic radiation beyond published environmental specifications, or otherwise inoperative modules.

When such an event occurs that disrupts communication on the TM2 expansion bus, the status of the bus is updated, and the digital I/O expansion output modules are set to **Keep Current Values**. However, in the case of RUNNING with an External Error Detected, the controller continues to solve logic and the embedded I/O continues to be Managed by Application Program *(see page 53)*.

To re-establish regular communication on the TM2 expansion bus, you must resolve the reason for the unsuccessful communication, and then cycle power on the logic controller system.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Include in your risk assessment the possibility of unsuccessful communication between the logic controller and any I/O expansion modules.
- If the “Keep Current Values” deployed during an I/O expansion External Error is incompatible with your application, use alternate means to control your application for such an event.

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

Use the `GetRightBusStatus` *(see Modicon M238 Logic Controller, System Functions and Variables, M238 PLCSystem Library Guide)* function regularly to monitor the expansion bus status.

TM2 Expansion Module Configuration

For more information about module configuration, refer to the programming and hardware guides of each expansion module type:

Expansion Module	Programming Guide	Hardware Guide
TM2 Digital I/O Modules	TM2 I/O Modules Configuration Programming Guide	TM2 Digital I/O Modules Hardware Guide *(see Modicon TM2, Digital I/O Modules, Hardware Guide)*
TM2 Analog I/O Modules	TM2 I/O Modules Configuration Programming Guide	TM2 Analog I/O Modules Hardware Guide *(see Modicon TM2, Analog I/O Modules, Hardware Guide)*
TM2 High-Speed Counting Modules	TM2 I/O Modules Configuration Programming Guide	TM2 High Speed Counter Modules Hardware Guide *(see Modicon TM2, High Speed Counter Modules, Hardware Guide)*
AS-Interface Communication Module	Modicon M238 Logic Controller Programming Guide *(see page 92)*	AS-Interface Master Communication Module Hardware Guide *(see Modicon TWDNOI10M3, AS-Interface Master Module, Hardware Guide)*

Maximum Hardware Configuration

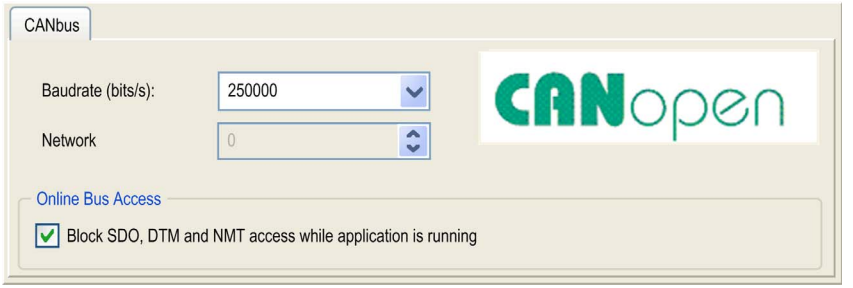
Up to 7 expansion modules can be added to the controller.

Chapter 11
CANopen Configuration

CANopen Interface Configuration

CAN Bus Configuration

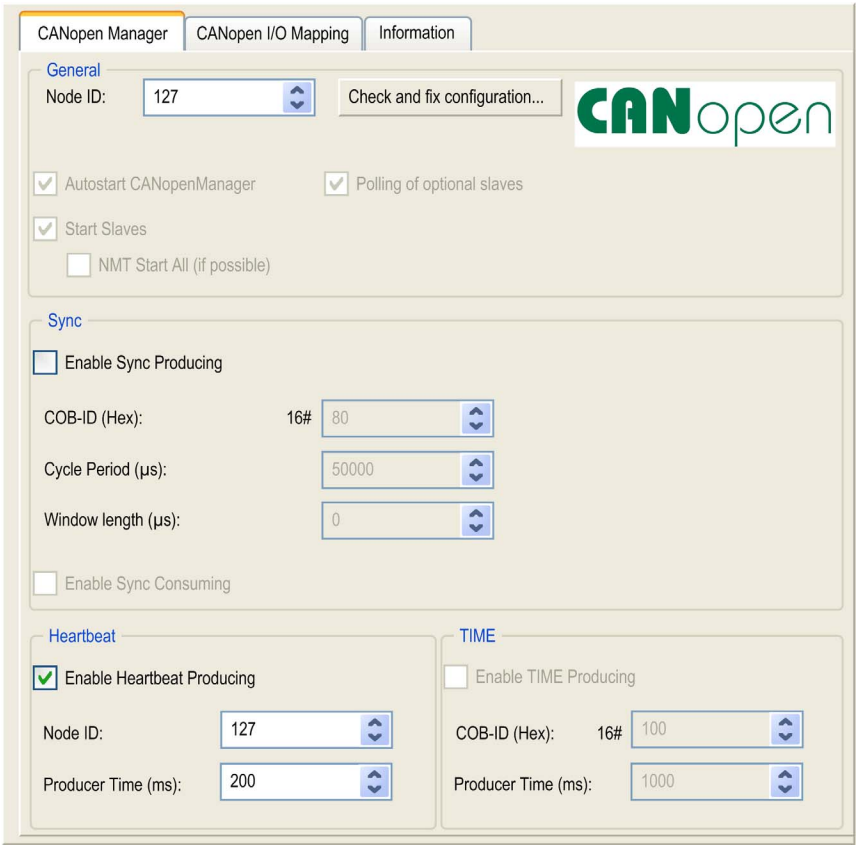
To configure the **CAN** bus of your controller, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **CAN**.
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.

When connecting a DTM to a device using the network, the DTM communicates in parallel with the running application. The overall performance of the system is impacted and can cause overload on the network.

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	Click the **Plus Button** next to the **CAN**x node in the **Devices Tree**. In the **Add Device** window, select **CANopen Performance** and click the **Add Device** button. 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)*
2	Double-click **CANopen_Optimized**. **Result:** The **CANopen Manager** configuration window appears: 

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 M238 Logic Controller CANopen master has the following operating limits:

Maximum number of slave devices	16
Maximum number of Received PDO (RPDO)	32
Maximum number of Transmitted PDO (TPDO)	32

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Do not connect more than 16 CANopen slave devices to the controller.
- Program your application to use 32 or fewer Transmit PDO (TPDO).
- Program your application to use 32 or fewer Receive PDO (RPDO).

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

Chapter 12
AS-Interface Configuration

Overview

This chapter explains how to configure and use the AS-Interface Master Module, and the module limitations.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Presentation of the AS-Interface V2 Fieldbus	92
General Functional Description	93
Software Setup Principles	96
Add an AS-Interface Master Module	97
Configure an AS-Interface Master	98
Add an AS-Interface Slave	100
Configure an AS-Interface Slave	103
Automatic Addressing of an AS-Interface V2 Slave	106
Modification of Slave Address	107
System Diagnostic in Online Mode	108
Programming for the AS-Interface V2 Fieldbus	112
Configuration of a Replaced AS-Interface V2 Slave	113

Presentation of the AS-Interface V2 Fieldbus

Introduction

The AS-Interface Fieldbus (Actuator Sensor-Interface) allows the interconnection on a single cable of sensor devices and actuators, with the lowest level of automation.

These sensors/actuators will be defined in this documentation as **slave devices**.

NOTE: For more information about the TWDNOI10M3 expansion module, refer to the TWDNOI10M3 Communication Module Hardware Guide

NOTE: All terms and definitions used in this chapter and throughout the document concerning AS-Interface are those as defined by the AS-Interface Association Specification version 2.11.

AS-Interface V2 Fieldbus

The AS-Interface Master module TWDNOI10M3 expansion module includes the following functionality:

- M3 profile: This profile includes all the functionality defined by the AS-Interface V2 standard
- One AS-Interface channel per module
- Automatic addressing for the slave with the physical address set to 0
- Management of profiles and parameters
- Protection from polarity inversion on the bus inputs

The AS-Interface Fieldbus then allows:

- Up to 31 standard address or 62 extended address slaves
- Up to 248 inputs and 186 outputs
- Up to 8 analog slaves (maximum of four analog channels per slave)
- A cycle time of 10 ms maximum

A maximum of 2 TWDNOI10M3 expansion modules can be connected to a M238.

General Functional Description

General Introduction

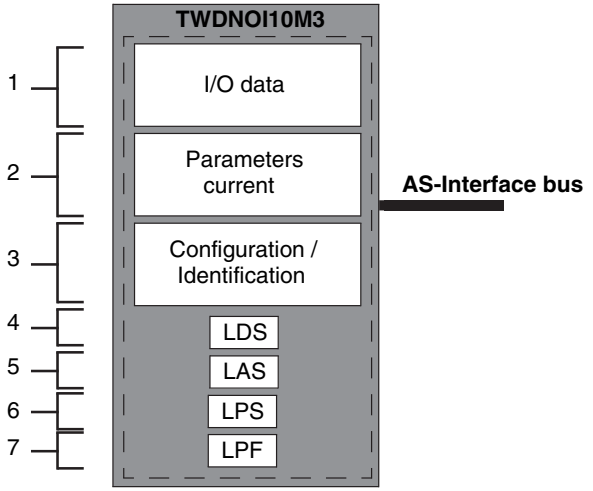
For the AS-Interface configuration, SoMachine software allows the user to:

- Manually configure the bus (declaration of slaves and assignment of addresses on the bus)
- Automatically configure the bus (by using the **Scan Network** and **Copy to project** command)
- Adapt the configuration according to what is present on the bus
- Acknowledge the slave parameters
- Control bus status

AS-Interface Master Structure

The AS-Interface module includes data fields that allow you to manage the lists of slaves and the images of input / output data.

The figure below shows **TWDNOI10M3** module architecture.



The following table describes the data field stored in volatile memory:

Address	Item	Description
1	I/O data (IDI, ODI)	Input/Output Data Image Images of 248 inputs and 186 outputs of AS-Interface V2 Fieldbus, configured in SoMachine and detected on the bus.
2	Current parameters (PI, PP)	Parameter Image / Permanent Parameter. Image of the parameters of all the slaves.

Address	Item	Description
3	Configuration / Identification (CDI, PCD)	This field contains all the I/O codes and the identification codes for all the slaves detected.
4	LDS	List of Detected Slaves. List of all slaves detected on the Fieldbus.
5	LAS	List of Active Slaves. List of slaves activated on the Fieldbus.
6	LPS	List of Projected Slaves. List of slaves configured with SoMachine.
7	LPF	List of Peripheral Faults. List of slaves determined to have generated peripheral errors.

Structure of Slave Devices

The standard address slaves each have:

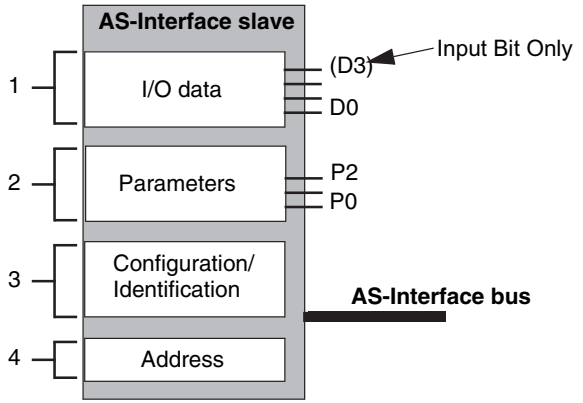
- 4 input/output bits
- 4 parameter bits

The slaves with extended addresses each have:

- 4 input/output bits (the last bit is reserved for inputs only)
- 3 parameter bits

Each slave has its own address, profile and sub-profile (defines variables exchange).

The figure below shows the structure of an extended address slave:



The following table describes the data of the structure:

Address	Item	Description
1	Input/output data	Input data is stored by the slave and made available for the AS-Interface master. Output data is updated by the master module.
2	Parameters	The parameters are used to control and switch internal operating modes to the sensor or the actuator.
3	Configuration/ Identification	This field contains: ● the I/O configuration code (IO code) ● the slave identification code (ID code) ● the slave extended identification codes (ID1 and ID2 codes)
4	Address	Physical address of slave.
Note: The operating parameters, address, configuration and identification data are saved in a non-volatile memory.		

Software Setup Principles

At a Glance

To respect the operating principles adopted in SoMachine software, the user should adopt a step-by-step approach for creating an AS-Interface application.

Setup Principle

The following table shows the software implementation phases of the AS-Interface Fieldbus.

Mode	Phase	Description
Logged out	Declaration of module *(see page 97)*	Choice of the slot for the AS-Interface Master module TWDNOI10M3 on the expansion bus.
	Declaration of slave devices *(see page 108)*	Selection for each device: ● of its address on the bus ● of its profile
Logged out or logged in	Programming *(see page 112)*	Programming diagnostic functions with the **IoDrvASI** *(see page 192)* library.
Logged in	Transfer	Transfer of the application to the controller.
	Diagnostic / Debugging *(see page 108)*	Debugging the application with the help of: the SoMachine interface to display slaves (address, profile), and to assign them the desired addresses.

NOTE: The declaration and deletion of the AS-Interface Master module on the expansion bus is the same as for other expansion modules. However, once two AS-Interface Master modules have been declared on the expansion bus, SoMachine will not permit any other AS-Interface Master modules to be declared.

Considerations Prior to Connection

You must ensure that all slaves have a unique address. In addition, the address of 0 is reserved for automatic addressing mode. If there is a slave with an address of 0 detected on the bus at start-up, the master will change to the offline phase and try to restart. You must ensure that all addresses are unique and that none are 0.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Verify that each slave has a unique address greater than 0.

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

Add an AS-Interface Master Module

Introduction

This section shows you how to add a TWDNOI10M3 module to a Modicon M238 Logic Controller configuration.

Add a TWDNOI10M3 Master Module

To add an AS-Interface Master module to your controller, select the **TWDNOI10M3** 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)*

Configure an AS-Interface Master

Introduction

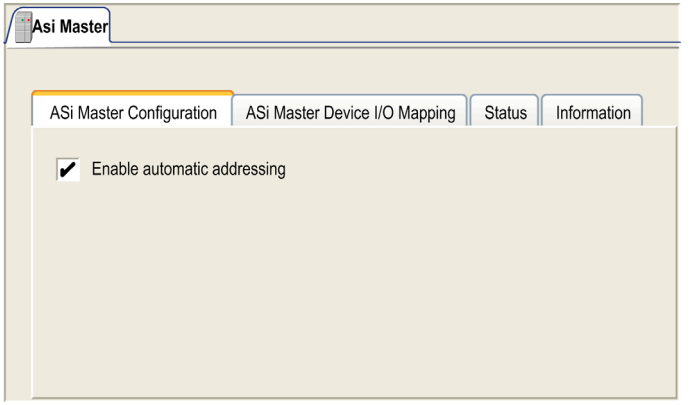
This section shows you how to configure an AS-Interface Master.

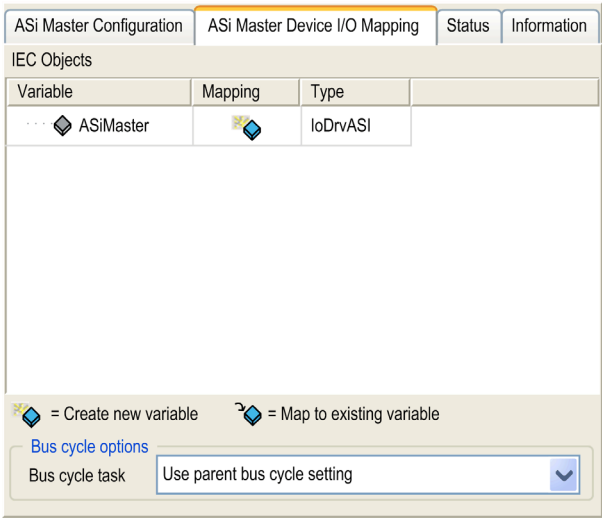
Access the Configuration Window

To access the configuration window, double-click **ASiMaster** in the **Devices tree**.

Description of the Configuration Window when Logged Out

The configuration window of the AS-Interface Master gives you access to the Automatic addressing parameters.

Tab Name	Configuration Window Description
ASi Master Configuration	 **Enable automatic addressing** (selected by default): Activate this option to enable automatic addressing. For more information, refer to Automatic Addressing of an AS-Interface V2 Slave *(see page 106)*.

Tab Name	Configuration Window Description
ASi Master Device I/O Mapping	 This configuration window contains the following fields: • IEC Objects • Bus cycle options For more information about I/O mapping, refer to the SoMachine Programming Guide.
Status	This tab provides status information (for example **Running**, **Stopped**) and device-specific diagnostic messages.
Information	If available for the current module, the following general information will be displayed: Name, Vendor, Type, Version Number, Categories, Order Number, Description, Image.

Add an AS-Interface Slave

Introduction

This section shows you how to add 1 or more slave devices to a TWDNOI10M3 module.

There are 3 methods to add a slave device to an AS-Interface Master module:

- **Catalog**: when using Schneider Electric devices.
- **Generic Slave**: when using third-party devices.
- **Scan for Devices**: quickly and easily configure an already existing bus.

NOTE: You may use steps from each of these methods during configuration.

Add a Slave Device using Schneider Electric catalog

The SoMachine catalog lists the Schneider Electric AS-Interface slave devices by their reference name.

NOTE: The profile of each slave device is pre-configured and cannot be modified.

