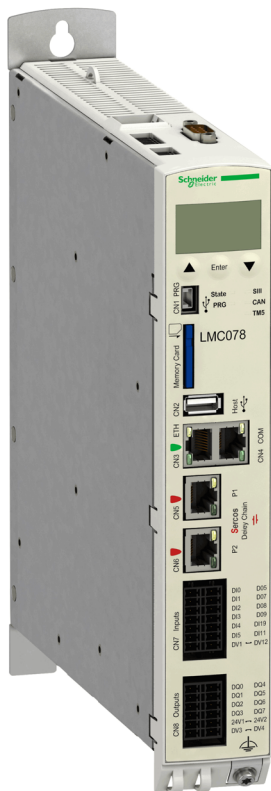


Modicon LMC078

Motion Controller

Programming Guide

03/2018



EIO0000001909.04

www.schneider-electric.com

Schneider
Electric

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

You agree not to reproduce, other than for your own personal, noncommercial use, all or part of this document on any medium whatsoever without permission of Schneider Electric, given in writing. You also agree not to establish any hypertext links to this document or its content. Schneider Electric does not grant any right or license for the personal and noncommercial use of the document or its content, except for a non-exclusive license to consult it on an "as is" basis, at your own risk. All other rights are reserved.

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.

© 2018 Schneider Electric. All Rights Reserved.

Table of Contents



	Safety Information	7
	About the Book	9
Chapter 1	About the Modicon LMC078 Motion Controller	15
	About the Modicon LMC078 Motion Controller	16
	Distributed I/O Architecture	18
Chapter 2	How to Configure the Controller	19
	How to Configure the Controller	19
Chapter 3	Libraries	23
	Libraries	23
Chapter 4	Supported Standard Data Types	25
	Supported Standard Data Types	26
	Parameter Types	27
Chapter 5	Memory Mapping	29
	Controller Memory Organization	30
	RAM Memory Organization	31
	Flash Memory Organization	33
	USB Memory Key	34
Chapter 6	Tasks	35
	Maximum Number of Tasks	36
	Task Configuration Screen	37
	Task Types	39
	Motion Task	42
	System and Task Watchdogs	45
	Task Priorities	46
	Default Task Configuration	49
Chapter 7	Controller States and Behaviors	51
7.1	Controller State Diagram	52
	Controller State Diagram	52
7.2	Controller States Description	56
	Controller States Description	56
7.3	State Transitions and System Events	60
	Controller States and Output Behavior	61
	Commanding State Transitions	64
	Error Detection, Types, and Management	69
	Remanent Variables	70

Chapter 8	Controller Device Editor	73
	Controller Parameters	74
	Configuration Parameters	76
	Controller Selection	86
	PLC Settings	88
Chapter 9	Embedded Inputs and Outputs Configuration	91
	Embedded I/O Configuration	92
	Master Encoder Input Configuration	100
Chapter 10	Communication Modules	107
10.1	PROFIBUS DP Slave Module Configuration	108
	Add a PROFIBUS DP Slave Module	109
	PROFIBUS DP Slave Module Configuration	111
	Acyclic Data Exchange	116
10.2	EtherNet/IP Adapter Configuration	119
	EtherNet/IP Adapter Configuration	120
	Cyclic Data Exchange	124
	Acyclic Data Exchange	125
10.3	Ethernet/IP Scanner Configuration	130
	Presentation	131
	Supported Devices	132
	EtherNet/IP Scanner Configuration	134
	EtherNet/IP Scanner I/O Mapping	136
	EtherNet/IP Scanner Status and Diagnostics	137
	Target Device Declaration	139
	Target Settings	141
	Connection Configuration	143
	Device Replacement with User Parameters	159
	EtherNet/IP I/O Mapping	163
Chapter 11	Ethernet Configuration	165
11.1	Ethernet Services	166
	Presentation	167
	IP Address Configuration	169
	Modbus TCP Client/Server	174
	FTP Server	176
	FTP Client	178
	LMC078 Motion Controller as an IOScanner Slave Device on Modbus TCP	179

11.2	Firewall Configuration	184
	Introduction	185
	Firewall Behavior	187
	Firewall Script Commands.	188
	Script Files.	192
Chapter 12	CANopen Configuration	193
	CANopen Interface Configuration	194
	CANopen Master Configuration.	195
	CANopen Slave Configuration.	197
Chapter 13	Sercos Configuration	199
	Overview of the Sercos Standard	200
	Sercos Interface Configuration	203
	Sercos Devices	208
	Device Addressing Editor	209
	Lexium LXM32S Drive Configuration	213
	TM5NS31 Sercos Interface Module	216
	Sercos Error Codes	217
Chapter 14	Serial Line Configuration	221
	Serial Line Configuration	222
	ASCII Manager	224
	SoMachine Network Manager.	226
	Modbus Serial IOScanner	227
	Adding a Device on the Modbus Serial IOScanner	229
	Modbus Manager.	236
	Adding a Modem to a Manager.	240
Chapter 15	Connecting a Modicon LMC078 Motion Controller to a PC	241
	Connecting the Controller to a PC.	241
Chapter 16	Firmware Update	245
	Updating Modicon LMC078 Motion Controller Firmware.	245
Appendices	249
Appendix A	How to Change the IP Address of the Controller	251
	changeIPAddress: Change the IP address of the controller	251
Appendix B	Diagnostic Messages	255
	Message Logger	256
	Diagnostic Messages	262

Appendix C	LMC078 Sercos3 Library	271
C.1	Data Types	272
	ST_SercosConfiguration Data Type	273
	ST_SercosConfigurationDevice Data Type	274
	ET_Sercos3CmdType Data Type	276
	ET_Sercos3IDNTType Data Type	277
C.2	Sercos Functions	278
	FC_SercosGetConfiguration Function	279
	FC_SercosReadServiceData Function	280
	FC_SercosReadServiceDataByTopAddr Function	283
	FC_SercosScanConfiguration Function	285
	FC_SercosWriteServiceData Function	287
	FC_SercosWriteServiceDataByTopAddr Function	289
C.3	Asynchronous Sercos Function Blocks	291
	FB_SercosReadServiceDataAsync : Read Data Asynchronously via theSercos Interface	292
	FB_SercosWriteServiceDataAsync: Write Data Asynchronously via theSercos Interface	294
	FB_SercosProcedureCommandAsync: Send Commands Asynchro- nously via the Sercos interface	296
Appendix D	Functions to Get/Set Serial Line Configuration in User Program	299
	GetSerialConf: Get the Serial Line Configuration	300
	SetSerialConf: Change the Serial Line Configuration	301
	SERIAL_CONF: Structure of the Serial Line Configuration Data Type	303
Appendix E	Controller Performance	305
	Processing Performance	305
Glossary	307
Index	317

Safety Information



Important Information

NOTICE

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



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



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

DANGER

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

WARNING

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

CAUTION

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

NOTICE

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

PLEASE NOTE

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

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

About the Book



At a Glance

Document Scope

The purpose of this document is to help you to program and operate your Modicon LMC078 Motion Controller.