To add a Slave Device to your controller, select the AS-Interface slave you want 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)*

Add a Generic Slave Device

If you want to manually configure your AS-Interface Slave device, you can add a generic AS-Interface Slave and configure its profile.

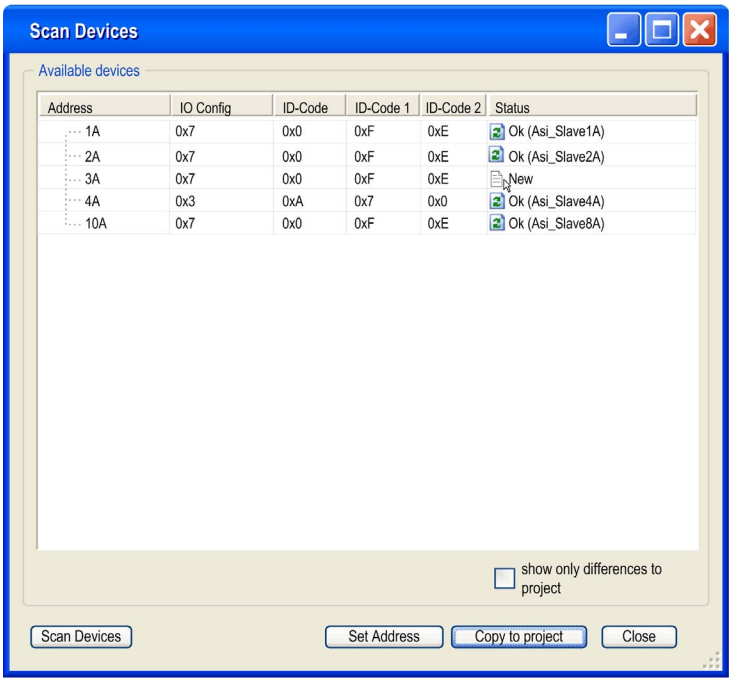
The procedure shows you how to add generic slave devices to your project:

Step	Action
1	Select the **0/Generic ASi Slave** 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)*
2	Double-click **0/Generic ASi Slave** to configure your ASI Slave *(see page 103)*.

Add a Slave Device with the Scan For Devices Command

The **Scan For Devices** command will search all AS-Interface Slave devices connected to the TWDNOI10M3 master module. This function requires that the master module is configured before executing the **Scan For Devices** command.

The procedure below shows you how to add slave devices with the SoMachine software **Scan For Devices** command:

Step	Action
1	Log in to your Controller. **NOTE:** Only the right-bus configuration including your master module must be correctly set up for this step. No application program is needed.
2	In the **Devices tree**, right-click the **ASiMaster** module and select **Scan For Devices**.
3	Slave devices detected on the Fieldbus are displayed with their address and profile.  Click on the **Scan Devices** button to refresh the list of slaves.

Step	Action
4	Activate the checkbox **show only differences to project**. This will display only the mismatching devices (physical versus configured). The **Status** column can accept the following values: • **OK**: If address and profile are matching. • **Configuration Mismatch**: if there is a matching address and a mismatching profile. • **New**: a slave is detected on the Fieldbus but there is no slave device at this address in the configuration.
5	If necessary, modify the addresses under the column **Address** of the **Scan Devices** window. Click the **Set Address** button to readdress the selected slave with a new address *(see page 107)*.
6	Click on the **Copy to project** button. The **Copy to project** function allows you to copy a slave detected on the network to the project Device tree. You can select several slaves using SHIFT+click, then use the **Copy to project** button to copy all selected devices to the project Device tree. Slaves with the same address will be overwritten Your project is now updated with all connected slave devices under the Device tree. You need to download the application again to make these changes operational.
7	If you want to add another AS-Interface slave, connect it to your Fieldbus and run a new scan (Step 3).

Configure an AS-Interface Slave

Introduction

This document shows you how to configure a slave connected to the TWDNOI10M3 module.

Access the AS-Interface Slave Configuration Window

To access the configuration window, double-click the AS-Interface Slave device in the **Devices tree**. The AS-Interface Slave configuration window is displayed.

Description of the AS-Interface Slave Configuration Window

The AS-Interface Slave Configuration tab provides access to all relevant slave configuration data: address, profile and parameters. For devices from the catalog, profile information is greyed out, and not available for modification.

Every slave must have a unique address. It can have any value from 1A to 31A and 1B to 31B (B addresses are only allowed with extended addressing slaves). In total, no more than 62 slaves are allowed. The slave profile determines if standard or extended addressing is available. For some slaves, more than one address is needed.

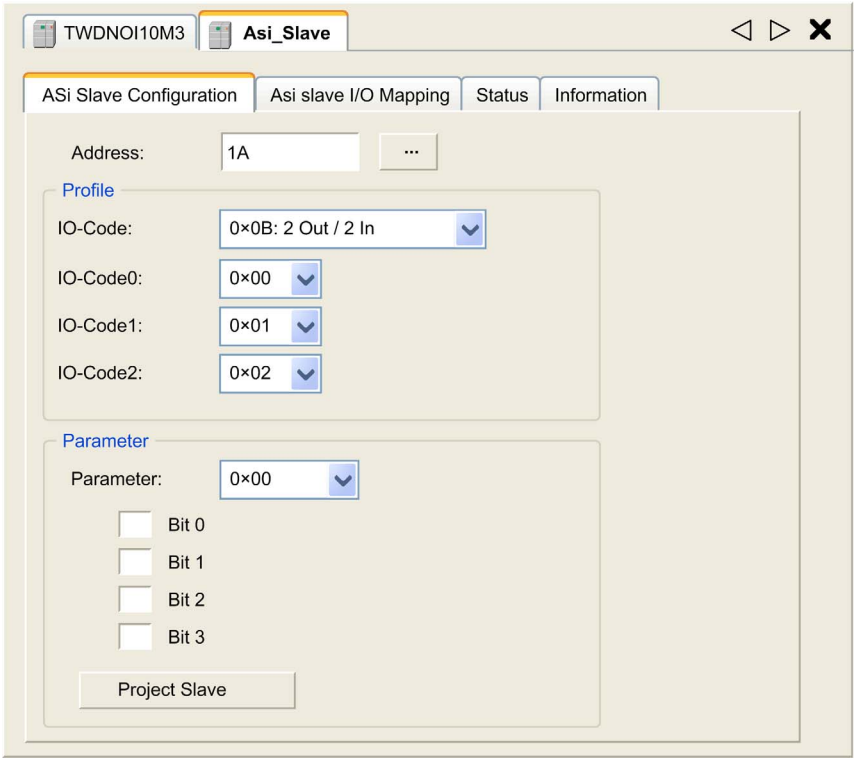
⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Verify that each slave has a unique address greater than 0.

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

The configuration window is shown below:



The table below describes the **ASi Slave Configuration** window fields:

Field Name	Field Description
Address	In this field the slave address has to be set. Use the Browser Button **[...]** to open a choice of available AS-Interface addresses not yet used by slave configurations in the project.
Profile	Use this selection list to configure the AS-Interface Slave profile: ● **IO-Code:** Defines the I/O configuration of the slave. There are16 different I/O configuration modes available from 00 hex (4 Inputs) to 0F hex (Tristate). ● **Id Code0..2:** Used for further distinction of slaves with the same I/O configuration.
Parameter	Use either the selection list or the check boxes to set the configured parameters (**AS-interface Permanent Parameters**) of the slave. The slave profile defines if parameters are being used and, if so, the meaning of each parameter.

Field Name	Field Description
Project Slave	Clicking on the **Project Slave** button sends the parameter bits to the slave (when logged in). To change slave parameters without downloading the entire application, set the new parameters and then press the **Project Slave** button. The new parameters will be written to the Parameter Image table. **NOTE:** Slave parameters changes through the Project Slave button are only written into the slave. The changes are not written in the controller current application, and will be overwritten by a reset or reboot. ● If you wish your changes to persist through a reset, update the current application by selecting Download in the Online menu. ● If you wish your changes to persist through a reboot, the Boot application must also be updated.

NOTE: The profile and parameters of a slave are not associated with a name. Several slaves with different names can have the same profile and parameters.

Description of the ASi Slave I/O Mapping Window

The **ASi Slave I/O Mapping** tab allows you assign project variables to the AS-Interface outputs or inputs.

NOTE: For more information about these fields, refer to the SoMachine online help, chapter Programming with SoMachine / Device Editors.

Automatic Addressing of an AS-Interface V2 Slave

At a Glance

Each slave on the AS-Interface Fieldbus must be assigned (via configuration) a unique physical address. This must be the same as the one declared in SoMachine.

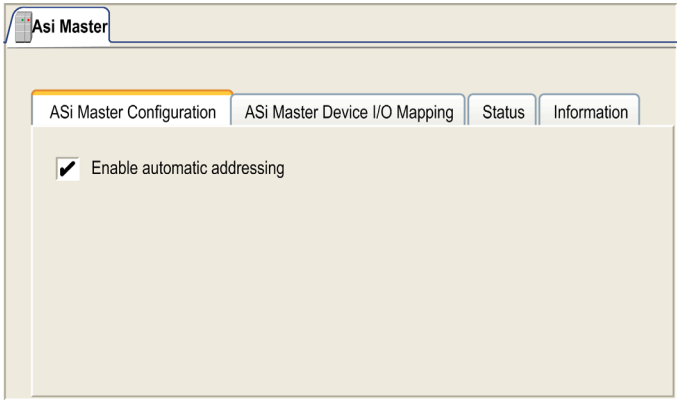
The AS-interface Automatic addressing function is supported by the master, allowing you to:

- replace a slave indicating an error
- insert a new slave

The new slave with physical address 0 will be automatically written with the address of a missing or unresponsive slave, if their profile and parameters match.

Procedure

The table below lists the steps required to set the **Automatic addressing** parameter.

Step	Action
1	In the **Devices tree**, open the **TWDNOI10M3** node and double-click **ASiMaster**.
2	Click on the **Enable automatic addressing** check box (if not already selected) found in the **ASi Master Configuration** tab:  **Result**: The **Enable automatic addressing** function is activated (box checked) or disabled (box not checked). **NOTE:** By default, the **Automatic addressing** parameter is selected in the configuration window.

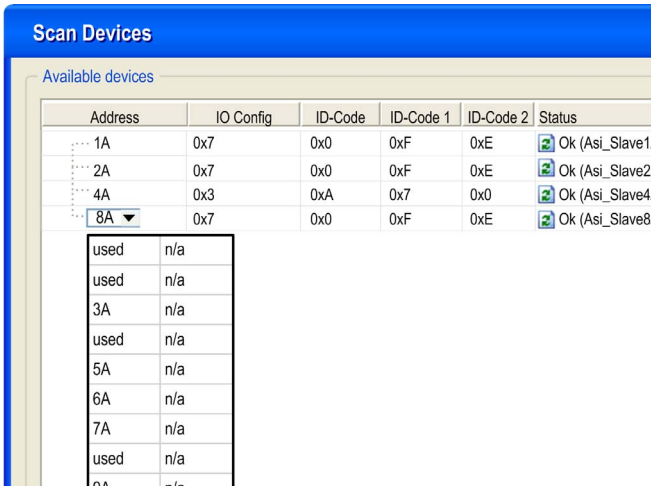
Modification of Slave Address

At a Glance

From the Scan Devices interface, you can modify the address of a slave.

Modification of Slave Address

The following table shows the steps required to modify a slave address:

Step	Description
1	Log in to your Controller. **NOTE:** Only the right-bus configuration from your master module must be correctly set up at this step. No application program is needed.
2	In the **Devices tree**, right-click on the **ASiMaster** module and select **Scan For Devices**.
3	In the **Scan Devices** window, select an available slave address under the list box.  The addresses already used by another slave are noted as **used** under the list box. **NOTE:** Address 0 is not proposed in the drop-down list because a slave should not be changed to an address of 0 normally (0 being used for fast device replacement). However, it is possible to do so manually by writing the value 0 in the address field. If Automatic addressing is enabled, the slave set to 0 address will be immediately reassigned to another address.
4	The new address is displayed in blue until you click on the **Set Address** button to confirm. Click on **Scan Devices** to refresh the window and see the modification (new address is shown in black).
5	Exit the **Scan Devices** window (click on the **Close** button).

System Diagnostic in Online Mode

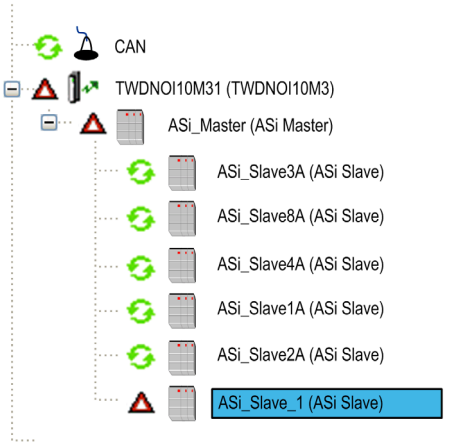
Introduction

The SoMachine interface dynamically provides an image of the physical bus when the controller containing the user application is connected to the PC. This image includes:

- Status of the AS-Interface master module and the configured slave devices (in the **Devices tree** tab, and in **Status** tab of each device editor)
- Image of the detected slaves on the bus (Scan for devices) *(see page 101)*.

Diagnostic in Devices Window

In the **Devices tree** tab, you can obtain a quick overview of the AS-Interface Slaves status:

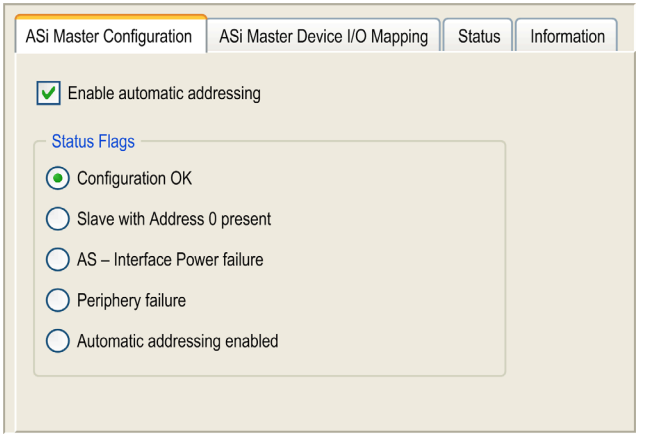


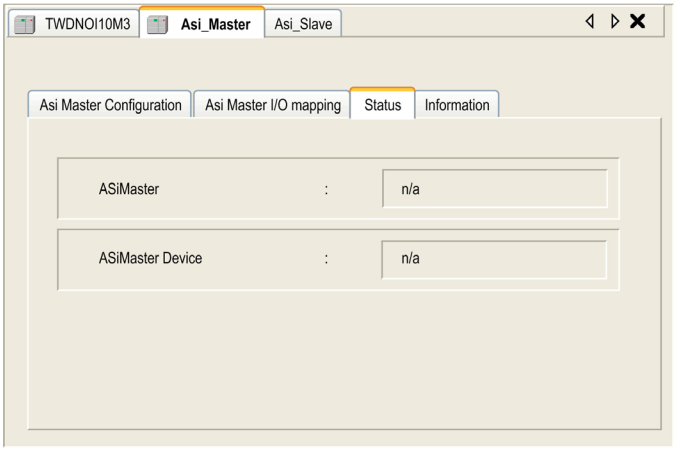
The status of each slave is indicated by an icon:

- **Green icon**: parameters are OK. Device is operational.
- **Red icon**: detected error in the device configuration. To get more information, go to the **Status** tab of the device editor.

Diagnostic of AS-Interface Master Device when Logged In

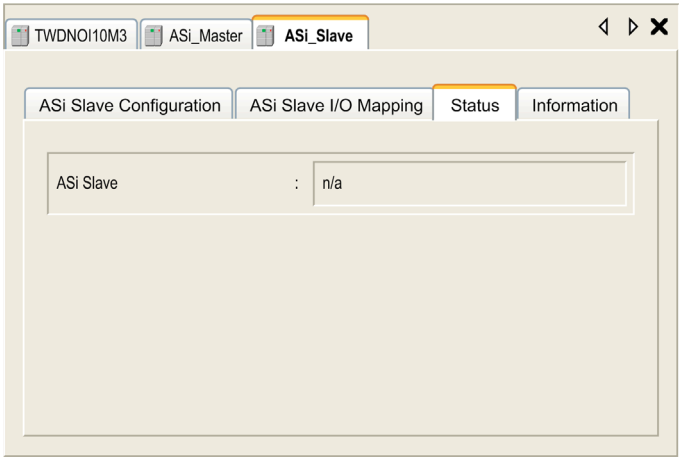
The Master device configuration window when logged in is shown in the following tables:

Tab Name	Description
ASi Master Configuration	When you are logged in, a new field named **Status Flags** appears:  **Enable automatic addressing:** Activate this option to enable automatic addressing. For more information, refer to Automatic Addressing of an AS-Interface V2 Slave *(see page 106)*. The **Status Flags** section shows the current state of the master: ● **Config OK:** Target and actual configuration match. ● **Slave with Address 0 present:** The master module has detected one slave module with the address 0. This address is typical of a new slave module with factory configuration. ● **AS-Interface Power failure:** AS-Interface system power is low. Verify your AS-interface power supply. ● **Periphery failure:** A periphery error has been detected. Read LPF (List of Peripheral Fault) to search for the affected device(s). ● **Automatic addressing enabled:** The automatic addressing function is enabled.