NOTE: Read and understand this document and all related documents (*see page 9*) before installing, operating, maintaining, or decommissioning your Modicon LMC078 Motion Controller.

The Modicon LMC078 Motion Controller users should read through the entire document to understand all features.

NOTE: For the purposes of this document, *real-time* is defined as processing that is capable of updating information at the same rate as it is receiving data.

Validity Note

This document has been updated for the release of TM3TI4D Add-on for SoMachine V4.3.

Related Documents


Title of Documentation	Reference Number
SoMachine Programming Guide	EIO0000000067 (ENG) EIO0000000069 (FRE) EIO0000000068 (GER) EIO0000000071 (SPA) EIO0000000070 (ITA) EIO0000000072 (CHS)
Modicon LMC078 Motion Controller Hardware Guide	EIO0000001925 (ENG) EIO0000001926 (FRE) EIO0000001927 (GER) EIO0000001928 (SPA) EIO0000001929 (ITA) EIO0000001930 (CHS) EIO0000001932 (TUR)

Title of Documentation	Reference Number
Modicon LMC078 Motion Controller System Functions and Variables PLCSystem Library Guide	EIO0000001917 (ENG) EIO0000001918 (FRE) EIO0000001919 (GER) EIO0000001920 (SPA) EIO0000001921 (ITA) EIO0000001922 (CHS) EIO0000001924 (TUR)
SoMachine - Motion Control Library Guide	EIO0000002221 (ENG) EIO0000002222 (GER) EIO0000002223 (CHS)
Modicon TM5 / TM7 Flexible System - System Planning and Installation Guide	EIO0000000426 (ENG) EIO0000000427 (FRE) EIO0000000428 (GER) EIO0000000429 (SPA) EIO0000000430 (ITA) EIO0000000431 (CHS)
Modicon TM5 Expansion Modules Configuration Programming Guide	EIO0000000420 (ENG) EIO0000000421 (FRE) EIO0000000422 (GER) EIO0000000423 (SPA) EIO0000000424 (ITA) EIO0000000425 (CHS)
Modicon TM7 Expansion Blocks Configuration Programming Guide	EIO0000000880 (ENG) EIO0000000881 (FRE) EIO0000000882 (GER) EIO0000000883 (SPA) EIO0000000884 (ITA) EIO0000000885 (CHS)
SoMachine Modbus and ASCII Read/Write Functions PLCCommunication Library Guide	EIO0000000361 (ENG) EIO0000000362 (FRE) EIO0000000363 (GER) EIO0000000364 (SPA) EIO0000000365 (ITA) EIO0000000366 (CHS)
SoMachine Modem Functions Modem Library Guide	EIO0000000552 (ENG) EIO0000000491 (FRE) EIO0000000492 (GER) EIO0000000493 (SPA) EIO0000000494 (ITA) EIO0000000495 (CHS)


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

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

Product Related Information

 WARNING
<p>LOSS OF CONTROL</p> <ul style="list-style-type: none"> ● 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. <p>Failure to follow these instructions can result in death, serious injury, or equipment damage.</p>

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
<p>UNINTENDED EQUIPMENT OPERATION</p> <ul style="list-style-type: none"> ● Only use software approved by Schneider Electric for use with this equipment. ● Update your application program every time you change the physical hardware configuration. <p>Failure to follow these instructions can result in death, serious injury, or equipment damage.</p>

Terminology Derived from Standards

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

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

Among others, these standards include:

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

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

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

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

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

Chapter 1

About the Modicon LMC078 Motion Controller

Introduction

This chapter provides information about the Modicon LMC078 Motion Controller and devices that SoMachine can configure and program.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
About the Modicon LMC078 Motion Controller	16
Distributed I/O Architecture	18

About the Modicon LMC078 Motion Controller

Overview

The Schneider Electric Modicon LMC078 Motion Controller (LMC078CECS20T) is a controller with various powerful features. It can control a wide range of applications.

The Modicon LMC078 Motion Controller centrally implements the controller and motion functions. A Modicon LMC078 Motion Controller synchronizes, coordinates, and creates the motion functions of a machine for a maximum of 24 axes (synchronized in as little as 4 ms).

This controller is the optimized solution for axis positioning using the SoMachine software platform, which includes embedded automation functions and an ergonomic interface for axis configuration. Combined with Lexium 32S servo drives, this lets you design and commission your applications.

For more information about Lexium 32S servo drives, refer to the LXM32S Product Manual.

The software configuration is described in the SoMachine Programming Guide and in the LMC078 Motion Controller Programming Guide (*see page 9*).

Key Features

The SoMachine software supports the following IEC61131-3 programming languages for use with these controllers:

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

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

The LMC078 Motion Controller supports the following fieldbuses:

- With embedded communication interfaces:
 - CANopen Master/Slave
 - Sercos III
 - Ethernet TCP/IP
 - Serial Line
- With optional communication modules:
 - EtherNet/IP Adapter/Scanner
 - PROFIBUS DP Slave

The LMC078 Motion Controller supports the following I/O types:

- Master encoder input
- Embedded I/Os
 - Digital I/Os
 - Advanced digital inputs (touchprobe and interrupt inputs)
- Distributed I/Os on CANopen and Sercos fieldbuses (TM5/TM7 modules)

Performance

The LMC078 Motion Controller has the following performance:

- Up to 8 axes with a minimum synchronization time of 1 ms
- Up to 16 axes with a minimum synchronization time of 2 ms
- Up to 24 axes, with a minimum synchronization time of 4 ms (available with product hardware version greater than or equal to RS02).
- Minimum task cycle time (not for motion): 250 μ s

To display the hardware version, either:

1. Display the configuration parameters (*see page 76*) of the controller.
2. Verify that the first 2 characters of the `HW_Code` parameter are “0” and “2” respectively.

or:

1. Consult the LC Display of the controller.
2. Use the menu buttons to display the `HwCode` menu item.
3. Verify that the first 2 characters of the `HwCode` parameter are “0” and “2” respectively.

Example `HW_Code` or `HwCode` parameter for hardware version RS02:

0224013000000000

Distributed I/O Architecture

Introduction

The LMC078 Motion Controller offers the possibility of creating distributed I/O islands via:

- Sercos fieldbus with TM5 fieldbus interface (TM5NS31)
- CANopen fieldbus with TM5 fieldbus interface (TM5NC31) or TM7 fieldbus interface (TM7NCOM•••)

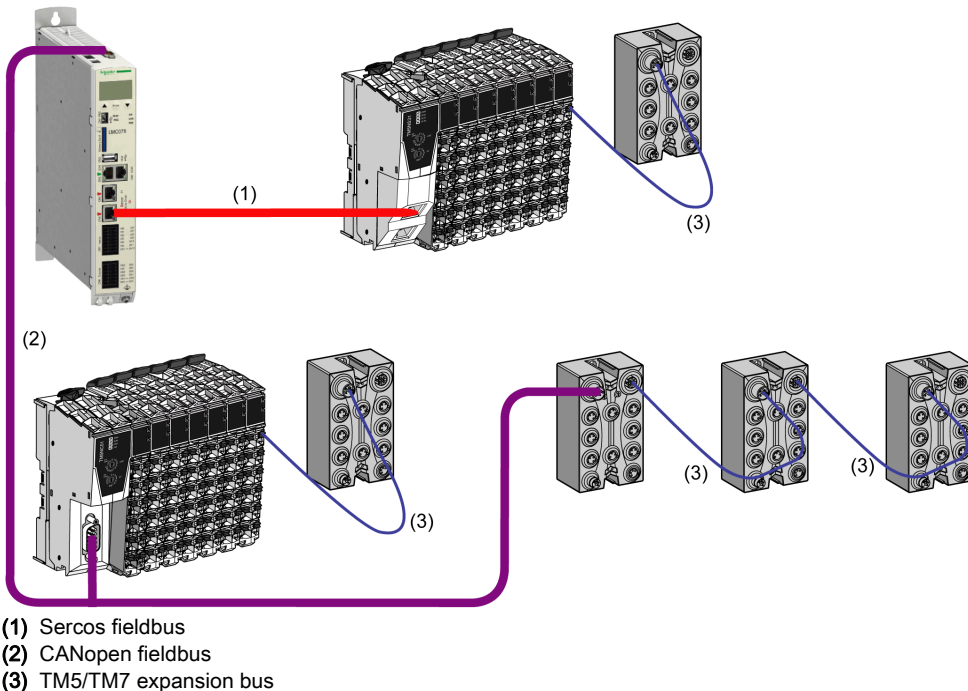
LMC078 Motion Controller Distributed Architecture

Optimized remote configuration and flexibility are provided by the association of:

- LMC078 Motion Controller
- TM5 and/or TM7 fieldbus interface
- TM5 and/or TM7 expansion modules

Application requirements determine the architecture of your LMC078 Motion Controller configuration.

This illustration presents a distributed configuration on Sercos and CANopen fieldbuses:



For more information on TM5 and TM7 expansion bus, refer to TM5 / TM7 Distributed I/Os Architecture (see *Modicon TM5 / TM7 Flexible System, System Planning and Installation Guide*).

Chapter 2

How to Configure the Controller

How to Configure the Controller

Introduction

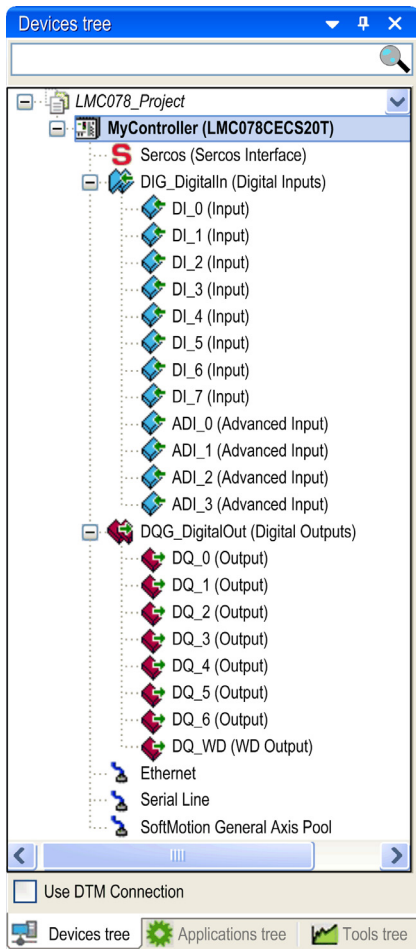
First, create a new project or open an existing project in the SoMachine software.

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

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

Devices Tree

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



Item	Use to Configure...
Sercos	Embedded Sercos III interface.
DIG_DigitalIn	Embedded digital inputs of the motion controller.
DQG_DigitalOut	Embedded digital outputs of the motion controller.

Item	Use to Configure...
Ethernet	Embedded Ethernet and serial line communications interfaces.
Serial Line	
SoftMotion General Axis Pool	SoftMotion devices (Virtual axis configuration).

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.
- Manage libraries.
- Access to the **Device addressing** tool (*see page 209*).
- Access to the **Message logger** tool (*see page 256*).
- Add CNC programs.

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 SoMachine Programming Guide.

Modicon LMC078 Motion Controller

When you select a Modicon LMC078 Motion Controller for your application, SoMachine loads the following libraries:

Library name	Description
SystemConfiguration	The content of this library is only used by SoMachine to create the driver function blocks instances.
LMC078 PLCSystem (<i>see Modicon LMC078 Motion Controller, System Functions and Variables, PLCSystem Library Guide</i>)	Contains functions and variables to get information and send commands to the controller system.
SystemConfigurationItf	Contains interfaces which manage the different properties of the system objects (controller, drive, and power supply).
IoStandard	CmploMgr configuration types, ConfigAccess , parameters and help functions: manages the I/Os in the application.
Standard	Contains functions and function blocks which are required matching IEC61131-3 as standard POU's for an IEC programming system. Link the standard POU's to the project (standard.library).
SM3_Basic	Contains functions for SoftMotion basic management, for more information refer to the SoMachine online help, chapter <i>CoDeSys Libraries / SoftMotion Libraries</i> .
SM3_CNC	Contains functions for SoftMotion CNC management, for more information refer to the SoMachine online help, chapter <i>CoDeSys Libraries / SoftMotion Libraries</i> .
LMC078 Sercos3 (<i>see page 199</i>)	Contains functions and variables to read/write data and send commands via the Sercos interface.

Library name	Description
Util	Contains functions for analog monitors, BCD conversions, bit/byte functions, controller data types, function manipulators, mathematical symbols, and signals.
CAA Device Diagnosis	This library offers functions and interfaces for the implementation of a simple but high-performance diagnostic functionality. The library defines methods which provide access to the required information for each device and each fieldbus.
CDS_MemMan	Memory manager library.

Chapter 4

Supported Standard Data Types

Introduction

This chapter provides the different IEC Data types supported by the Controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Supported Standard Data Types	26
Parameter Types	27

Supported Standard Data Types

Supported Standard Data Types

The controller supports the following IEC data types:

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

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

Parameter Types

Parameter Types

This table describes the controller parameter types:

Type ⁽¹⁾	Online editable	Offline editable	Fontcolor (2)	Properties	Read value	Write value
ER	Yes	Yes	Black	Input with user initialization. Transfer only after reset.	Quick memory access.	Not used.
ED	Yes	Yes	Black	Input with user initialization. Transfer directly after the change.	Quick memory access.	Quick memory access.
ED	Yes	No	Gray	Input with automatic initialization to default value. Transfer directly after the change. Can only be changed online by SoMachine.	Quick memory access.	Quick memory access.
EF	Yes	Yes	Black	Input with user initialization. Transfer directly after the change.	Quick memory access.	Functional access (internal calculation required).
EF	Yes	No	Gray	Input with automatic initialization. Transfer directly after the change. Can only be changed online by SoMachine.	Quick memory access.	Functional access (internal calculation required).
ES	Yes	No	Gray	Input. Transfer directly after the change. Can only be changed online by SoMachine.	Quick memory access.	Communication via Sercos. Delay of the caller (typically 10...100 ms).
AK	No	No	Gray	Output. Constant value.	Quick memory access.	Not possible.
AD	No	No	Gray	Output. Dynamic value.	Quick memory access.	Not possible.