Tab Name	Description
Status	This tab of the **Configuration Editor** provides information about the status of an AS-Interface Master device.  The fields of this tab show status information (for example **n/a**, **Running**, **Stopped**).

Diagnostic of AS-Interface Slave Device in Online Mode

The slave device configuration window is shown below:

Tab Name	Description
Status	This tab of the **Configuration Editor** provides information about the status of an AS-Interface Slave device:  The fields of this tab show status information (for example **n/a**, **Running**, **Stopped**).

Programming for the AS-Interface V2 Fieldbus

Introduction to the IoDrvASI Library

The **IoDrvASI** *(see page 191)* library includes a **Function** and some **Function Blocks** that allow you to operate the AS-Interface Fieldbus in the application.

This library is automatically included in your SoMachine when you add a TWDNOI10M3 expansion module.

Function

The **IoDrvASI** library includes the following function:

Function Name	Description
ASI_CheckSlaveBit *(see page 192)*	Checks if a bit at a certain offset within the provided array of AS-Interface status bytes (e.g. LDS, LAS, LPF) is set. This function is used to extract the information for 1 slave from `ASI_SlaveStatusCheck` function block output data. Returns true if bit is set, otherwise returns false.

Function Blocks

The **IoDrvASI** library includes the following function blocks:

Function Block Name	Description
ASI_CmdSetAutoAddressing *(see page 194)*	Activate/Deactivate the master device with the automatic addressing mode.
ASI_CmdSetDataExchange *(see page 196)*	Enable data exchange between master and slave devices.
ASI_CmdSetOfflineMode *(see page 198)*	Set the bus in offline mode.
ASI_MasterStatusCheck *(see page 200)*	Provide master flags, which indicate the state of the master.
ASI_SlaveAddressChange *(see page 202)*	Replace current slave address by a new user-determined address.
ASI_SlaveParameterUpdate *(see page 205)*	Update the image of the slave device.
ASI_SlaveStatusCheck *(see page 207)*	Provides information about slave devices (LAS, LDS, LPF).
ASI_ReadParameterImage *(see page 209)*	Read or refresh the parameter image table.

Configuration of a Replaced AS-Interface V2 Slave

Automatic Configuration

When a slave must be replaced, it can be automatically replaced with a slave with the same AS-Interface profile.

This happens without the AS-Interface V2 Fieldbus having to stop, and without requiring any manipulation if the configuration mode's **Automatic addressing** utility is active *(see page 106)*.

The replacement slave must initial have the address 0 (a new slave is usually factory set with a default address of 0), and the same profile as the slave it will replace. It will automatically assume the address of the replaced slave once installed, and will then be inserted into the list of detected slaves (LDS) and the list of active slaves (LAS).

Manual Configuration

Alternative options without automatic addressing are available:

- You can configure the replacement slave with the same address as the slave it will replace using the pocket programmer. As previously noted, the replacement must have the same product reference number and the same profile and sub-profile of the slave to replace. It is thus automatically inserted into the list of detected slaves (LDS) and into the list of active slaves (LAS). This feature is available only if one, and not more than one, slave is inoperative.
- Change the address using the Scan For Devices window *(see page 100)*.

Chapter 13
Serial Line Configuration

Introduction

This chapter describes how to configue the serial line communication of the Modicon M238 Logic Controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Serial Lines Configuration	116
ASCII Manager	118
SoMachine Network Manager	120
Modbus IOScanner	121
Modbus Manager	129
Adding a Modem to a Manager	134

Serial Lines Configuration

Introduction

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

The Serial Line port(s) of your controller is (are) configured for the SoMachine protocol by default when new or when you update the controller firmware. The SoMachine protocol is incompatible with other protocols such as that of Modbus Serial Line.

In an active Modbus configured serial line, if a new controller is connected or if a controller firmware is updated, this can cause the others devices available on the serial line to stop communicating.

Verify 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.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Verify that your application has the Serial Line port(s) properly configured for Modbus before physically connecting the controller to an operational Modbus serial line network.

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

Serial Line Configuration for TM238LDD24DT and TM238LDA24DR

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

Result: The **Serial Line** configuration window is displayed.

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

Element	Description
Baud rate	Transmission speed
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 polarization resistor or not) • RS232 **NOTE:** Two line polarization resistors are integrated in the controller; they are switched on or off by this parameter.

Serial Line Configuration for TM238LFDC24DT and TM238LFAC24DR

To configure the Serial Line 1 and Serial Line 2, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **Serial Line 1**. **Result:** The **Serial Line 1** configuration window is displayed.
2	In the **Devices tree**, double-click **Serial Line 2**. **Result:** The **Serial Line 2** configuration window is displayed.

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

Element	Description
Baud rate	Transmission speed
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: • SL1: select RS485 (using polarization resistor or not) or RS232 • SL2: only RS485 is available

ASCII Manager

Introduction

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

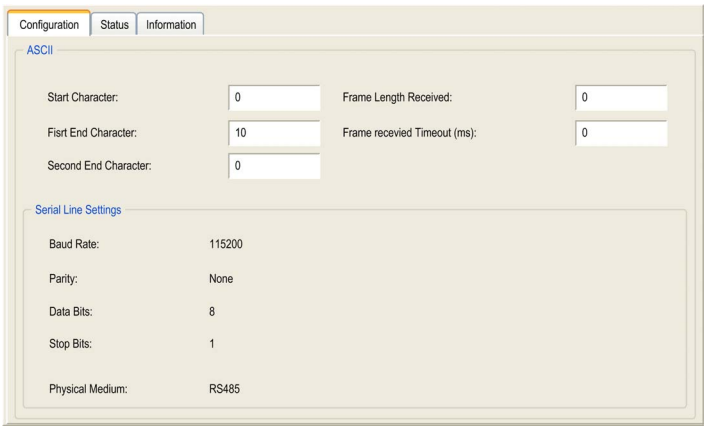
Adding the ASCII_Manager

To add an ASCII_Manager on Serial Line, proceed as follows:

Step	Action
1	In the **Devices tree**, delete **SoMachine_Network_Manager** or **Modbus_Manager** from the **Serial Line** node if any.
2	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)*

Configure the ASCII_Manager

To configure the ASCII Manager of your controller, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **ASCII_Manager**. **Result:** The **ASCII_Manager** configuration window is displayed. 

Set the parameters as described in the following 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 116)*.

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

Adding a Modem

For more details about how to add a Modem to the ASCII Manager, refer to the Adding Modem to a Manager section *(see page 134)*.

SoMachine Network Manager

Introduction

The SoMachine Network Manager must be used if you want to exchange variables with a XBTGT/XBTGK device using the SoMachine software protocol, or when the Serial Line is used for SoMachine programming.

Adding the Manager

To add a **SoMachine-Network_Manager** to your project, 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)*

Configure the Manager

There is no configuration for SoMachine Network Manager.

Adding a Modem

For more details about how to add a Modem to the **SoMachine-Network_Manager**, refer to the Adding Modem to a Manager section *(see page 134)*.

Modbus IOScanner

Introduction

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

Adding the Manager

To add a Manager on Serial Line, proceed as follows:

Step	Action
1	In the **Devices tree**, delete **SoMachine_Network_Manager** or **Modbus_Manager** from the **Serial Line x** node if any.
2	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)*

Configuring the Manager

To configure a Modbus IOScanner on Serial Line, proceed as follows:

Step	Action
1	In the **Devices tree**, double-click **Modbus IOScanner**. **Result:** The configuration window is displayed.

Set the parameters as described in the following 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 a ASCII format, LRC error-checking (7 data bits) This parameter must be identical for each Modbus device on the link.
Response Timeout (ms)	Timeout used in the exchanges.
Time between frames (ms)	Time to avoid bus-collision. This parameter must be identical for each Modbus device on the link.

NOTE: Do not use function blocks of the PLCCommunication library on a serial line with a Modbus IOScanner configured. This disrupts the Modbus IOScanner exchange.

Adding a Device on the Modbus IOScanner

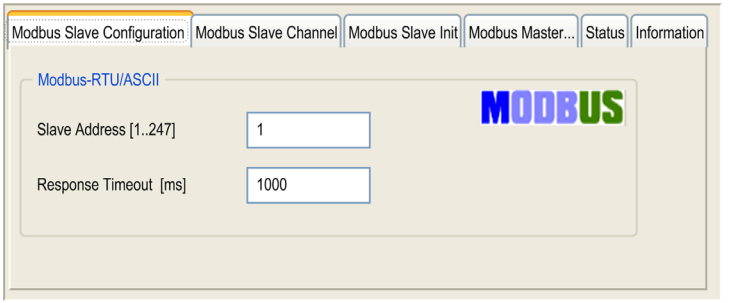
To add a device on the **Modbus IOScanner**, select the **Generic Modbus Slave** in the **Hardware Catalog** , drag it to the **Devices tree**, and drop it on the **Modbus_IOScanner**.

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

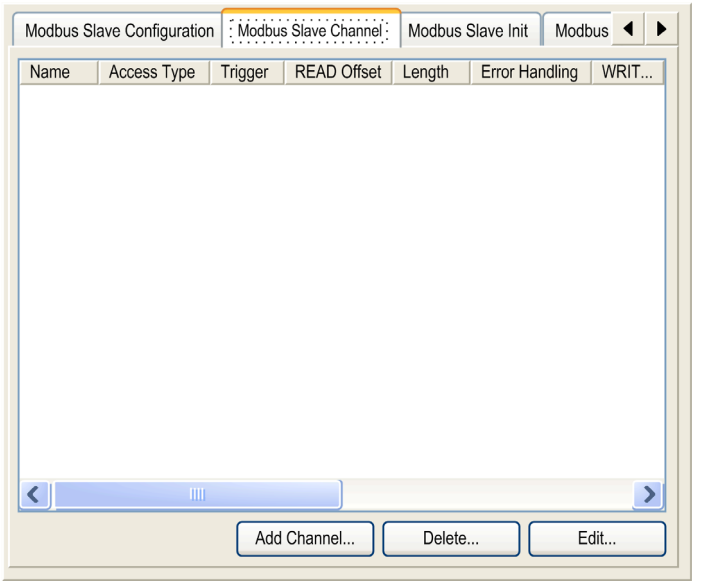
- Using the Drag-and-drop Method *(see SoMachine, Programming Guide)*
- Using the Contextual Menu or Plus Button *(see SoMachine, Programming Guide)*

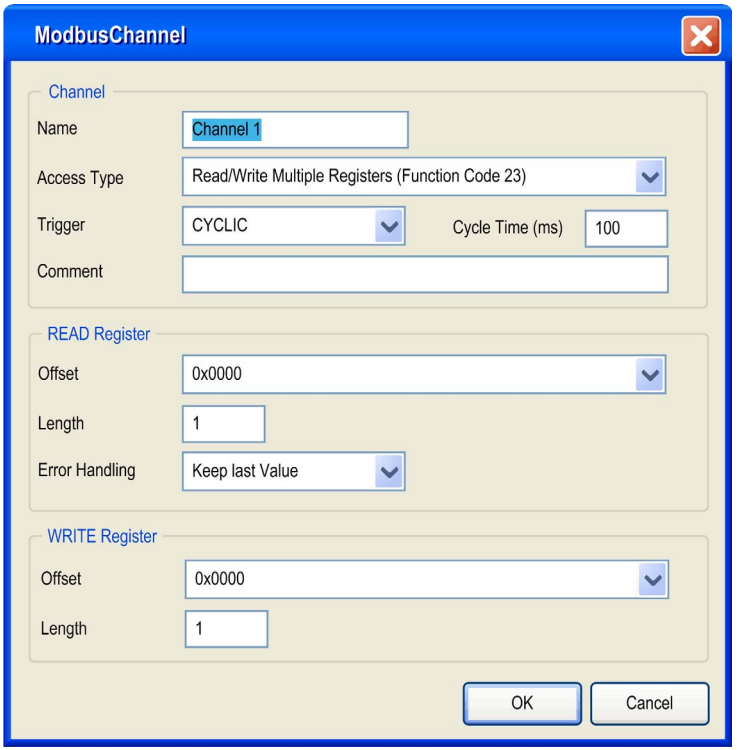
Configuring a Device Added on the Modbus IOScanner

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

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

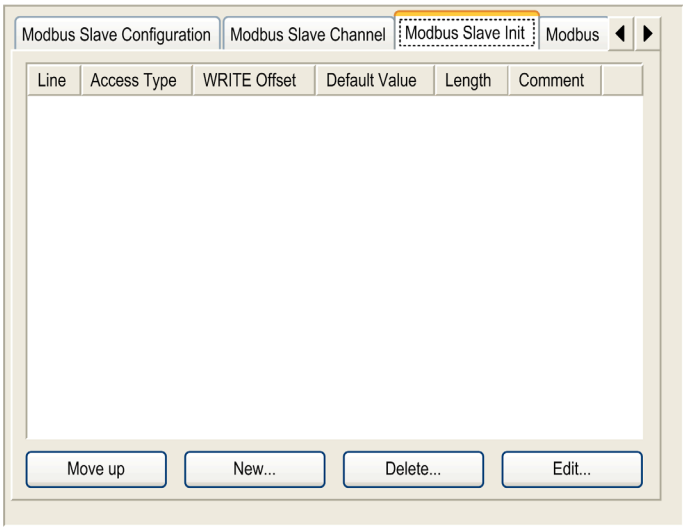
To configure the **Modbus Channels**, proceed as follows:

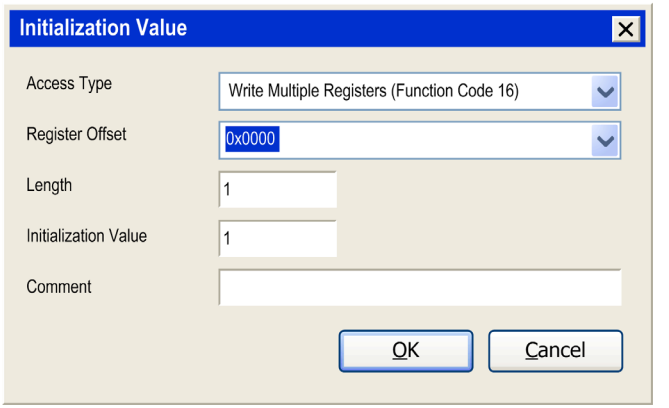
Step	Action
1	Click the **Modbus Slave Channel** tab. 

Step	Action
2	To configure an exchange, click the **Add Channel** button:  In the field **Channel**, you can add the following values: • **Name**: Enter a name for your channel • **Access Type**: Read or Write or Read/Write multiple registers. • **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

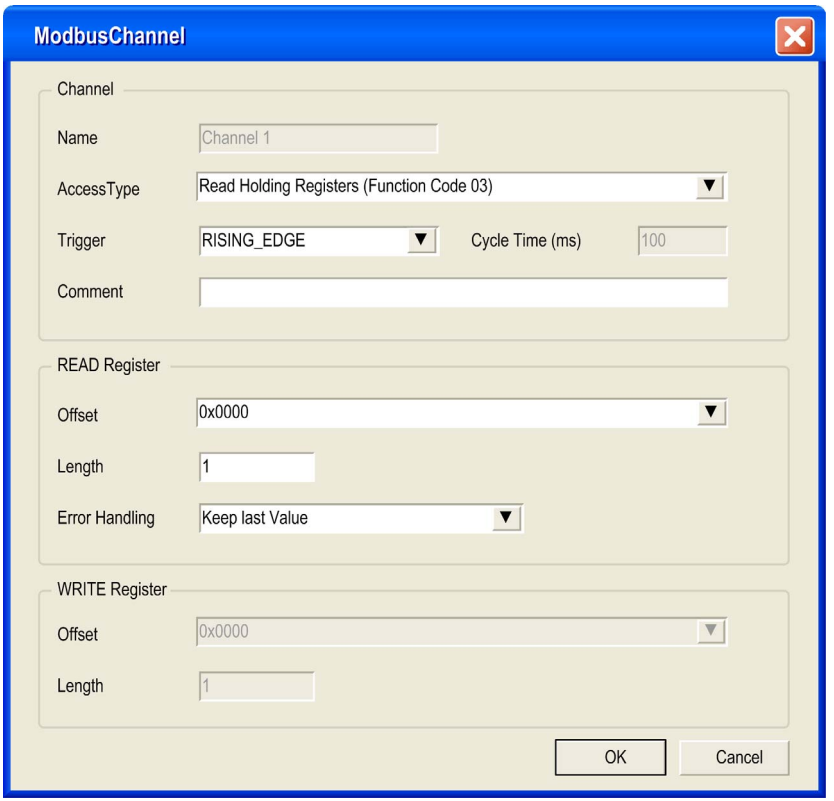
Step	Action
2 bis	In the field **READ Register** (if your channel is a Read or a Read/Write one), you can configure the %MW to be read on the Modbus slave. Those addresses will be mapped on %IW (see **Modbus Master I/O Mapping** tab): ● **Offset**: Offset of the %MW to read. 0 means that the first object 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 a Write or a Read/Write one), you can configure the %MW to be written to the Modbus slave. Those addresses will be mapped on %QW (see 'Modbus Master I/O Mapping' tab): ● **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.
3	Click the **Delete** button to remove a channel. Click the **Edit** button to change the parameters of a channel.
4	Click **OK** to validate the configuration of this channel.

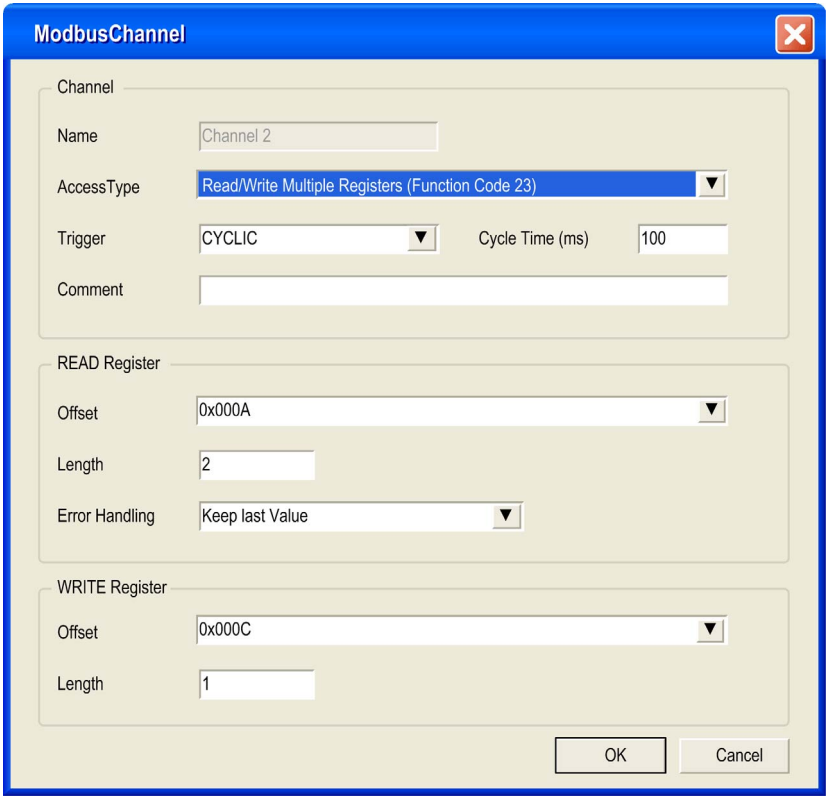
To configure your **Modbus Initialization Value**, proceed as follows:

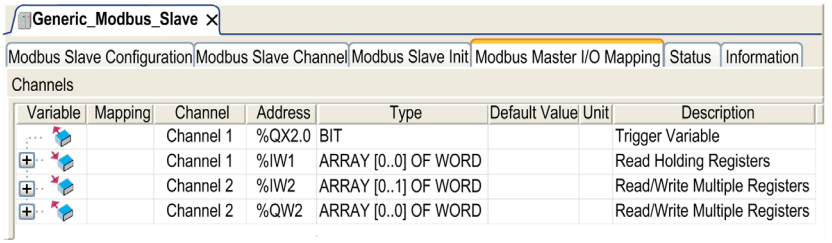
Step	Action
1	Click the **Modbus Slave Init** tab: 

Step	Action
2	Click the button **New** to create a new initialization value:  The **Initialization Value** window contains the following parameters: • **Access Type**: Only Write Multiple Register is allowed • **Register Offset**: Offset of the %MW that will be initialized • **Length**: Number of %MW that will be initialized. For example if **Offset** = 2 and **Length** = 3, %MW2, %MW3 and %MW4 wil be initialized • **Initialization Value**: Value the registers are initialized with • **Comment**: Add a comment about this initialization
3	Click the **Move up** or **Move down** button to change the position of an initialization in the list. Click the **Delete** button to remove an initialization in the list. Click the **Edit** button to change the parameters of an initialization.
4	Click **OK** to create a new **Initialization Value**.

These next set of interface windows show the mapping of the objects generated by the defined channels. If channel 1 and channel 2 are configured as pictured in the following two graphics, then the Modbus Master I/O Mapping is as pictured in the third graphic below:




ModbusChannel
Channel
Name
Channel 2
AccessType
Read/Write Multiple Registers (Function Code 23)
Trigger
CYCLIC
Cycle Time (ms)
100
Comment
READ Register
Offset
0x000A
Length
2
Error Handling
Keep last Value
WRITE Register
Offset
0x000C
Length
1
OK
Cancel


Generic_Modbus_Slave
Modbus Slave Configuration
Modbus Slave Channel
Modbus Slave Init
Modbus Master I/O Mapping
Status
Information
Channels
Variable
Mapping
Channel
Address
Type
Default Value
Unit
Description
Channel 1
%QX2.0
BIT
Trigger Variable
Channel 1
%IW1
ARRAY [0..0] OF WORD
Read Holding Registers
Channel 2
%IW2
ARRAY [0..1] OF WORD
Read/Write Multiple Registers
Channel 2
%QW2
ARRAY [0..0] OF WORD
Read/Write Multiple Registers

Modbus Manager

Introduction

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

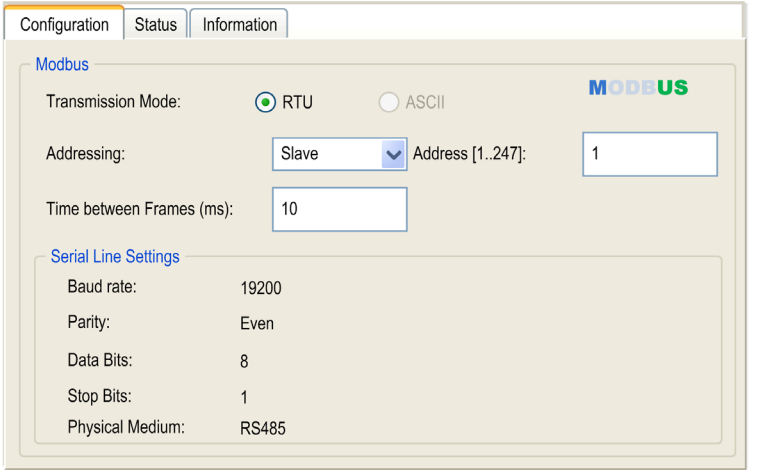
Adding the Modbus Manager

To add a Modbus Manager on Serial Line, proceed as follows:

Step	Action
1	In the **Devices tree**, delete **SoMachine_Network_Manager** from the **Serial Line** node if any.
2	To add an expansion module 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)*

Configure the Modbus Manager

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

Step	Action
1	In the **Devices tree**, double-click **Modbus_Manager**. **Result:** The **Modbus_Manager** configuration window will be displayed. 

Set the parameters as described in the following 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 a ASCII format, LRC error-checking (7 data bits). This parameter must be identical for each Modbus device on the link.
Addressing	Specify if the M238 device is master or slave.
Address	Modbus address of the device.
Time between frames (ms)	Time to avoid bus-collision. This parameter must be identical for each Modbus device on the link.
Serial Line Settings	Parameters specified in the Serial Line configuration *(see page 116)* 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 *(see SoMachine, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)*.

Modbus Slave

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

Types		Function	Function Codes Code/Sub Code
Data Access (1 Bit)	**Physical Discrete Inputs and Outputs**	Read Coils	01
		Read Discrete Inputs	02
		Write Multiple Coils	15
Data Access (16 Bits)	**Physical Input Registers**	Read Holding Registers	03
		Write Single Register	06
		Write Multiple Registers	16
		Read/Write Multiple Registers	23
Diagnostics		Diagnostics	08
		Read Device Identification	43/14

NOTE: Only located variables of the controller application can be accessed via Modbus.

The following table contains the Sub-function codes supported by the diagnostic Modbus request 08:

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

The table below list 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 e.g. TM238LFDC24DT
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 and it will be possible for incorrect values to be written to memory areas responsible for output operations.

⚠ WARNING

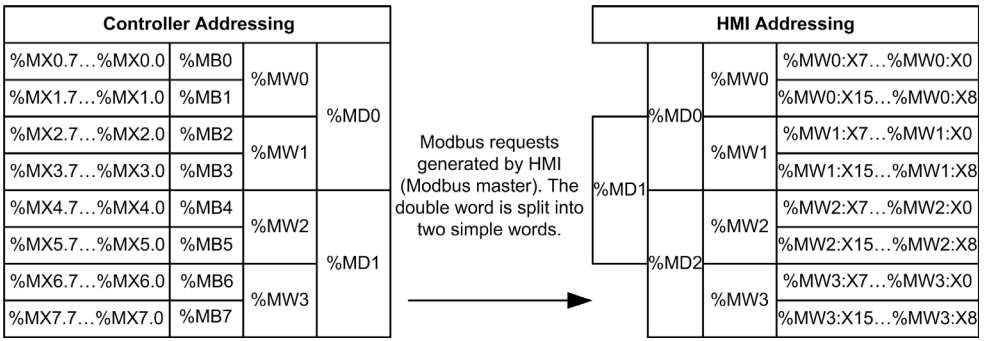
UNINTENDED EQUIPMENT OPERATION

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

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

When the controller and the 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

For more details about how to add a Modem to the Modbus Manager, refer to the Adding Modem to a Manager section *(see page 134)*.

Adding the Ethernet Gateway

For more details about how to add a 499TWD01100, refer to the Declaring the Ethernet Gateway section *(see page 137)*.

Adding a Modem to a Manager

Introduction

A modem can be added to the following managers:

- ASCII Manager
- Modbus Manager
- SoMachine Network Manager

Adding a Modem to the 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 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)*

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

Chapter 14
499TWD01100 Ethernet/Modbus Gateway

Connection and Configuration of the Ethernet Gateway

Overview

Configure the Ethernet Gateway module with the following instructions.

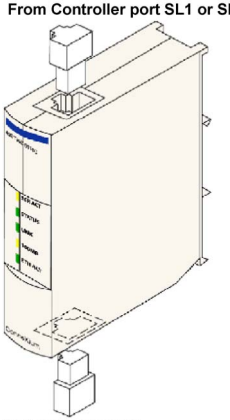
For more details about the Ethernet Gateway, refer to the 499TWD01100 Ethernet/Modbus Gateway for M238 Hardware Guide *(see 499TWD01100, Ethernet/Modbus Gateway for M238, Hardware Guide)*.

NOTE: When the Ethernet Gateway module is configured with the SoMachine programming software, the module IP configuration is stored in the controller. Therefore, maintenance personnel can exchange the gateway module without additional configuration.

Connecting the 499TWD01100 Ethernet Gateway Module

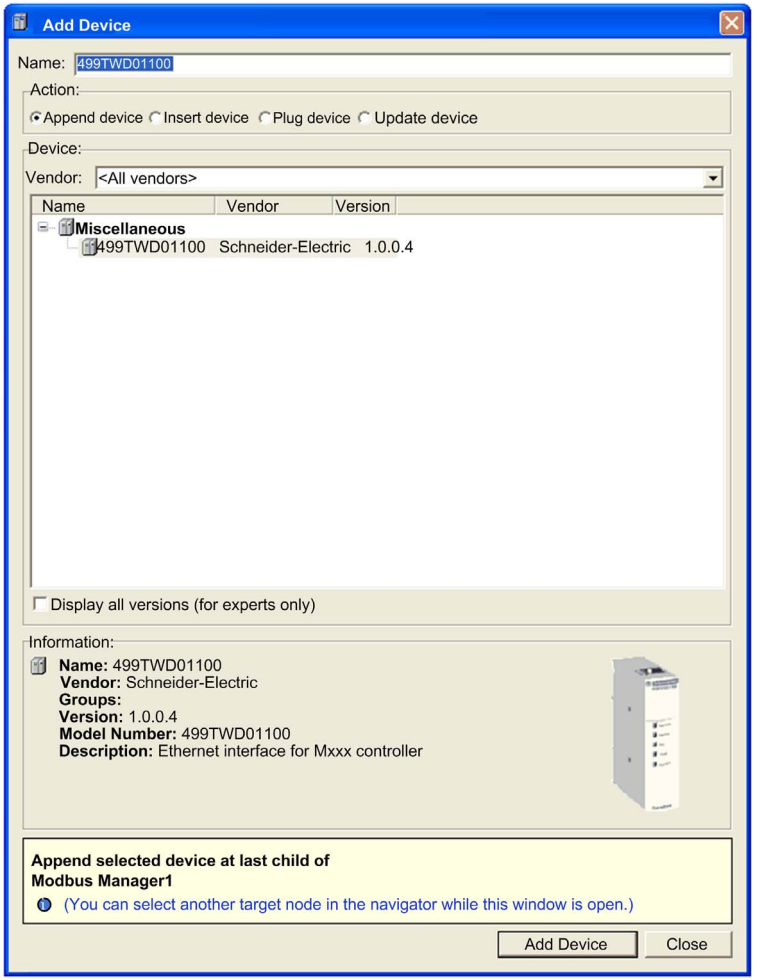
To install the Ethernet gateway on a controller, follow these steps:

Step	Description	Action
1	Preparation	Consult the 499TWD01100 Ethernet/Modbus Gateway for M238 Hardware Guide *(see 499TWD01100, Ethernet/Modbus Gateway for M238, Hardware Guide)* to have more information about how to: ● know the mounting positions for the module, ● add and remove the module from a DIN rail, ● mount the module on a panel surface, ● respect the minimum clearances for the module in a control panel.
2	Mounting the 499TWD01100 Module	Install the module on a DIN rail or panel.
3	Protective Ground	Attach a grounded wire to the M3 screw terminal on the bottom of the gateway module.

Step	Description	Action
4	Serial and Ethernet Connections **Top connector:** **From Controller port SL1 or SL2**  **Bottom connector:** **From Ethernet, either straight or crossover cable**	Connect the gateway-to-controller XBT Z9980 cable (supplied) to the serial port of the Ethernet Gateway, and connect the other end to the appropriate serial port of the Controller: ● SL1 for TM238LDD24DT & TM238LDA24DR ● SL2 for TM238LFDC24DT & TM238LFAC24DR Connect the RJ45 connector from a standard Ethernet network cable (not supplied) into the Ethernet port of the Gateway.

Declaring the 499TWD01100 Ethernet Gateway Module

The table below shows the different steps when declaring the 499TWD01100 gateway module.

Step	Action	Comment
1	Right click the **Modbus_Manager** of Serial Line and select **Add Device**.	**Modbus_Manager** of SL1 for TM238LDD24DT & TM238LDA24DR **Modbus_Manager** of SL2 for TM238LFDC24DT & TM238LFAC24DR
2	Select 499TWD01100 Ethernet Module in the device list.	

Step	Action	Comment
3	Click **Add Device**.	A 499TWD01100 node is created in the **Devices** window.
4	Double click the 499TWD01100 node to access the configuration window.	-

Configuring the 499TWD01100 Ethernet Gateway Module

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.

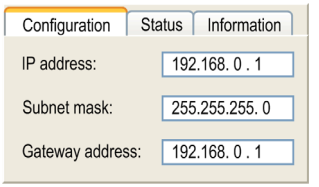
⚠ WARNING

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.

To configure Ethernet parameters, follow this procedure:

Step	Action	Comment
1	Double click the 499TWD01100 node to access the configuration window.	The **Ethernet Configuration** dialog box appears, as shown in the example below. 
2	Enter the static IP Address for the gateway in dotted decimal notation. (See *notes 1 and 2*.)	-

Step	Action	Comment
3	Enter the valid **Subnet mask** assigned to the gateway by your network administrator. Please note that you cannot leave this field blank; you must enter a value. (See *notes 1 and 3*.)	By default, the programming software automatically computes and displays a default subnet mask based on the network class that you have provided in the IP Address field above. Default subnet mask values, according to the category of the gateway network IP address, follow these rules: Class A network -> Default subnet mask: 255.0.0.0 Class B network -> Default subnet mask: 255.255.0.0 Class C network -> Default subnet mask: 255.255.255.0
4	Enter the IP address of the **Gateway**. (See *notes 1 and 4*.)	On the LAN, the gateway must be on the same segment as the 499TWD01100. This information typically is provided to you by your network administrator. Note that no default value is provided by the application, and that a valid gateway address is required in this field.
5	Verify and validate your configuration.	-
6	Power off the Controller, then power on again.	A power cycle is required to force the M238 to transfer the IP address to the 499TWD01100.

NOTE:

1. Consult your network or system administrator to obtain valid IP parameters for your network.
2. Each connected device on an Ethernet network segment must have a unique IP address. When connected to the network, the gateway runs a check for duplicate IP address. If a duplicate IP address is located over the network, the STATUS LED will emit 4 flashes periodically. You must then enter a new duplicate-free IP address in this field.
3. Unless the gateway has special subnet requirements, use the default subnet mask.
4. If there is no gateway device on your network, simply enter the gateway's IP address in the Gateway Address field.

Chapter 15
Connecting the Modicon M238 Logic Controller to a PC

Connecting the Controller to a PC

Overview

To transfer, run, and monitor the applications, use a USB cable to connect the controller to a computer that has SoMachine installed.

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 Mini-B Port Connection

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

BMXXCAUSBH045: Grounded and shielded, this USB cable is suitable for long duration connections.

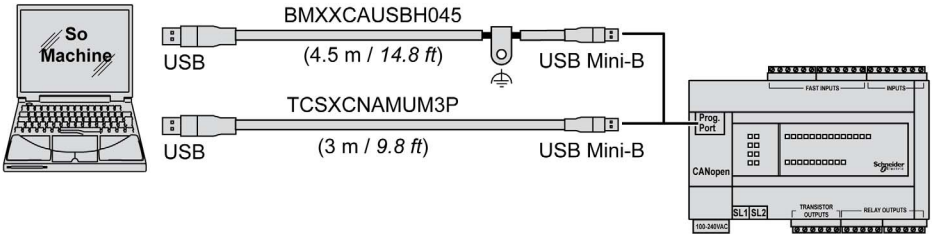
NOTE: You can only connect 1 controller or any other device associated with SoMachine and its component to the PC at any one time.