Type ⁽¹⁾	Online editable	Offline editable	Fontcolor ⁽²⁾	Properties	Read value	Write value
AF	No	No	Gray	Output. Dynamic value.	Functional access (internal calculation required).	Not possible.
AS	No	No	Gray	Output. Dynamic value.	Communication via Sercos. Delay of the caller (typically 10...100 ms).	Not possible.

⁽¹⁾ The parameter type is displayed in the **Description** column of the controller **Configuration** screens (controller parameters (*see page 76*), embedded I/O parameters (*see page 92*), encoder parameters (*see page 100*), Sercos parameters (*see page 203*)).

⁽²⁾ The font color is the color of the parameter displayed in the **Configuration** screens. If the parameter is displayed in a black font, it is editable offline.

Sercos Reset Parameter

Sercos reset parameters are not accepted immediately following the input but only after the next Sercos run-up (Phase 0 -> Phase 4).

This table lists the Sercos reset parameters of the controller:

Parameter	Group	Acceptance	Parameter type
WorkingMode	Identification	Phase 2 -> Phase 3	EF
IdentificationMode	Identification	Phase 2 -> Phase 3	EF
ConfiguredTopologyAddress	Identification	Phase 2 -> Phase 3	EF
ConfiguredApplicationType	Identification	Phase 2 -> Phase 3	EF
ConfiguredSercosAddress	Identification	Phase 2 -> Phase 3	EF
ConfiguredSerialNumber	Identification	Phase 2 -> Phase 3	EF

Chapter 5

Memory Mapping

Introduction

This chapter describes the memory maps and sizes of the different memory areas in the Modicon LMC078 Motion 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
Controller Memory Organization	30
RAM Memory Organization	31
Flash Memory Organization	33
USB Memory Key	34

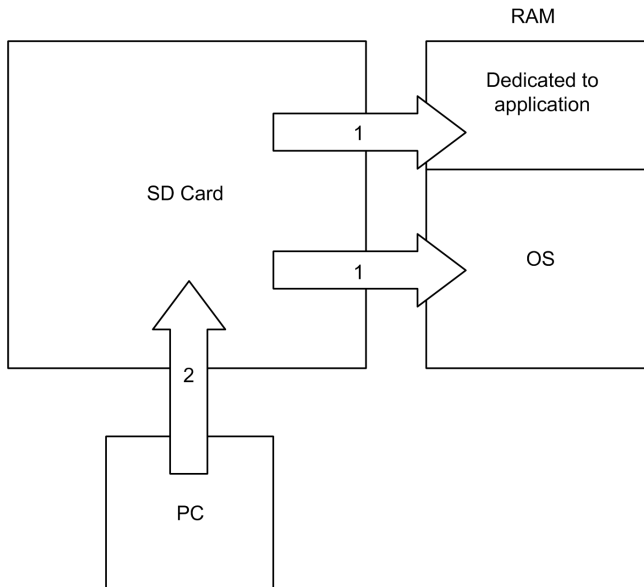
Controller Memory Organization

Introduction

The controller memory is composed of 3 types of physical memory:

- The SD card (*see page 33*) memory contains files (application, configuration files, and OS).
- The RAM (Random Access Memory) (*see page 31*) is used for application and OS execution.
- The NVRAM contains retained and persistent retained variables.

Files Transfers in Memory



Item	Controller state	File transfer events	Connection	Description
1	–	Initiated automatically at power-on and reboot	Internal	File transfer from SD card to RAM. The content of the RAM is overwritten.
2	All states	Initiated by user	Ethernet or USB programming port	Files can be transferred via: <ul style="list-style-type: none"> • FTP server (<i>see page 176</i>) • SoMachine

NOTE: Files on the SD card can be read, written, or erased, depending on the controller state. The modification of files in the SD card does not affect a running application. Any changes to files in the SD card are taken into account at the next reboot.

RAM Memory Organization

Introduction

This section describes the RAM (Random Access Memory) size for different areas of the Modicon LMC078 Motion Controller.

Memory Mapping

The RAM (512 Mbytes) is composed of two areas:

- OS memory
- Dedicated application memory

The NVRAM (128 Kbytes) is composed of two areas:

- Retained variables
- Persistent retained variables

The NVRAM containing persistent and retained variables is preserved by an internal battery during power outages or when the controller is powered off.

Declaring variables as **Persistent** increases the cycle time of the controller by approximately 0.2 ms per 1000 variables.

Persistent variables are saved in NVRAM and are preserved by an internal battery during power outages or when the controller is powered off.

Configure the minimum number of persistent variables required for your application to help avoid degradation of controller performance.

This table describes the dedicated application memory:

Area	Element	Size (byte)
System area	Input (%I)	Minimum 65536
	Output (%Q)	Minimum 65536
	Memory (%M)	Minimum 65536
User area	Symbols	Minimum 1048576
	Variables	
	Application	
	Libraries	

NOTE: The defined sizes are by default allocated during the boot up phase. Dynamic memory allocation is also possible.

This table describes the NVRAM memory:

Area	Size (byte)
Retained variables	1000...84501
Retained and retained persistent variables	1000...84501

NOTE: To verify the memory usage of each area, right-click on the controller node in the **Devices tree** and click **Device Memory Info**.

System Variables

For more information on system variables, refer to the *LMC078 PLCSystem Library Guide*.

Memory Addressing

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

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

Example of overlap of memory ranges:

%MD0 contains %MB0...%MB3, %MW0 contains %MB0 and %MB1, %MW1 contains %MB2 and %MB3.

Flash Memory Organization

Introduction

The SD card contains the file system used by the controller.

You can also use the SD card as a mass storage media for your files.

File Organization

This table presents the file organization of the SD card:

Directory	File	Content
\	Application.app Application.crc	Application
\ESystem\	bootc4.sys	Boot loader
	sysc3.sys	VxWorks kernel and firmware
	sysc3.cfg	Lzs2 component configuration
\ESystem\FBUSFW\	NETX100-BSL.bin	NetX boot loader
	cifXrcX.nxf	NetX basic firmware
	DPS_XC0.nxo	NetX firmware for PROFIBUS DP slave
	DPS_XC2.nxo	
	nx100eis.nxo	NetX firmware for EtherNet/IP adapter
	nx100eim.nxo	NetX firmware for EtherNet/IP scanner
	nx100ecs.nxo	NetX firmware for EtherCAT slave
\ESystem\FirmwareDatabase\D3\	TM5NS31_V245.fw	Firmware of the TM5NS31 Sercos interface module
\ESystem\Languages\	english.xml	LCD language
\romfs\	Prsnlty.ini	EtherBrick configuration

NOTE: You can use the functions of the **CAA File** library to access to the files of the SD card. For more information on the function blocks of this library, refer to the *CoDeSys libraries* topic in the SoMachine online help.