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.

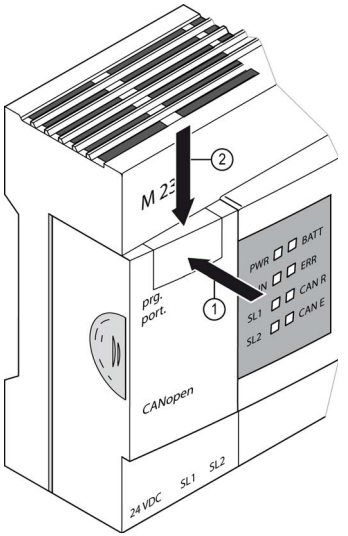
⚠ WARNING
UNINTENDED EQUIPMENT OPERATION OR INOPERABLE EQUIPMENT ● You must use a shielded USB cable such as a BMX XCAUSBH0•• secured to the functional ground (FE) of the system for any long-term connection. ● Do not connect more than one controller at a time using USB connections. **Failure to follow 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.

The following illustration shows the USB connection to a PC:



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 BMXXCAUSBH045, 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	Open the USB cover to access the Programming Port:  **1** Push horizontally on the USB cover and hold down. **2** Slide the USB cover downward.
3	Connect the USB cable connector to the PC.
4	Connect the Mini connector of your USB cable to the M238 USB connector.

Chapter 16
Loader Device Accessory

Overview

This manual describes how to use the Loader Device Accessory

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
16.1	About the Loader Device Accessory	144
16.2	Transfer From SoMachine to the USB Memory Key	152
16.3	File Transfer with a USB Memory Key	153
16.4	Other Functionalities	159

Section 16.1
About the Loader Device Accessory

What Is in This Section?

This section contains the following topics:

Topic	Page
Loader Device Accessory Description	145
Physical Description	147
LED Status and Diagnostic	148
Firmware and SoMachine Software Compatibility	150

Loader Device Accessory Description

Overview

The Loader Device Accessory is an accessory that allows you to download or upload a project through the controller programming port using a USB memory key.

The device can only be used with applications from SoMachine V2.0 or greater.

NOTE:
The USB memory key is a standard USB memory key with the following characteristics:

- 1 GB minimum
- USB 2.0 or less
- formatted in FAT16 or FAT32

Which Files are Transferred?

The Loader Device Accessory allows to transfer the following file types:

- a `*.app` file and
- a `*.crc` file

These file types are transferred as pairs. The number of file pairs that are transferred depends on the SoMachine project:

If...	Then ...	SoMachine Default File Names
the SoMachine project does not contain a **Symbol configuration** or if the project is built with a SoMachine version superior to V3.0	one pair of files is transferred.	• `application.app` • `application.crc`
the SoMachine project contains a **Symbol configuration** built with a SoMachine version inferior to V3.1	2 pairs of files are transferred.	• `application.app` • `application.crc` and • `application_symbol.app` • `application_symbol.crc`

Functionalities

The Loader Device Accessory can be used in the following cases:

- Transfer from SoMachine to the USB memory key *(see page 152)*
- Upload from the controller to the USB memory key *(see page 154)*
- Download from the USB memory key to the controller *(see page 156)*
- Set the controller to RUNNING state *(see page 160)*
- Change the firmware of the Loader Device Accessory *(see page 161)*

⚠ WARNING

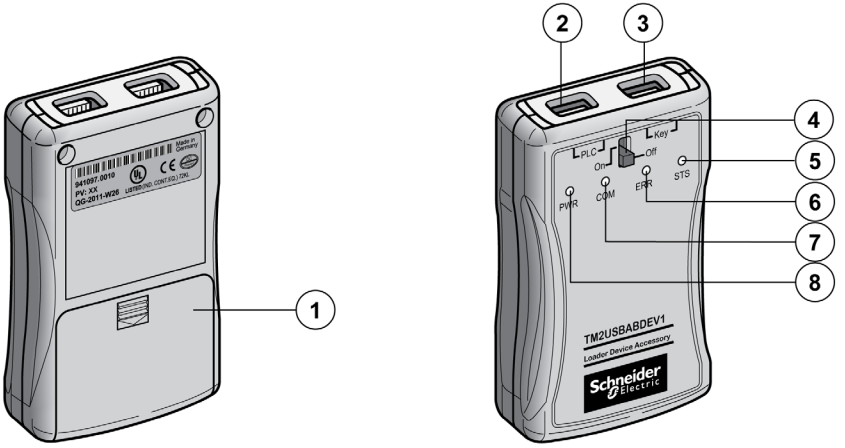
UNINTENDED EQUIPMENT OPERATION

- You must have operational knowledge of your machine or process before connecting this device to your controller.
- Verify that your machine or process is in a know safe state (STOP, SHUTDOWN, INERT, etc.) before transitioning the Loader Device Accessory to the “On” switch position while connected to your controller.
- Verify that guards are in place so that any potential unintended equipment operation will not cause injury to personnel or damage to equipment.
- You must have read and understood the user documentation of this and other devices involved in the functions performed by the Loader Device Accessory.

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

Physical Description

Overview



1 Batteries
2 USB port for the Controller USB-mini B programming port
3 USB port for the USB memory key
4 ON / OFF switch
5 Status LED
6 ERR LED
7 COM LED
8 Power LED

Battery

The Loader Device Accessory use 2 AA / LR6 alkaline batteries.

⚠ WARNING

INOPERABLE EQUIPMENT

- Store the batteries in a cool, well ventilated area.
- Prolonged short circuit causes the battery to lose energy.
- Do not try to recharge the batteries provided with the device.
- Do not install the batteries backwards.
- Immediately discard batteries that show any sign of leakage or damage.
- Discard used batteries in accordance with local regulations.
- Do not mix with other battery types.

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

LED Status and Diagnostic

LED Status

The following table describes the status of the Loader Device Accessory depending on the LED states:

PWR		COM	ERR	STS	Status	Possible Causes
●		○	○	○	Stand by	–
●	◔[1]	◔	○	◔	Normal communication	–
●	◔[1]	○	○	◕	Operation completed	–
●	◔[1]	○	◔	○	Operation error detected	This error is indicated when: ● the USB memory key is full or damaged (file corrupted), ● the communication between the controller and the Loader Device Accessory has been interrupted. ● the controller could not run or stop.
●	◔[1]	○	◔	○	Dialog error detected	This error is indicated when there is a communication error. For example, when: ● the USB memory key and the controller USB ports are swapped (wrong port used), ● USB memory key cannot be read or written (write protection active).
●	◔[1]	◔	◔	○	File error detected	This error is indicated when the files are not found. For example, when: ● the USB memory key may be defective or is otherwise incompatible with the Loader Device Accessory, ● the USB memory key or the controller is not connected or not formatted, ● the USB memory key already has more than the expected number of files, ● the controller has not been rebooted since the last connection with SoMachine.

● LED on ○ LED off ◕ LED 2 s flashing ◕ LED 1 s flashing ◔ LED 0.5 s flashing ◔ LED 0.1 s flashing

[1] Battery low

PWR		COM	ERR	STS	Status	Possible Causes
●	☢[1]	☢	☢	○	Compatibility error detected	This error is indicated when the controller reference and the reference stored in the `*.app` file of the USB memory key are not compatible *(see page 150)*.
☢		☢	○	○	Battery low	–
☢		☢	☢	○	Replace battery	–
☢		○	☢	○	USB error detected	This error is indicated when a wrong USB memory key version is used *(see page 145)*.

● LED on ○ LED off ☢ LED 2 s flashing ☢ LED 1 s flashing ☢ LED 0.5 s flashing ☢ LED 0.1 s flashing

[1] Battery low

Firmware and SoMachine Software Compatibility

Firmware Compatibility Rule

Logic controller firmware versions are made of 4 digits: X.Y.Z.T.

NOTE: The controller version of your application can be found with SoMachine. See the programming guide of your particular controller.

To successfully transfer an application to a controller, the firmware must be compatible.

A compatible firmware must follow these rules:

- The controller reference must be the same
- X.Y digits must be identical
- The Z digit from the controller must be greater or equal to the Z digit from the application.
- T digit is irrelevant.

SoMachine Software Compatibility

The device can only be used with applications from SoMachine V2.0 or greater.

In order to have compatibility with versions of SoMachine prior to version 3.1, it is necessary to perform additional steps while creating the necessary files for download to a controller with a USB memory key.

When using SoMachine version 3.0 or earlier, or a project created with, or a context of, a version 3.0 or earlier, follow the steps below:

Step	Action
1	Open or create the project you want to transfer with the USB memory key using SoMachine.
2	Connect your USB memory key to your computer. **NOTE:** The USB memory key must be empty.
3	Select in the menu **Build → Clean All.**
4	Select in the menu **Build → Build All.**
5	Select in the menu **Online → Create Boot Application**. **NOTE:** SoMachine must be running in offline mode to be able to execute this command.
6	Define your project name and save it in the root of the USB memory key. **NOTE:** Save the files in the root of the USB memory key. Otherwise they will not be detected by the Loader Device Accessory.

You must execute the **Clean All**, **Build All** sequence (steps 3 and 4 in the table above) in order that your Retain Data are properly initialized in your project.

⚠ WARNING
UNINTENDED MACHINE OPERATION Be sure to execute the **Clean All**, **Build All** sequence described above if using a version of SoMachine inferior to version 3.1, or a project originally created with, or in a context of, a version inferior to version 3.1. **Failure to follow these instructions can result in death, serious injury, or equipment damage.**

Section 16.2
Transfer From SoMachine to the USB Memory Key

Transfer From SoMachine to the USB Memory Key

Procedure

The following procedure describes how you can create the necessary files for download to a controller with a USB memory key.

Step	Action
1	Open or create the project you want to transfer with the USB memory key using SoMachine.
2	Connect your USB memory key to your computer. **NOTE:** The USB memory key must be empty.
3	Select in the menu **Online → Create Boot Application**. **NOTE:** SoMachine must be running in offline mode to be able to execute this command.
4	Define your project name and save it in the root of the USB memory key. **NOTE:** Save the files in the root of the USB memory key. Otherwise they will not be detected by the Loader Device Accessory.

NOTE: Only save one project on the USB memory key.

NOTE: If your project was created with a version of SoMachine inferior to version 3.1, or created in a context of a version less than 3.1, refer to the SoMachine software compatibility *(see page 150)* for important information.

Section 16.3
File Transfer with a USB Memory Key

What Is in This Section?

This section contains the following topics:

Topic	Page
Upload From the Controller to the USB Memory Key	154
Download From the USB Memory Key to the Controller	156

Upload From the Controller to the USB Memory Key

Procedure

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- You must have operational knowledge of your machine or process before connecting this device to your controller.
- Be sure that your machine or process is in a know safe state (STOP, SHUTDOWN, INERT, etc.) before transitioning the Loader Device Accessory to the “On” switch position while connected to your controller.
- Be sure that guards are in place so that any potential unintended equipment operation will not cause injury to personnel or damage to equipment.
- You must have read and understood the user documentation of this and other devices involved in the functions performed by the Loader Device Accessory.

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

The following procedure describes how to upload on your USB memory key the current project stored in the controller:

Step	Action
1	Remove any file from the root of your USB memory key. If there is a controller pair of files in the USB memory key, the download *(see page 156)* function starts automatically.
2	Power on your controller.
3	Connect the USB memory key to the Loader Device Accessory.
4	Connect the Loader Device Accessory to the controller with USB cable.
5	Move the Loader Device Accessory On/Off switch to the On position. **NOTE:** The controller is set to STOPPED state.
6	Wait for the files to be uploaded on the USB memory key. (LED *(see page 148)* at Normal operation state).
7	At the end of the upload (LED *(see page 148)* at Operation complete state), move the Loader Device Accessory On/Off switch to the Off position.
8	Disconnect the Loader Device Accessory from the controller.
9	Disconnect the USB memory key and the USB cable from the Loader Device Accessory.
10	Execute a Run command *(see page 56)* or use the Set the controller to RUNNING state *(see page 160)* feature to set the controller to RUNNING state.

The Loader Device Accessory is automatically switched off

- after 10 s, if the upload has been successfully completed.
- after 60 s, if the upload has been unsuccessful.

Download From the USB Memory Key to the Controller

Procedure

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- You must have operational knowledge of your machine or process before connecting this device to your controller.
- Be sure that your machine or process is in a know safe state (STOP, SHUTDOWN, INERT, etc.) before transitioning the Loader Device Accessory to the “On” switch position while connected to your controller.
- Be sure that guards are in place so that any potential unintended equipment operation will not cause injury to personnel or damage to equipment.
- You must have read and understood the user documentation of this and other devices involved in the functions performed by the Loader Device Accessory.

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

The following procedure describes how to download to your controller the current project stored in the USB memory key:

Step	Action
1	To download a project in the USB key: ● without **Symbol configuration**: 1 pair of files (`*.app` and `*.crc`) must be stored in the root of the USB memory key. ● with **Symbol configuration**: 2 pairs of files (`*.app` and `*.crc`, `*_Symbols.app` and `*_Symbols.crc`) must be stored in the root of the USB memory key. If there is no pair of files stored in the root of the USB memory key, the upload function *(see page 154)* starts automatically.

NOTICE
LOSS OF DATA ● Be sure if the physical controller reference and the controller reference in the application are compatible *(see page 150)*. ● Be sure that all the necessary files are present before downloading any files to the controller. **Failure to follow these instructions can result in equipment damage.**

2	Power on the controller.
3	Connect the USB memory key to the Loader Device Accessory.
4	Connect the Loader Device Accessory to the controller with USB cable.
5	Move the Loader Device Accessory On/Off switch to the On position. **NOTE:** The controller is set to STOPPED state.
6	Wait until the files are downloaded to the controller. (LED *(see page 148)* at Normal operation state.) **NOTE:** The controller is in EMPTY State.

Interrupting a download can erase the controller memory.

NOTICE
INOPERABLE EQUIPMENT ● Do not disconnect the USB memory key during the download. ● Do not disconnect the USB cable to the controller during the download. ● Do not switch off the Loader Device Accessory during the download. ● Do not remove power from the controller. **Failure to follow these instructions can result in equipment damage.**

NOTE: In the event that the download was interrupted, move the Loader Device Accessory switch to the off position and back to the on position to restart the download.

Step	Action
7	At the end of the download (LED *(see page 148)* at Operation complete state), move the Loader Device Accessory On/Off switch to the Off position.
8	Reboot the controller by doing a power cycle for the application to be recognized.

Depending on your controller and/or configuration, the controller may be either in a RUNNING or a STOPPED state.

⚠ WARNING
UNINTENDED EQUIPMENT OPERATION Consult the controller state and behavior diagram *(see page 43)* 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.**

9	Disconnect the Loader Device Accessory from the controller.
10	Disconnect the USB memory key and the USB cable from the Loader Device Accessory.
11	If your controller is in a STOPPED state, execute a Run command *(see page 56)* or use the Set the controller to RUNNING state *(see page 160)* feature to set the controller to RUNNING state.

The Loader Device Accessory is automatically switched off
- after 10 s, if the download has been successfully completed.
- after 60 s, if the download has been unsuccessful.

Section 16.4
Other Functionalities

What Is in This Section?

This section contains the following topics:

Topic	Page
Set the Controller to RUNNING State	160
Change the Firmware of the Loader Device Accessory	161

Set the Controller to RUNNING State

Procedure

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- You must have operational knowledge of your machine or process before connecting this device to your controller.
- Be sure that your machine or process is in a know safe state (STOP, SHUTDOWN, INERT, etc.) before transitioning the Loader Device Accessory to the “On” switch position while connected to your controller.
- Be sure that guards are in place so that any potential unintended equipment operation will not cause injury to personnel or damage to equipment.
- You must have read and understood the user documentation of this and other devices involved in the functions performed by the Loader Device Accessory.

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

The following procedure describes how to set a controller to RUNNING state using the Loader Device Accessory.

Step	Action
1	Disconnect the USB key from the Loader Device Accessory.
2	Connect the Loader Device Accessory to the controller.
3	Move the Loader Device Accessory On/Off switch to the On position. **NOTE:** The controller is set in RUNNING state.
4	Move the Loader Device Accessory On/Off switch to the Off position.
5	Disconnect the Loader Device Accessory from the controller.

The Loader Device Accessory is automatically switched off

- after 10 s, if the update has been successfully completed.
- after 60 s, if the update has been unsuccessful.

Change the Firmware of the Loader Device Accessory

Procedure

The following procedure describes how to change the Loader Device Accessory firmware:

Step	Action
1	The USB memory key must only contain the following files: ● `key.x.production.hex` ● `plc.x.production.hex` where x is the product firmware version. **NOTE:** Connect to www.schneider-electric.com to download the latest firmware change.
2	Connect the USB memory key to the Loader Device Accessory.
3	Move the Loader Device Accessory On/Off switch to the On position.
4	At the end of the change (LED *(see page 148)* at Operation complete state), move the Loader Device Accessory On/Off switch to the Off position.
5	Disconnect the USB key from the Loader Device Accessory.