USB Memory Key

Introduction

The USB memory key is used as a mass storage media for your files. It can be accessed via the FTP or the application.

NOTE: You can use the functions of the **CAA File** library to access to the USB memory key. For more information on the function blocks of this library, refer to the *CoDeSys libraries* topic in the SoMachine online help.

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
- Event
- External event
- Status

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Maximum Number of Tasks	36
Task Configuration Screen	37
Task Types	39
Motion Task	42
System and Task Watchdogs	45
Task Priorities	46
Default Task Configuration	49

Maximum Number of Tasks

Maximum Number of Tasks

The Modicon LMC078 Motion Controller supports up to 1000 application program tasks.

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 screenshot shows the 'MAST x' Configuration window. It is divided into four main sections:

- Priority (0..31):** A text input field containing the value '1'.
- Type:** A dropdown menu set to 'Cyclic' and an 'Interval (e.g. t#200ms):' field containing '#20ms'.
- Watchdog:** A section with a checked 'Enable' checkbox, a 'Time (e.g. t#200ms):' field containing '100' with a unit dropdown set to 'ms', and a 'Sensitivity:' field containing '1'.
- POU Table:** A table with a toolbar above it. The toolbar includes 'Add Call', 'Remove Call', 'Change Call', 'Move Up', 'Move Down', and 'Open POU'. The table has two columns: 'POU' and 'Comment', and is currently empty.

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

Field Name	Definition
Priority	<p>Configure the priority of each task with a number from 0 to 31 (0 is the highest priority, 31 is the lowest).</p> <p>Only one task at a time can be running. The priority determines when the task will run:</p> <ul style="list-style-type: none"> ● a higher priority task will pre-empt a lower priority task ● tasks with same priority will run in turn (2 ms time-slice) <p>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 unpredictable. For important safety information, refer to Task Priorities (see page 46).</p>
Type	<p>These task types are available:</p> <ul style="list-style-type: none"> ● Cyclic ● Event (see page 40) ● External (see page 40) ● Status (see page 40)
Watchdog	<p>To configure the watchdog (see page 45), define these 2 parameters:</p> <ul style="list-style-type: none"> ● Time: enter the timeout before watchdog execution. ● Sensitivity: defines the number of expirations of the watchdog timer before the controller stops program execution and enters a HALT state.
POUs	<p>The list of POUs (see SoMachine, Programming Guide) (Programming Organization Units) controlled by the task is defined in the task configuration window:</p> <ul style="list-style-type: none"> ● 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. ● POUs are executed in the order shown in the list. To move the POUs in the list, select a POU and use the command Move Up or Move Down. <p>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.</p>

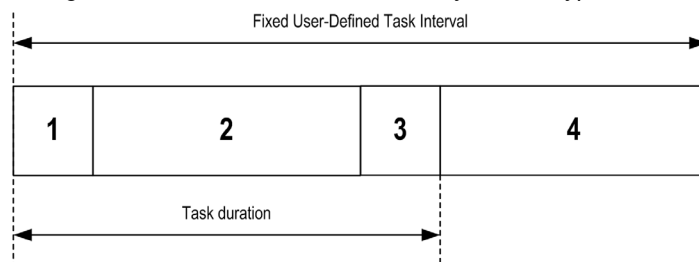
Task Types

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. The output values of distributed I/O modules are not yet written to the physical outputs during this operation. The embedded output values are immediately written to the physical outputs.
3. **Write outputs of distributed I/O modules:** 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 LMC078 Motion Controller Settings (*see page 88*).
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 without executing other lower priority tasks or any system processing. This affects the execution of all tasks.


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.)

The minimum cycle time for cyclic task is 250 μ s. The configured cycle time has to be a multiple of 250 μ s (for example, 500 μ s, 750 μ s, 1 ms, and so on).

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 <code>my_Var</code> variable.

NOTE: The maximum frequency admissible for the event triggering an Event task is 100 Hz.

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 a rising edge on an advanced input (`DI8...DI11`). To associate the **INIRQ1** event to an External event task, select it from the **External** event drop-down list on the **Configuration** tab.


There are up to 6 types of events that can be associated with an External event task:

- **INIRQx**: rising edge on an advanced input
- **RTP_READ**: real-time process after real-time data read
- **RTP_MENC**: real-time process after master encoder
- **RTP_LENC**: real-time process after logical encoder
- **RTP_AXIS**: real-time process after computing of **RefValues** function blocks
- **MDT_WRITE_ACCESS**: used to trigger the Motion task (write access to MDT (Sercos Master Data Telegram (*see page 200*)))

Status Task

This type of task is event-driven and is initiated by a program variable. It starts if the boolean variable associated to the trigger event is TRUE unless pre-empted by a higher priority task. In that case, the Status 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 a Status task, proceed as follows:

Step	Action
1	Double-click the TASK in the Applications tree .
2	Select Status 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 <code>my_Var</code> variable.

Motion Task

Introduction

This section presents the characteristics of the Motion task and provides information on the performance possible when using an optimally configured motion system. The Motion task is created automatically with the **External** event name of **MDT_WRITE_ACCESS**. This mechanism allows a synchronization of the Motion task with the bus cycle of the Sercos bus.

The parameter **Priority (0...31)** is ignored. The task is executed with the priority of the real-time process (higher than IEC task priority 0).

The **SR_Motion** POU is automatically created and attached to the Motion task.

NOTE:

An adequately defined cycle time meets both of the following requirements:

- The program processing defined in your Motion task must have enough time to execute in full. Test the execution time of your Motion task under all operating conditions to determine this value.
- The Sercos **Cycle Time** (*see page 203*) must be of sufficient duration to allow the physical exchange of all data between the controller and all of the configured devices.

If you do not configure a sufficient **Cycle Time** this can result in a system watchdog exception or even a loss of synchronization for the controlled devices. For example, an insufficient **Cycle Time** may result in the detection of the loss of the Sercos master for all controlled devices. In this case, devices detecting a loss of the Sercos master assume their programmed fallback states. Always confirm that your **Cycle Time** is sufficient to allow full execution of the Motion task and a complete physical exchange of all data before placing your system into service.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Calculate the required minimum cycle time for your task processing and physical data exchange.
- Define a task (software) watchdog for the Motion task with a watchdog period slightly larger than the **Cycle Time** defined for the Sercos interface.
- Thoroughly test your Sercos system under normal and exception state conditions before placing your system into service.

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

This picture displays the settings for the Motion task:

POU	Comment
SR_Motion	

NOTE: Do not delete the Motion task or change its Name, Type, or External Event attributes. If you do so, SoMachine does not detect an error when you build the application, but an error will be returned by the Motion Library as soon as you attempt to use the application.

Motion Task Programming Requirements

You must use the Motion task to manage every aspect of programming related to the Sercos bus and its connected motion devices such as drive controllers.

This includes:

- Local inputs used to acquire motion events
- Encoder inputs used to acquire motion events
- Task processing for all motion functions (Motion task POU, and so on)
- Encoder outputs configured to respond to motion events
- Local outputs configured to respond to motion events

WARNING

UNINTENDED EQUIPMENT OPERATION

Use the Motion task to manage all motion-related inputs, outputs, task processing, and Sercos communications.

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

Motion Task Performance

The Modicon LMC078 Motion Controller is capable of achieving high performance. The controller can manage up to:

- 8 axes with a minimum synchronization time of 1 ms
- 16 axes with a minimum synchronization time of 2 ms
- 24 axes with a minimum synchronization time of 4 ms (available with hardware version \geq RS02). For further information, refer to Performance (*see page 17*).

The subset of functions that can be used and still allow you to achieve similar performance (with an efficiently-written application) are:

- Virtual axes
- Relative and absolute positioning
- Speed control
- Cam profiles
- Electronic gearing
- Linear and circular interpolation by using G code

System and Task Watchdogs

Introduction

Two types of watchdog functionality are implemented for the Modicon LMC078 Motion Controller:

- **System Watchdog:** This watchdog is defined in and managed by the controller firmware. This is 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 Watchdog

The system watchdog is managed by the controller firmware and is therefore sometimes referred to as hardware watchdog in the SoMachine online help. When the system watchdog exceeds its threshold conditions, an error is detected and is displayed on the controller.

If the RTP (Real Time Process) is not triggered during an interval of 100 ms, a system watchdog is detected. The controller enters the HALT state, a controller reboot is required to get back into RUNNING mode.

NOTE: The system watchdog is not configurable by the user.

The digital output 7 (DQ_WD) can be configured as watchdog output controlled by the system watchdog (Watchdog Output Configuration).

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 (Diagnostic messages (*see page 262*)).

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.

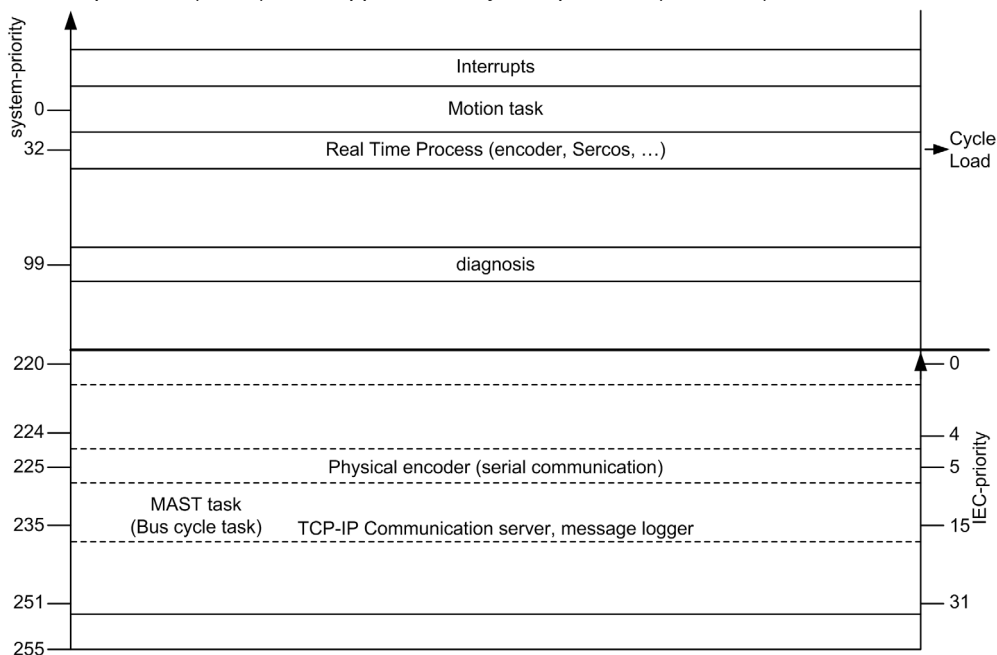
 **WARNING**

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.

The IEC priorities (0...31) are mapped to the system priorities (220...251):



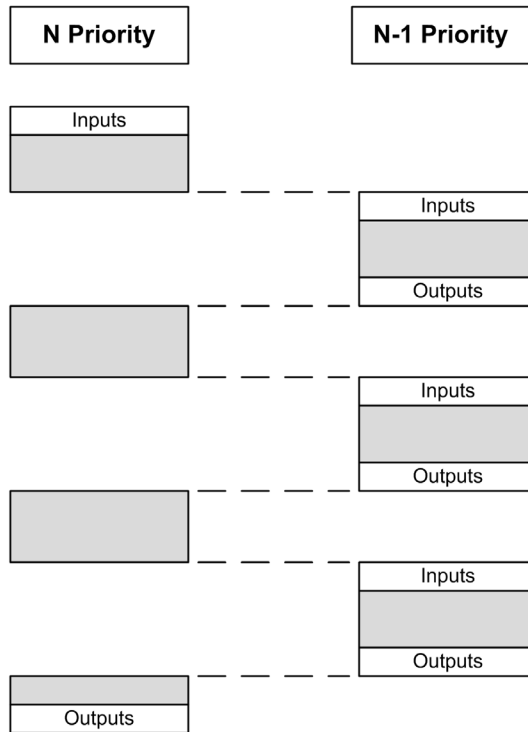
The Motion task (attached to the event **MDT_WRITE_ACCESS**) is created with the system priority 0 triggered by the real time process (system priority 32).

Task Priority Suggestions

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

Task Preemption Due to Task Priorities

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



NOTE: If the same input is used in different tasks the input image may change during the task cycle of the lower priority task.

To improve the likelihood of proper output behavior during multitasking, a message is displayed if outputs in the same byte are used in different tasks.

The embedded outputs are updated immediately after their writing and not at the end of the task cycle.

The embedded inputs are updated immediately after their state modification and not only at the beginning of the task cycle.

 **WARNING**

UNINTENDED EQUIPMENT OPERATION

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

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

Default Task Configuration

Default Task Configuration

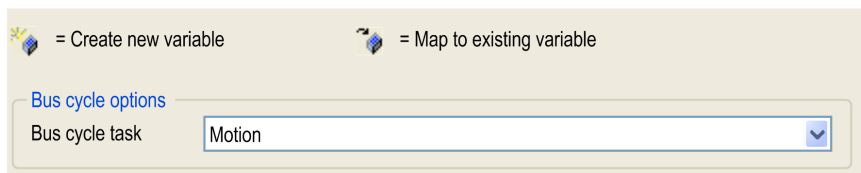
For the Modicon LMC078 Motion Controller:

- The MAST task is automatically created by default as a cyclic task. Its preset priority is medium (15), its preset interval is 10 ms, and its task watchdog service is activated with a time of 10 ms and a sensitivity of 5. Refer to Task Priorities (*see page 46*) for more information on priority settings. Refer to System and Task Watchdogs (*see page 45*) for more information on watchdogs.
- A Motion task is created automatically. This task is declared as an external event task, and reduces the number of external event tasks you can configure for other operations by one. The Motion task is executed with the priority of the real-time process (the priority parameter is ignored). Refer to Task Priorities (*see page 46*) for more information.
- A `SR_Motion` POU is created automatically and called by the Motion task.