The Loader Device Accessory is automatically switched off

- after 10 s, if the change has been successfully completed.
- after 60 s, if the change has been unsuccessful.

Chapter 17
Changing the Controller Firmware

Overview

This chapter provides detailed instructions on using the Windows Exec Loader to change the firmware of your M238 controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Changing Through Serial Line	164
Changing Through USB	167
Launching the Exec Loader Wizard	169
Step 1 - Welcome	170
Step 2 - Settings	171
Step 3 - File and Device Exec Properties	173
Step 4 - Transfer Progress	175

Changing Through Serial Line

Introduction

The firmware change through serial line *(see page 116)* is not available for all M238 Product Versions:

Controller Reference	Change through Serial Link Availability
TM238LDD24DT	Product Version (PV) < 08
TM238LFDC24DT	Product Version (PV) < 08
TM238LFDC24DTSO	Product Version (PV) < 02
TM238LDA24DR	No serial line change
TM238LFAC24DR	No serial line change

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.

Changing the firmware requires one of the following cable:

- The TSXCUSB485 the VW3A8306Rxx cable
- or the TCSMCNAM3M002P cable.

This changing procedure is a maintenance operation. It requires that the controller be disconnected from the systems and applications it effects. The PC and the controller must stay connected during this operation.

NOTE: If the PC and the controller are unintentionally disconnected during the firmware change, the controller will not function correctly until a new, successful firmware change operation is performed.

Installing the USB Cable

Follow these steps to install the TSXCUSB485 cable properly:

Step	Action
1	On the TSXCUSB485 adaptor, select the OTHER MULTI mode on rotary switch and the OFF position for polarization.
2	Connect the TSXCUSB485 adaptor into an USB Port of your PC. **NOTE:** On first connection to the computer, it could be required to install the driver (see below).
3	Connect the VW3A8306Rxx cable to the RJ45 connector of the TSXCUSB485.
4	Connect the second end of the VW3A8306Rxx cable into SL1 port of the Modicon M238 Logic Controller.
5	Launch the Exec Loader wizard serial *(see page 169)*

Follow these steps to install the TCSMCNAM3M002P cable properly:

Step	Action
1	Connect the TCSMCNAM3M002P adaptor into an USB Port of your PC. **NOTE:** On first connection to the computer, it could be required to install the driver (see below).
2	Connect the second end of the TCSMCNAM3M002P cable into SL1 port of the Modicon M238 Logic Controller.
3	Launch the Exec Loader wizard serial *(see page 169)*

USB Cable Driver Installation

After connection, the USB cable is detected by the PC. If the cable driver is not installed, a popup saying new hardware has been found is displayed. In this case, install the driver.

Step	Screen	Action
1	Found New Hardware Wizard	Can Windows connect to Windows Update to search for software? Select **No, not this time** and click **Next**.
2	Found New Hardware Wizard	What do you want the Wizard to do? Select **Install the software automatically (Recommended)** and click **Next**.
3	Hardware installation	Click **continue** anyway.
4	Completing the Found New Hardware Wizard.	Click **Finish**.

Changing Through USB

Introduction

The firmware change through USB Link is not available for all M238 Product Versions:

Controller Reference	Change through USB Availability
TM238LDD24DT	Product Version (PV) >= 08
TM238LFDC24DT	Product Version (PV) >= 08
TM238LFDC24DTSO	Product Version (PV) >= 02
TM238LDA24DR	All versions
TM238LFAC24DR	All versions

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 port(s) of your controller are configured for the SoMachine protocol by default when new or when you change the controller firmware. The SoMachine protocol is incompatible with that of other protocols such as Modbus Serial Line. Connecting a new controller to, or changing 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. Verify that the controller is not connected to an active Modbus serial line network before first downloading a valid application that has the concerned port or ports properly configured for the intended protocol.

NOTICE
UNINTENDED EQUIPMENT OPERATION Verify your application has the Serial Line port(s) 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.**

Before starting the firmware change procedure, verify you have:

- USB cable TCSXCNAMUM3P
- Modicon M238 Logic Controller

This changing procedure is a maintenance operation. It requires that the controller be disconnected from the systems and applications it effects. The PC and the controller must stay connected during this operation.

NOTE: If the PC and the controller are unintentionally disconnected during a firmware change, the controller will not function correctly until a new, successful firmware change operation is performed.

Installing Cables

Follow these steps to install the cables properly:

Step	Action
1	Plug the TCSXCNAMUM3P cable to an USB Port of your PC. *(see page 141)*
2	Plug the second end of the cable to the USB port of the controller.
3	Launch the Exec Loader Wizard USB *(see page 169)*

Launching the Exec Loader Wizard

Introduction

The Exec Loader wizard is a Windows-based wizard that guides you through the steps necessary to change the firmware of your M238 controller.

Opening the Exec Loader Wizard

To launch the Exec Loader wizard, complete the following steps:

Step	Action
1	Close all your windows applications, including virtual machines.
2	If the gateway is running, right-click the **CoDeSys Gateway Sys Try (running)** icon in the task bar and select **Stop Gateway**. When the gateway is stopped, the **CoDeSys Gateway Sys Tray (stopped** icon appears in the task bar:
3	If your Controller and the PC are connected with: • **USB Interface**, click **Start → Programs → Schneider Electric → SoMachine → Tools → Exec Loader Wizard USB** • **Serial Interface**, click **Start → Programs → Schneider Electric → SoMachine → Tools → Exec Loader Wizard Serial**

Overview of Change Steps

The wizard provides a screen for each step. The following table summarizes the 4 steps required to change your firmware:

Step	Screen	Function
1	Welcome *(see page 170)*	Introduction to the Exec Loader wizard.
2	Settings *(see page 171)*	Select the correct firmware file to transfer to your controller.
3	File and Device Exec Properties *(see page 173)*	Compare the hardware IDs and the firmware version information of the firmware file and the controller.
4	Transfer Progress *(see page 175)*	Monitor the transfer of the firmware file to the controller.

Step 1 - Welcome

Step 1 - Welcome

The wizard provides a screen for each step. The Welcome screen is an introduction to the Exec Loader wizard.

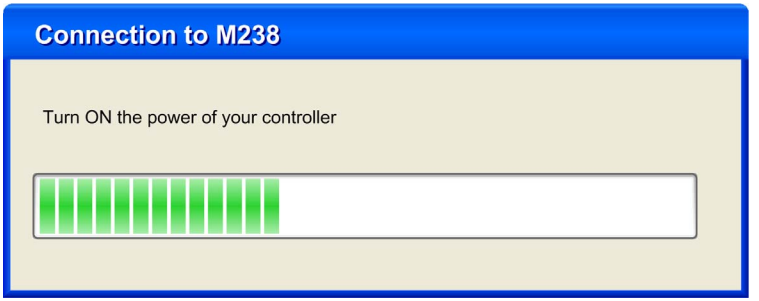
To continue:

- Select **Next** to continue the procedure and display the next screen, Step 2 - Settings *(see page 171)*.
- Select **Close** to close the screen without changing the firmware of your controller.

Step 2 - Settings

Selecting Settings

Use these steps to select the appropriate firmware:

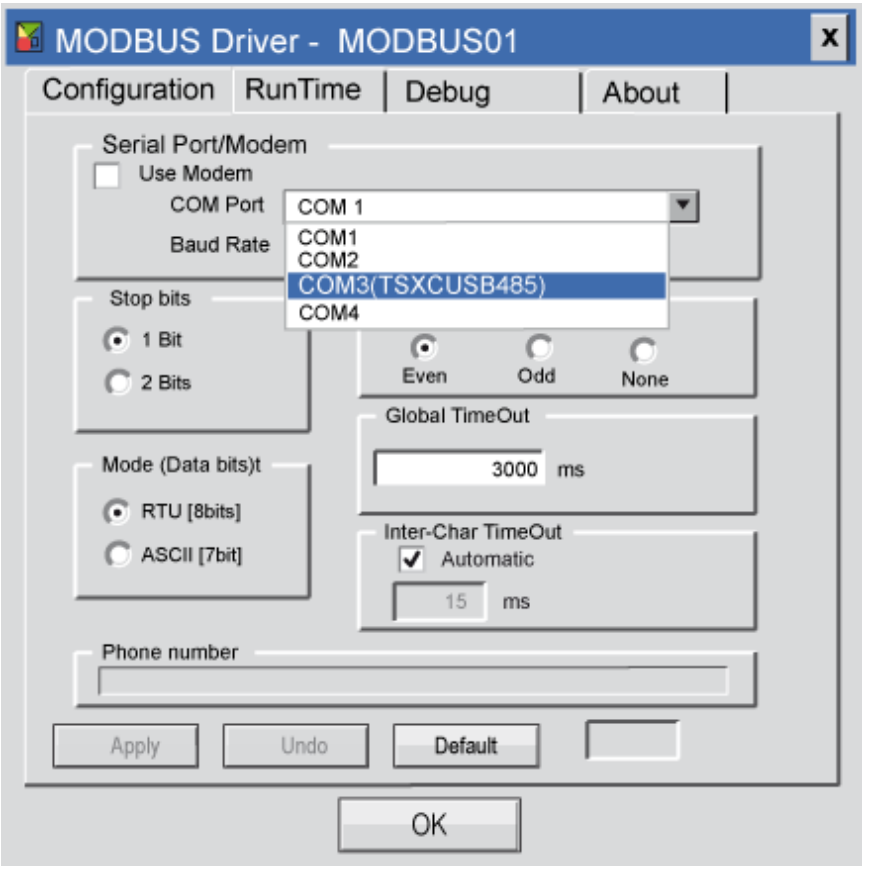
Step 2	Action
1	In **Settings**, click **Browse** and select the correct file for your controller model. Example: *C:\Program Files\Schneider Electric\SoMachine\Firmware\M238\TM238LFDC24DT.mfw*
2	Remove power from the Controller, as indicated on the screen.
3	Select **Next**. During the progress bar, you will be prompted to re-apply power to the controller.  When the Exec Loader Wizard has successfully opened a connection with the controller, it goes automatically to step 3 *(see page 173)*. If it doesn't, the progress bar disappears and the Exec Loader Wizard goes back to screen of Step1. Then select **Next** to try again. **NOTE:** Verify the USB connection or reboot the controller by doing a power cycle.

Troubleshooting with the TSXCUSB485 cable

If the controller is not detected during Step 2, start the Modbus Driver by clicking **Start → Programs → Schneider Electric → Communication Drivers → Modbus Driver**.

Double-click the corresponding icon in the task bar to open the Modbus Driver screen, and verify that the USB cable is connected to the selected COM port.

Modbus Driver screen:

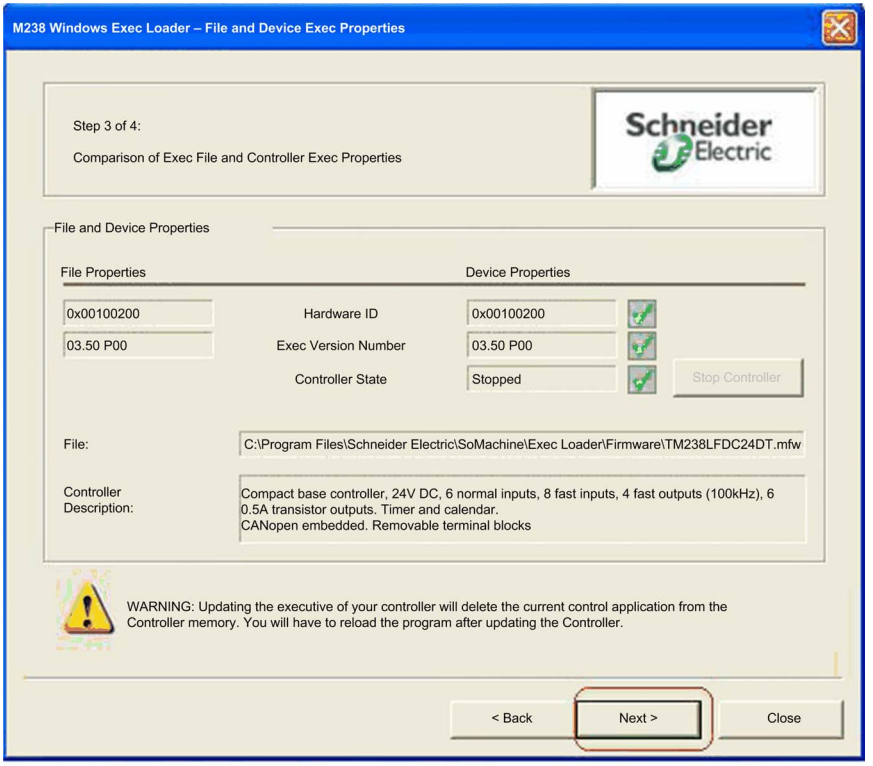


Step 3 - File and Device Exec Properties

Overview

At this step, the following information is checked by the Exec Loader Wizard for both the firmware file and your controller before the procedure can continue:

- **Hardware ID** - the selected firmware file is correct for the target controller.
- **Exec Version Number** - the selected firmware file is newer than the currently installed firmware.



Hardware ID

The **Hardware ID** is a unique identifier for each controller reference:

- **Green check mark**: OK
- **Red cross**: incorrect firmware file. Select a firmware file corresponding to your controller reference (go back to step 2 *(see page 171)*)

Exec Version Number

The **Exec Version Number** identifies the version of the firmware:

- **Green check mark**: you will change your controller to a newer version of the firmware
- **Yellow check mark**: you will change your controller to an older version of the firmware or change your controller with the same version of the current firmware

Starting the Transfer

Click on the **Next** button to start the transfer.

Step 4 - Transfer Progress

Overview

In this screen you can monitor the transfer progress.

The remaining time information is available after a while.

If Transfer is Successful

If the transfer is successful, a message box is displayed to allow for another transfer. Two options are available:

- Yes - the wizard returns to Step 2 - Settings *(see page 171)* and you can set up another transfer.
- No - click on the **Close** button to exit the wizard. This completes the change procedure.

If Transfer is not Successful

If the transfer is interrupted (for example, due to a loss of communication), a message box is displayed allowing a retry of the transfer. Two options are available:

- Yes - the wizard returns to Step 3 - Files and Device Exec Properties *(see page 173)* and you can try another transfer.
- No - click on the **Close** button to exit the wizard.

Your controller remains inoperative until a successful transfer has been accomplished.

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.**

Chapter 18
Compatibility

Software and Firmware Compatibilities

SoMachine Compatibility and Migration

Software and Firmware compatibilities are described in the SoMachine Compatibility and Migration User Guide.

Chapter 19
Modicon M238 Logic Controller - Troubleshooting and FAQ

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Troubleshooting	180
Frequently Asked Questions	187

Troubleshooting

Introduction

This section describes the procedures to troubleshoot your Modicon M238 Logic Controller.

Transferring the Application is not Possible

Possible causes:

- PC cannot communicate with the controller.
- Is your application valid?
- Is the CoDeSys gateway running?

Resolution:

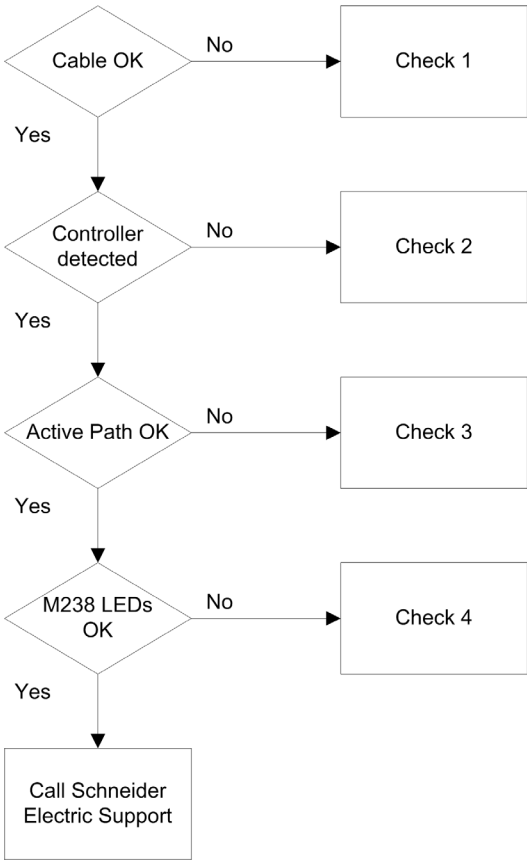
- Refer to the part below (Communication Between SoMachine and the Modicon M238 Logic Controller *(see page 181)*).
- Your application program must be valid. Refer to the debugging part of the SoMachine Programming Guide.
- The CoDeSys gateway must be running:
  a. click the CoDeSys Gateway **SysTray (stopped)** icon in the task bar,
  b. select **Start Gateway**.

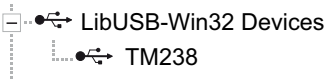
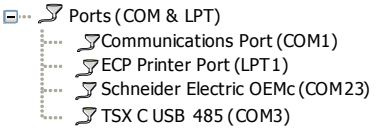
Communication Between SoMachine on a Computer and the Modicon M238 Logic Controller is not Possible.

Possible causes:

- Incorrect cable usage.
- PLC not detected by the PC.
- Communication settings are not correct.
- The controller is not operating correctly.