NOTE: Do not delete the Motion task or change its **Name**, **Type**, or **External event** attributes. If you do so, SoMachine does not detect an error when you build the application, but an error will be returned by the Motion Library as soon as you attempt to use the application.

The default **Cycle Time** is 1 ms (Sercos Interface Configuration (*see page 203*)).

The **Bus cycle options** in the **I/O Mapping** tab of all devices that are controlled by the motion application have to be set to the Motion task:



NOTE: You may consider putting all the code inside this task if the performance of the controller, the size of the program and the functions executed in the program permit this.

You will need to monitor the execution time of this task during development and commissioning of the machine. The parameters `AvailableLoad` (controller object), `CycleLoad` (Sercos object) and `RTBWriteRes` (Sercos object) can be used to estimate the load that the code is causing in the real-time process driving the Sercos bus. If you do not monitor the execution time, it could result in the delaying the set points for drives and output values of I/Os. The diagnostic message `8507 SERCOS write cycle overflow` indicates this situation.

Other Tasks

If code needs to be moved to other tasks, you should create additional tasks of **Cyclic** type and select a priority in the range of 16 and 31.

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	52
7.2	Controller States Description	56
7.3	State Transitions and System Events	60

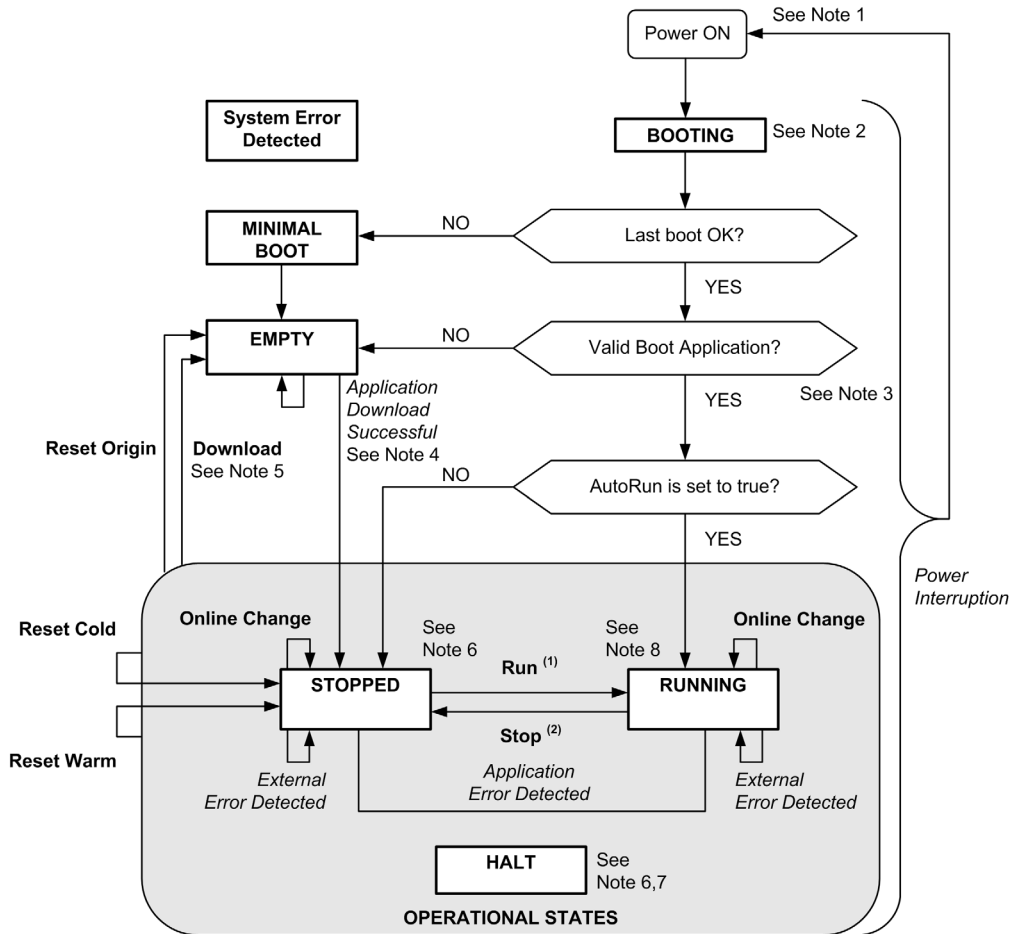
Section 7.1

Controller State Diagram

Controller State Diagram

Controller State Diagram

This diagram describes the controller operating mode:



Legend:

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

(1) For details on STOPPED to RUNNING state transition, refer to Run Command (*see page 64*).

(2) For details on RUNNING to STOPPED state transition, refer to Stop Command (*see page 64*).

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 61*) for further details.

Note 2

There is a 4-5 second delay between entering the BOOTING state and the LED indication of this state. The boot process can take up to 60 seconds under normal conditions. The outputs will assume their initialization states.

Note 3

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 verify 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 66*).

Note 4

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 SD card.

Note 5

The default behavior after downloading an application program is for the controller to enter the STOPPED state irrespective of the **AutoRun** 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. 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, 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 6

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 56*) for further details.

Note 7

To exit the HALT state it is necessary to issue one of the reset commands (reset warm, reset cold, reset origin), download an application or cycle power.

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

Note 8

The RUNNING state has two exception conditions.

They are:

- RUNNING with external error detected: this exception condition is indicated by the **STS** status LED, which displays solid green with one red flash. You may exit this state by clearing the external error. No controller commands are required.
- RUNNING with breakpoint: this exception condition is indicated by the **STS** status LED, which displays three green flashes. Refer to Controller States Description (*see page 56*) for further details.


Section 7.2

Controller States Description

Controller States Description

Introduction

This section provides a detailed description of the controller states.

 WARNING
UNINTENDED EQUIPMENT OPERATION
<ul style="list-style-type: none"> • 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, 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 with the `FC_DiagMsgRead` function of the LMC078 PLCSystem library (see *Modicon LMC078 Motion Controller, System Functions and Variables, PLCSystem Library Guide*) or in SoMachine with the message logger of the controller.

Controller States Table

The following table describes the controller states:

Controller state	Description	STS LED
BOOTING	The controller executes the boot firmware and its own internal self-tests. It then verifies the checksum of the firmware and user applications. It does not execute the application nor does it communicate.	Green/red flashing
BOOTING after detection of a <i>system error</i>	This state is the same as the normal BOOTING state except that a flag is set to make it appear as if no boot application is present and the LED indications are different.	Rapid red flashing
MINIMAL BOOT	There is not a valid firmware file present in the SD card. The controller does not execute the application. Refer to Upgrading Modicon LMC078 Motion Controller Firmware (see page 245).	Red flashing

Controller state	Description	STS LED
EMPTY	There is no application present or an invalid application.	Single green flash
EMPTY after detection of a <i>system error</i>	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.	Rapid red flashing
RUNNING	The controller is executing a valid application.	Green
RUNNING with breakpoint	This state is the same as the RUNNING state with the following exceptions: <ul style="list-style-type: none"> • 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.	3 green flashes
RUNNING with detection of an <i>external error</i>	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 58) for an explanation of the behavior of outputs and field buses in this state.	Green flashing
STOPPED with detection of an <i>external error</i>	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: <ul style="list-style-type: none"> • Embedded outputs assume their Initialization values (see page 61). • CAN bus behaves as if the Update IO while in stop option was not selected when managed by a task responsible for the application error. Else, CAN bus behavior follows the actual setting. • The LED indications are different. 	Single red flashing

Details of the STOPPED State

The following statements are true for the STOPPED state:

- Ethernet, serial (Modbus, ASCII, and so on), and USB communication services remain operational and commands written by these services can continue to affect the application, the controller state, and the memory variables.
- All outputs initially assume their configured default state (**Keep current values** or **Set all outputs to default**) or the state dictated by output forcing if used. 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: Commands received by Ethernet, serial, USB, and CAN communications can continue to write to the memory variables. Changes to the %Q output memory variables are written to the physical outputs.