Resolution:



Verify	Action
1	verify that: ● the cable is correctly linked to the controller and to the PC and not damaged, ● you use the specific cable or adapter depending on the connection type: ❍ TCS XCNA MUM3P cable for an USB connection. ❍ TSX CUSB 485 and an Ethernet cable for a serial line RS485/RS232 connection.
2	Verify that the Modicon M238 Logic Controller has been detected by your PC: **1.** click **Start → Control Panel → System**, select the **Hardware** tab and click **Device Manager**, **2.** Verify that the Modicon M238 Logic Controller node appears in the list: ❍ if you use the USB connector:  ❍ if you use the serial line through TSXCUSB485:  **3.** If the Modicon M238 Logic Controller node does not appear or if there is an icon in front of the node, unplug/plug the cable on the controller side.
3	Verify that the active path is correct: **1.** double click the Controller node in the **Devices** window, **2.** Verify that the Modicon M238 Logic Controller node appears in **bold** and not in *italic*. If not: **a.** stop the CoDeSys Gateway: right click the **CoDeSys Gateway SysTray (running)** icon in the task bar and select **Stop Gateway** , **b.** unplug/plug the cable on the controller side, **c.** start the CoDeSys gateway: right click the **CoDeSys Gateway SysTray (stopped)** icon in the task bar and select **Start Gateway** , **d.** select the gateway in the Controller configuration window of SoMachine and click **Scan network**. Select the Modicon M238 Logic Controller node and click **Set active path**. **NOTE:** If the PC is connected to an Ethernet network, the IP address might change. In this case, the Modicon M238 Logic Controller node appears in italics (the path set to the controller is incorrect). To refresh the active path: **1.** Select the Modicon M238 Logic Controller node **2.** Click **Resolve Name**: if the PC detects the controller on the network, a new path is defined and the node does not appear in italics anymore **3.** Click **Set Active Path**
4	Refer to the System Diagnostic using LED Display section *(see M238 Logic Controller, Hardware Guide)*.

Application program is not executed

Possible causes:

No POU declared in the task.

Resolution:

As POUs are managed by tasks, you must add at least one POU to a task:

1. double click a task in the **Applications tree** tab,
2. click **Add Call** in the task configuration window,
3. select the POU you want to execute in the **Input Assistant** window and click **OK**.

Possible causes:

- Application does not go to RUN state.
- One input is configured in RUN/STOP mode.

Resolution:

Use the input configured in RUN/STOP mode to run the application.

CoDeSys Gateway does not start (CoDeSys Gateway SysTray icon is black)

Possible cause:

Connection during a long time.

Resolution:

If the **CoDeSys Gateway SysTray** icon is black (stopped):

1. Open the task Manager,
2. stop the **Gatewayservice.exe**, and start it again:
   - Restart your computer **or**,
   - in **Control Panel**, open **Administrative Tools** and **Computer Management**,
   - in **Service**, double click **CoDeSys Gateway**,
   - Click **Start Service** button.
3. Verify if the **CoDeSys Gateway SysTray** icon is red (running).

Serial Line Communication is not Possible

Possible causes:

- Communication settings are not identical between serial line devices.
- The controller is not operating correctly.

Resolution:

Verify that:

- protocol communication settings (baud rate, parity...) are identical for all serial line devices.
- The correct communication manager is added on the Serial Line object:
  - **Modbus manager** if the line is used for Modbus protocol,
  - **SoMachine-Network Manager** if the line is used for communication to access IEC variables.
- the controller operates correctly. Refer to the System Diagnostic using LED Display part *(see M238 Logic Controller, Hardware Guide)*.

Creating the Boot Application is not Possible

Possible cause:

Operation not possible while the controller is in RUN state.

Resolution:

- Select **Stop Application**,
- Select **Create Boot Project**.

PTO Function does not Start

Possible cause:

The AUX input is configured as the `Drive Ready` input but no signal is being supplied.

Resolution:

- If the `AUX` variable is set to `Drive Ready`, verify that the drive is correctly operating
- or set the `Dis_Drive_Ready` variable of the `PTOsimple` function block to 0.

Changing Device Name do not work

Possible cause:

Application is running.

Resolution:

- Select **Stop Application**,
- Change device name.

CANopen Heartbeat is not sent on a regular basis

Possible cause:

The Heartbeat configured value is not a multiple of the CANopen bus Cycle Task interval.

Resolution:

Change the Heartbeat value to a multiple of the CANopen bus Cycle Task interval.

Monitoring of the POU is slow

Possible cause:

Task interval is too small or the number/size of POUs is too great.

Resolution:

- Increase the configured task interval.
- Split the application into smaller POUs.

ERR LED is flashing fast on the PLC

Possible cause:

A system error was detected.

Resolution:

Verify your application program (pointer management, arrays management, etc...).

Controller is in HALT State

Possible cause:

The PLC has stopped due to a watchdog event.

Resolution:

- If a task watchdog is configured:
  a. Run the application without task watchdog
  b. Get the maximum task cycle time from the task monitor
  c. Set the task watchdog greater than the maximum task cycle time
- If a task watchdog is not configured:
  - If a **Cyclic task** is configured, increase the cycle time to a value > 1.25 times the average task time
  - If several tasks are configured, and one of these is a **Freewheeling task**, try reconfiguring the Freewheeling task as a Cyclic task

Possible cause:

The cycle time is extended when the CANopen configurator is called, leading to a task watchdog exception.

The controller may report a watchdog exception during the following events:

- downloading of configuration data to the modules of the network (i.e. when downloading the application to the controller, after a power-on of the controller when a boot application is valid, or after a reset warm/cold).
- CANopen cables connection may have been disconnected or dislodged.

Resolution:

1. Run the application without task watchdogs
2. Get the maximum task cycle time from the task monitor
3. Set the task watchdog greater than the maximum task cycle time

Possible cause:

A division by 0 is detected in the application program.

Resolution:

Verify your application program.

Source Download leads to Communication Interruptions

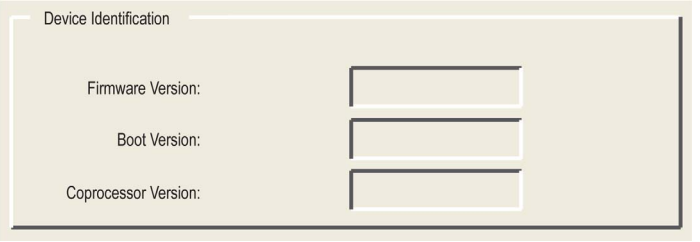
The following table describes the possible causes of a communication interruption during **Source Download**:

Possible Cause	Resolution
You attempted to download the source while the controller was in a RUN state.	Stop the controller before attempting the download.
The source file exceeded the available memory space in the controller.	If sending additional files with the source, consider deselecting them to reduce the overall size of the download. See **Project → Project Settings → Source Download → Additional Files...** in the SoMachine main menu.

Frequently Asked Questions

How can I Determine the Firmware, Boot and Coprocessor Version of the Controller?

In online mode, double click the Controller node in the **Devices tree** tab. In the Controller window, select the **Services** tab. The device identification area gives information about versions:



What Programming Languages are supported by a Modicon M238 Logic Controller?

Refer to Supported Programming languages *(see page 15)*.

What Variable Types are supported by a Modicon M238 Logic Controller?

The following variable types are supported:

- BOOL
- Integer data types
- REAL
- LREAL
- STRING
- WSTRING
- Time data types

When should I use Freewheeling or Cyclic Task Type?

Freewheeling or cyclic task type usage Task Configuration *(see page 33)*:

- Freewheeling: use this setting if a variable cycle time is permissible for your application. The next cycle will start after a waiting duration equal to 30% of the last cycle execution duration.
- Cyclic: use this mode if you want to control the cycle time.

What are the Effects of Cold/Warm Restart?

Refer to the effects of reset cold/warm section *(see page 56)*.

Can I connect the PC (SoMachine) and the Controller through 499TWD01100 Ethernet Gateway?

No, because the Ethernet Gateway only supports Modbus protocol.

Can I connect several M238, through several USB ports of my PC?

No, because driver conflicts may occur.

Why the communication between the HMI and the controller is interrupted when making online changes?

When online changes are made to a M238 application, the Symbol Configuration is downloaded. This results in a temporary interruption of the communication.

Appendices

Overview

This appendix lists the documents necessary for technical understanding of the M238 Programming Guide.

What Is in This Appendix?

The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	AS-Interface Library	191
B	Function and Function Block Representation	211
C	Functions to Get/Set Serial Line Configuration in User Program	221
D	Controller Performance	227

Appendix A
AS-Interface Library

Overview

This chapter describes the function blocks included in the **IoDrvASI** library.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
ASI_CheckSlaveBit	192
ASI_CmdSetAutoAddressing	194
ASI_CmdSetDataExchange	196
ASI_CmdSetOfflineMode	198
ASI_MasterStatusCheck	200
ASI_SlaveAddressChange	202
ASI_SlaveParameterUpdate	205
ASI_SlaveStatusCheck	207
ASI_ReadParameterImage	209

ASI_CheckSlaveBit

Function Description

This function returns the status bit of a specified AS-Interface slave from a specified AS-Interface status table (**LDS, LAS, or LPF**).

This function is to be used in combination with ASI_SlaveStatusCheck *(see page 207)* function block used to read **LDS**, **LAS**, and **LPF** status tables from the AS-Interface master.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation (see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
byAddress	BYTE	AS-Interface Slave address (bit offset 0 to 63). 0 = address 0 1...31 = address 1...31 for standard addressing mode, or 1 A...31 A for extended addressing mode 32 = not used 33 to 63 = 1B...31B for extended addressing mode

The following table describes the output variables:

Output	Type	Comment
`ASI_CheckSlaveBit`	`BOOL`	Returns the value of the bit at the offset `byAddress` within the `abyStatusBytes` array.

The following table describes the input/output variables:

Input/Output	Type	Comment
`abyStatusBytes`	`ARRAY[0..7] OF BYTE`	AS-Interface status table (e.g.: LDS, LAS, or LPF *(see page 207)*)

ASI_CmdSetAutoAddressing

Function Description

This function blocks allows to activate or deactivate the auto-addressing mode. By default, auto-addressing is set to the value that is configured in AS-Interface Master module configuration window *(see page 98)*.

Graphical Representation (LD/FBD)

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
`xExecute`	`BOOL`	Rising edge: action starts. Falling edge: resets outputs. If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error has been detected. In this case, the corresponding output values (`xDone`, `xError`, `iError`) are present at the outputs for exactly one cycle.
`xAutoAddressingActive`	`BOOL`	TRUE= enables auto-addressing mode. FALSE= disables auto-addressing mode.

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE if command terminated successfully. The value returned by the status flag ASI_MasterStatusCheck.Auto_Address_Assign is equal to the command requested by ASI_CmdSetAutoAddressing.xAutoAddressingActive.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: detected error, function block aborts action FALSE: no error has been detected

The following table describes the input/output variables:

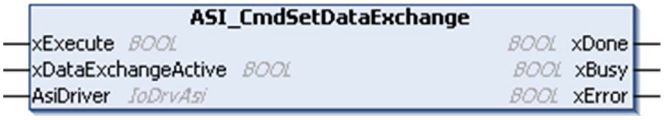
Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ASI_CmdSetDataExchange

Description

This function enables or disables the data exchange between AS-Interface Master and Slave Modules. The data exchange is active after reset.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
`xExecute`	`BOOL`	Rising edge: action starts. Falling edge: resets outputs. If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error has been detected. In this case, the corresponding output values (`xDone`,`xError`, `iError`) are present at the outputs for exactly one cycle.
`xDataExchangeActive`	`BOOL`	TRUE= enables the data exchange. FALSE= disables the data exchange.

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE if command terminated successfully.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: detected error, function block aborts action FALSE: no error has been detected

The following table describes the input/output variables:

Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ASI_CmdSetOfflineMode

Description

This function block permits to set the AS-Interface Master Module in offline mode. After a reset of the application, offline mode is disabled.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
`xExecute`	`BOOL`	Rising edge: action starts. Falling edge: resets outputs. If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error has been detected. In this case, the corresponding output values (`xDone`,`xError`, `iError`) are present at the outputs for exactly one cycle.
`xOfflineModeActive`	`BOOL`	TRUE= enables offline mode. FALSE= disables offline mode.

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE if command terminated successfully. The value returned by the status flag ASI_MasterStatusCheck.Auto_Address_Assign is equal to the command requested by ASI_CmdSetAutoAddressing.xAutoAddressingActive.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: detected error, function block aborts action FALSE: no error has been detected

The following table describes the input/output variable:

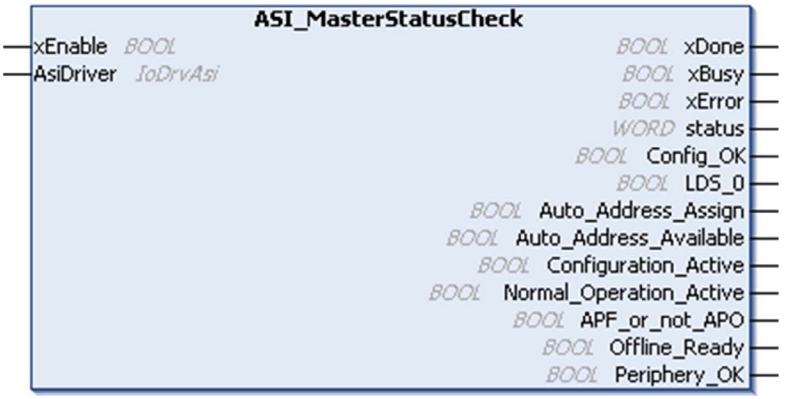
Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ASI_MasterStatusCheck

Description

This function block returns the status of the AS-Interface Master Module.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 211)*.

I/O Variables Description

The following table describes the input variable:

Input	Type	Comment
`xEnable`	`BOOL`	TRUE: action running FALSE: action stopped. Outputs `xDone`, `xBusy`, `xError` and `iError` are reset.

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	Not used.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: an error has been detected, function block aborts action FALSE: no error has been detected
status	WORD	Status bits of AS-Interface master in one WORD: ● bit 0 to 7 = status[0] ● bit 8 to 15 = status[1]
Config_OK	BOOL	Config OK (bit 0)
LDS_0	BOOL	Address 0 Slave (bit 1)
Auto_Address_Assign	BOOL	Auto-addressing mode enabled (bit 2)
Auto_Address_Available	BOOL	Auto-addressing will be processed as soon as a slave with zero address and valid configuration data is connected (bit 3)
Configuration_Active	BOOL	Configuration mode active (bit 4)
Normal_Operation_Active	BOOL	Normal operation mode active (bit 5)
APF_or_not_APO	BOOL	Power outage (bit 6)
Offline_Ready	BOOL	Offline mode active (bit 7)
Periphery_OK	BOOL	No peripheral error detected (all entries in LPF are 0) (bit 8)

The following table describes the input/output variable:

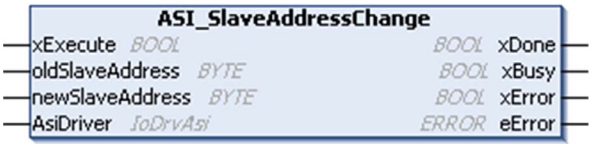
Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ASI_SlaveAddressChange

Description

This function block enables to change the address of an AS-Interface Slave Module.

Graphical Representation



IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation (see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
`xExecute`	`BOOL`	Rising edge: action starts. Falling edge: resets outputs. If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error has been detected. In this case, the corresponding output values (`xDone, xError,eError`) are present at the outputs for exactly one cycle.
`oldSlaveAddress`	`BYTE`	Address of Slave to be readdressed 0 = address 0 1...31 = address 1...31 for standard addressing mode, or 1 A...31 A for extended addressing mode 32 = not used 33...63 = 1 B...31 B for extended addressing mode

Input	Type	Comment
newSlaveAddress	BYTE	New address of Slave 0 = address 0 1...31 = address 1...31 for standard addressing mode, or 1 A...31 A for extended addressing mode 32 = not used 33...63 = 1 B...31 B for extended addressing mode

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE if command terminated successfully.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: detected error, function block aborts action FALSE: no error has been detected
eError	ERROR	Contains the detected error code

The following table describes the input/output variable:

Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ERROR: Detected Error Codes

The ERROR enumeration data type contains the following values:

Enumerator	Value	Description
NO_ERROR	00 hex	No error detected
FIRST_ERROR	64 hex	Library specific detected errors start delimiter
TIME_OUT	65 hex	Operation aborted on Time Out
ADDRESS_IN_USE	66 hex	New slave address parameter already assigned
INVALID_ADDRESS	67 hex	Invalid old or new slave address parameter
NO_SLAVE	68 hex	Old slave address parameter not assigned
INVALID_PARAMETER	69 hex	Parameter value out of range
NO_EXT_ADDR_SUPP	6A hex	Extend address not supported

Enumerator	Value	Description
FIRST_MF	78 hex	Manufacturer specific detected errors start delimiter
...	...	...
LAST_ERROR	96 hex	Specific detected error stop delimiter

ASI_SlaveParameterUpdate

Description

This function block permits to set the parameters of an AS-Interface Slave Module.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
xExecute	BOOL	Rising edge: action starts. Falling edge: resets outputs. If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error has been detected. In this case, the corresponding output values (xDone, xError,eError) are present at the outputs for exactly one cycle.
slaveAddress	BYTE	Address of the AS-Interface Slave. 0 = address 0 1...31 = address 1...31 for standard addressing mode, or 1 A...31 A for extended addressing mode 31 = not used 33...63 = 1 B...31 B for extended addressing mode
parameters	BYTE	New value of the Slave parameters (value from 00 h to 0Fh).

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE if command terminated successfully.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: detected error, function block aborts action FALSE: no error has been detected
eError	ERROR *(see page 203)*	Contains the detected error code.

The following table describes the input/output variable:

Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ASI_SlaveStatusCheck

Description

This function block reads the local lists dedicated to AS-Interface Slave Modules: detected slaves, activated slaves, and slaves reporting the detection of a peripheral error.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation* *(see page 211)*.

I/O Variables Description

The following table describes the input variable:

Input	Type	Comment
xEnable	BOOL	Enable execution.

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	Not used
xBusy	BOOL	Function block active
xError	BOOL	TRUE: an error has been detected, function block aborts action FALSE: no error has been detected

Output	Type	Comment
LAS	ARRAY[0..7] OF BYTE	List of Activated Slaves (LAS): one bit is set for each activated slave. LAS[0] bit 0 = slave at address 0 LAS[0] bit 1 = slave at address 1A ... LAS[3] bit 7 = slave at address 31A LAS[4] bit 0 =not used LAS[4] bit 1 =slave at address 1B ... LAS[7] bit 7 =slave at address 31B
LDS	ARRAY[0..7] OF BYTE	List of Detected Slaves (LDS): one bit is set for each slave that is detected by the master. LDS[0] bit 0 = slave at address 0 LDS[0] bit 1 = slave at address 1A ... LDS[3] bit 7 = slave at address 31A LDS[4] bit 0 =not used LDS[4] bit 1 =slave at address 1B ... LDS[7] bit 7 =slave at address 31B
LPF	ARRAY[0..7] OF BYTE	List of Peripheral detected errors (LPF): one bit is set for each slave that has detected a peripheral error. LPF[0] bit 0 = slave at address 0 LPF[0] bit 1 = slave at address 1A ... LPF[3] bit 7 = slave at address 31A LPF[4] bit 0 =not used LPF[4] bit 1 =slave at address 1B ... LPF[7] bit 7 =slave at address 31B

The following table describes the input/output variable:

Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

ASI_ReadParameterImage

Description

This function block allows to read and refresh the parameter image table.

Graphical Representation

IL and ST Representation

To see the general representation in IL or ST language, refer to the chapter *Function and Function Block Representation (see page 211)*.

I/O Variables Description

The following table describes the input variables:

Input	Type	Comment
xExecute	BOOL	Rising edge: action starts. Falling edge: resets outputs. If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error detected. In this case, the corresponding output values (xDone, xError,iError) are present at the outputs for exactly one cycle.

The following table describes the output variables:

Output	Type	Comment
xDone	BOOL	TRUE if command terminated successfully.
xBusy	BOOL	Function block active
xError	BOOL	TRUE: an error has been detected, function block aborts action FALSE: no error has been detected

Output	Type	Comment
pPITable	POINTER TO ARRAY [0..31] OF BYTE	Parameter image: contains the actual copies of the parameter output of all active slaves. pPITable^[0] bit 0...3 = slave at address 0 pPITable^[0] bit 4...7 = slave at address 1A ... pPITable^[15] bit 4...7 = slave at address 31A pPITable^[16] bit 0...3 = not used pPITable^[16] bit 4...7 = slave at address 1B ... pPITable^[31] bit 4...7 = slave at address 31B

The following table describes the input/output variable:

Input/Output	Type	Comment
AsiDriver	IoDrvAsi	AS-Interface driver instance.

Appendix B
Function and Function Block Representation

Overview

Each function can be represented in the following languages:

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

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

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Differences Between a Function and a Function Block	212
How to Use a Function or a Function Block in IL Language	213
How to Use a Function or a Function Block in ST Language	217

Differences Between a Function and a Function Block

Function

A function:

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

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

Function Block

A function block:

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

Examples: timers, counters

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

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

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

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

General Information

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

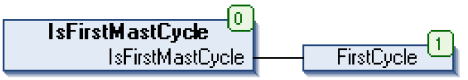
Functions `IsFirstMastCycle` and `SetRTCDrift` and Function Block `TON` are used as examples to show implementations.

Using a Function in IL Language

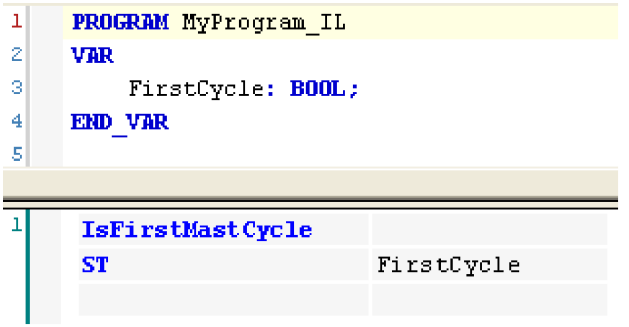
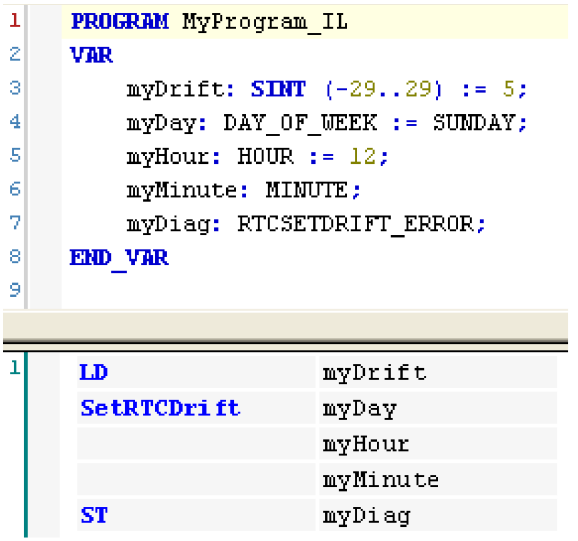
This procedure describes how to insert a function in IL language:

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

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

Function	Graphical Representation
without input parameter: `IsFirstMastCycle`	
with input parameters: `SetRTCDrift`	

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

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

Using a Function Block in IL Language

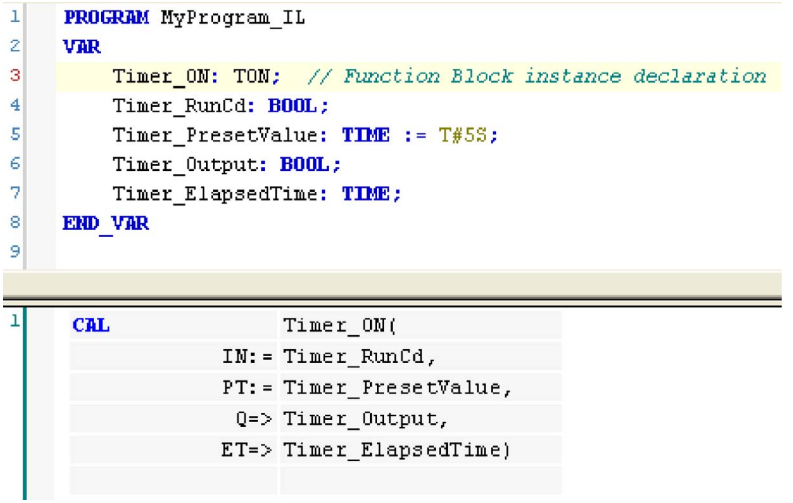
This procedure describes how to insert a function block in IL language:

Step	Action
1	Open or create a new POU in Instruction List language. **NOTE:** The procedure to create a POU is not detailed here. For more information, refer to Adding and Calling POUs *(see SoMachine, Programming Guide)*.
2	Create the variables that the function block requires, including the instance name.
3	Function Blocks are called using a `CAL` instruction: ● Use the **Input Assistant** to select the FB (right-click and select **Insert Box** in the context menu). ● Automatically, the `CAL` instruction and the necessary I/O are created. Each parameter (I/O) is an instruction: ● Values to inputs are set by ":=". ● Values to outputs are set by "=>".
4	In the `CAL` right-side field, replace `???` with the instance name.
5	Replace other `???` with an appropriate variable or immediate value.

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

Function Block	Graphical Representation
TON	

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

Function Block	Representation in SoMachine POU IL Editor
TON	

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

General Information

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

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

Using a Function in ST Language

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

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

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

Function	Graphical Representation
`SetRTCDrift`	

The ST language of this function is the following:

Function	Representation in SoMachine POU ST Editor

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

Using a Function Block in ST Language

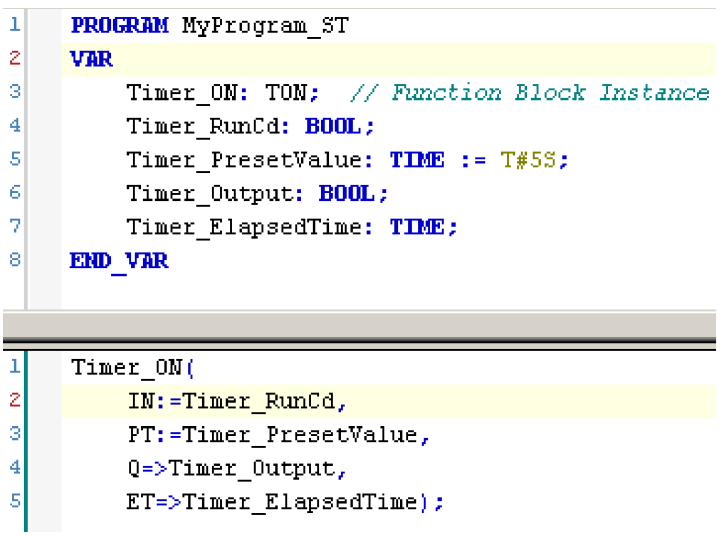
This procedure describes how to insert a function block in ST language:

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

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

Function Block	Graphical Representation
`TON`	

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

Function Block	Representation in SoMachine POU ST Editor
TON	

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

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

Appendix C
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:

Topic	Page
GetSerialConf: Get the Serial Line Configuration	222
SetSerialConf: Change the Serial Line Configuration	223
SERIAL_CONF: Structure of the Serial Line Configuration Data Type	225

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, PLCCommunication 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 D
Controller Performance

Processing Performance

Introduction

This chapter provides information about the Modicon M238 Logic Controller 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	439 µs
Addition/subtraction/multiplication of DINT	506 µs
Addition/subtraction/multiplication of REAL	5111 µs
Addition/subtraction/multiplication of LREAL	9535 µs
Division of REAL	7250 µs
Division of LREAL	23045 µs
Operation on BOOLEAN, for example, Status:= Status and value	971 µs
LD INT + ST INT	420 µs
LD DINT + ST DINT	459 µs
LD REAL + ST REAL	648 µs
LD LREAL + ST LREAL	1235 µs

Basic System Time

This table presents the basic overhead performance for each MAST cycle:

I/O Type	Overhead for Each MAST Cycle
Embedded Inputs & Internal Processing	700 µs
Embedded Outputs	200 µs

HSC, PWM, PTO, and Frequency Generator Processing

This table presents the processing performance for complex functions for each MAST cycle:

Complex Function Type	Overhead for Each MAST Cycle
HSC Simple	150 µs
HSC Main	350 µs
PWM	150 µs
PTO Simple	200 µs
Frequency Generator	150 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
750 µs	950 µs	1750 µs

Glossary

!

%IW

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

%MW

According to the IEC standard, %MW represents a memory word register (for example, a language object of type memory word).

%QW

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

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.

ASIC

(*application specific integrated circuit*) A silicon processor (chip) custom designed especially for an application.

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.

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.

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.

controller

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

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.

cyclic task

The cyclic scan time has a fixed duration (interval) specified by the user. If the current scan time is shorter than the cyclic scan time, the controller waits until the cyclic scan time has elapsed before starting a new scan.

D

digital I/O

(*digital input/output*) An individual circuit connection at the electronic module that corresponds directly to a data table bit. The data table bit holds the value of the signal at the I/O circuit. It gives the control logic digital access to I/O values.

DIN

(*Deutsches Institut für Normung*) A German institution that sets engineering and dimensional standards.

DINT

(*double integer type*) Encoded in 32-bit format.

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

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.

expansion bus

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

F

FB

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

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.

function

A programming unit that has 1 input and returns 1 immediate result. However, unlike FBs, it is directly called with its name (as opposed to through an instance), has no persistent state from one call to the next and can be used as an operand in other programming expressions.

Examples: boolean (AND) operators, calculations, conversions (BYTE_TO_INT)

function block diagram

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

H

hex

(*hexadecimal*)

HSC

(*high-speed counter*)

I

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).

input/output

The index of the ARRAY.

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

LAN

(*local area network*) A short-distance communications network that is implemented in a home, office, or institutional environment.

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).

LED

(*light emitting diode*) An indicator that illuminates under a low-level electrical charge.

LINT

(*long integer*) A whole number encoded in a 64-bit format (4 times `INT` or 2 times `DINT`).

located variable

Refer to (*unlocated variable*).

LRC

(*longitudinal redundancy checking*) An error-detection method for determining the correctness of transmitted and stored data.

LREAL

(*long real*) A floating-point number encoded in a 64-bit format.

LWORD

(*long word*) A data type encoded in a 64-bit format.

M

machine

Consists of several *functions* and/or *equipment*.

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.

Modbus

The protocol that allows communications between many devices connected to the same network.

ms

(*millisecond*)

N

NAK

(*negative acknowledge*)

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*.

OS

(*operating system*) A collection of software that manages computer hardware resources and provides common services for computer programs.

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).

POU

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

program

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

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.

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.

RS-232

A standard type of serial communication bus, based on 3 wires (also known as EIA RS-232C or V.24).

RS-485

A standard type of serial communication bus, based on 2 wires (also known as EIA RS-485).

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.

RTU

(*remote terminal unit*) A device that interfaces with objects in the physical world to a distributed control system or SCADA system by transmitting telemetry data to the system and/or altering the state of connected objects based on control messages received from the system.

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.

SL

(*serial line*)

SoMachine

A comprehensive controller development system software tool for configuring and programming the Modicon logic controller and devices compliant with IEC 61131-3.

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.

symbol

A string of a maximum of 32 alphanumeric characters, of which the first character is alphabetic. It allows you to personalize a controller object to facilitate the maintainability of the application.

system variable

A variable that provides controller data and diagnostic information and allows sending commands to the controller.

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.

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.

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

A

AS-Interface Library
- ASI_CheckSlaveBit, *192*
- ASI_CmdSetAutoAddressing, *194*
- ASI_CmdSetOfflineMode, *198*
- ASI_MasterStatusCheck, *200*
- ASI_ReadParameterImage, *209*
- ASI_SlaveAddressChange, *202*
- ASI_SlaveParameterUpdate, *205*
- ASI_SlaveStatusCheck, *207*

AS-Interface Library
- ASI_CmdSetDataExchange, *196*

AS-Interface V2 Fieldbus
- Add a Slave with Scan Devices, *101*
- Add an AS-Interface Slave, *100*
- Add an Slave With Catalog, *100*
- Add AS-Interface Module, *97*
- Automatic Slave Addressing, *106*
- Configure an AS-Interface Master, *98*
- Configure an AS-Interface Slave, *103*
- Diagnostic, *108*
- General Functional Description, *93*
- Inoperative Slave, *113*
- Manually Add a Generic Slave, *100*
- Modification of Slave Address, *107*
- Presentation, *92*
- Programming, *112*
- Software Setup Principle, *96*

C

Controller Configuration
- Applications, *68*
- Controller Selection, *69*
- PLC Settings, *71*
- Services, *73*

D

Download application, *60*

E

Embedded Functions Configuration
- Embedded HSC Configuration, *76*
- Embedded I/O Configuration, *78*

embedded functions configuration
- embedded PTO_PWM configuration, *82*

Ethernet Gateway Configuration
- Connection and Configuration of the Ethernet Gateway, *135*

Expansion Module
- Adding Expansion Module, *85*
- Configure Expansion Module, *86*

External Event, *37*

F

FAQ, *187*

features
- key features, *15*

Firmware Change
- Change Through Serial Link, *164*
- Change Through USB, *167*
- ExecLoader Introduction, *169*

Firmware change
- Transfer Progress, *175*

Firmware Update
- File and Device Properties, *173*
- Settings, *171*
- Welcome, *170*

functions
- differences between a function and a function block, *212*
- how to use a function or a function block in IL language, *213*
- how to use a function or a function block in ST language, *217*

G

GetSerialConf, *222*

H

Hardware Initialization Values, *54*

L

libraries, *19*

M

Memory Mapping, *23*

O

Output Behavior, *54*, *54*, *54*
Output Forcing, *54*

P

programming languages
    IL, ST, FBD, SFC, LD, CFC, *15*

R

Reboot, *59*
Remanent variables, *63*
Reset cold, *57*
Reset origin, *58*
Reset warm, *57*
Run command, *56*

S

Serial Line
    Serial Line Configuration, *116*
SERIAL_CONF, *225*
SetSerialConf, *223*
Software Initialization Values, *54*
State diagram, *43*
Stop command, *56*

T

Task
    Cyclic task, *35*
    Event task, *37*
    External Event Task, *37*
    Freewheeling task, *36*
    Types, *35*
    Watchdogs, *38*
Troubleshooting, *180*