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 **Behavior for outputs in Stop** field is set to **Keep current values**, the TPDOs continue to be issued with the last actual values.
- If the **Behavior 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 variables are frozen at their last values.
- The task processing operation is not executed.
- The write outputs operation ceases. The %Q output memory variables can be updated via the Ethernet, serial, and USB connections. However, the physical outputs are unaffected and retain the state specified by the configuration options.

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	61
Commanding State Transitions	64
Error Detection, Types, and Management	69
Remanent Variables	70

Controller States and Output Behavior

Introduction

The Modicon LMC078 Motion 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**
- **Execute program**
- hardware **Initialization Values**
- software **Initialization Values**
- **Output Forcing**

Managed by Application Program

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

Keep Current Values

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

This output behavior applies in the STOPPED controller state. It also applies to CAN bus in the HALT controller state. Outputs are set to and maintained in their current state, although the details of the output behavior vary greatly depending on the setting of the **Update I/O while in stop** option and the actions commanded via configured fieldbusses. Refer to Controller States Description (*see page 56*) for more details on these variations.

Set All Outputs to Default

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

This output behavior applies when the application is going from RUN state to STOPPED state or if the application is going from RUN state to HALT state. It also applies to CAN bus in the HALT controller state. Outputs are set to and maintained in their current state, although the details of the output behavior vary greatly depending on the setting of the **Update I/O while in stop** option and the actions commanded via configured fieldbusses. Refer to Controller States Description (*see page 56*) for more details on these variations.

Execute Program

You determine the outputs behavior by a program available within the project.

Select this option by choosing **Execute program** in the **Behavior for outputs in Stop** drop-down menu of the **PLC settings** subtab of the **Controller Editor**.

Click the button ... and select a **POU** with the **Input Assistant**.

This program is executed when the controller is in STOPPED state.

Hardware Initialization Values

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

In the initialization state, outputs are set to 0.

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:

- AutoRun parameter is set to 1 in the **Configuration** tab (*see page 77*): automatic start after booting.
- IECProgramStateSet parameter is set to 1 in the **Configuration** tab (*see page 77*).
- SoMachine Online Menu: Select the **Start** command.
- **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 52*) 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:

- IECProgramStateSet parameter is set to 0 in the **Configuration** tab (*see page 77*).
- SoMachine Online Menu: Select the **Stop** command.
- **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 52*) 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.
- Using the `FC_PrgResetAndStart` function of the LMC078 PLCSystem Library (*see Modicon LMC078 Motion Controller, System Functions and Variables, 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. All fieldbus communications are stopped and then restarted after the reset is complete.
8. All I/O are reset to their initialization values.

For details on variables, refer to Remanent Variables (*see page 70*).

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.

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. All fieldbus communications are stopped and then restarted after the reset is complete.
8. All I/O are reset to their initialization values.

For details on variables, refer to Remanent Variables (*see page 70*).

Reset Origin

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

Starting Conditions: RUNNING, STOPPED, or HALT states.

Methods for Issuing a Reset Origin Command:

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

Effects of the Reset Origin Command:

1. The application stops.
2. Forcing is erased.
3. All user files (Boot application, data logging, Post Configuration) are erased.
4. Diagnostic indications for errors are reset.
5. The values of the retain variables are reset.
6. The values of the retain-persistent variables are reset.
7. All non-located and non-remanent variables are reset.
8. All fieldbus communications are stopped.
9. All I/O are reset to their initialization values.

For details on variables, refer to Remanent Variables (*see page 70*).

Reboot

Effect: Commands a reboot of the controller.

Starting Conditions: Any state.

Methods for Issuing the Reboot Command:

- Power cycle
- Using the `FC_SysReset` function of the LMC078 PLCSystem Library (*see Modicon LMC078 Motion Controller, System Functions and Variables, PLCSystem Library Guide*)

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:
 - the `AutoRun` parameter is set to 1, and if the controller was not in HALT state before the power cycle, and if the remanent variables are valid.
 - The Reboot was provoked by a script and:
 - the **Starting Mode** is set to **Start in run**, and if the Run/Stop input or switch 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.
 - b. The controller state will be STOPPED if:
 - The Reboot was provoked by a Power cycle:
 - the `AutoRun` parameter is set to 0.

- c. The controller state will be EMPTY if:
 - There is no boot application or the boot application is invalid, or
 - The reboot was provoked by specific System Errors.
 - d. The controller state will be INVALID_OS if there is no valid firmware.
2. Forcing is maintained if the boot application is loaded successfully. If not, forcing is erased.
 3. Diagnostic indications for errors are reset.
 4. The values of the retain variables are restored if saved context is valid.
 5. The values of the retain-persistent variables are restored if saved context is valid.
 6. All non-located and non-remanent variables are reset to their initialization values.
 7. All fieldbus communications are stopped and restarted after the boot application is loaded successfully.
 8. 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 70*).

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

NOTE: If you 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 SD card.

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 52*).

- FTP: Load Boot application file to the SD card using FTP. The updated file is applied at the next reboot.

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. All fieldbus communications are stopped and then any configured fieldbus of the new application is started after the download is complete.
9. 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 70*).

Effects of the FTP Download Command:

There are no effects until the next reboot. At the next reboot, the effects are the same as a reboot with an invalid context. Refer to Reboot (*see page 66*).

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	<p>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:</p> <ul style="list-style-type: none"> • A connected device reports an error to the controller. • The controller detects an error with an external device, for example, when the external device is communicating but not properly configured for use with the controller. • The controller detects an error with the state of an output. • The controller detects a communication interruption with a device. • The boot application in the SD card is not the same as the one in RAM. 	<p>RUNNING with External Error Detected Or STOPPED with External Error Detected</p>
Application Error	<p>An application error is detected when improper programming is encountered or when a task watchdog threshold is exceeded.</p>	HALT
System Error	<p>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.</p> <p>NOTE: There are some system errors that can be managed by runtime and are therefore treated like application errors.</p>	BOOTING → EMPTY

NOTE: Refer to the LMC078 PLCSystem library Guide (*see Modicon LMC078 Motion Controller, System Functions and Variables, PLCSystem Library Guide*) for more detailed information on diagnostics.

Remanent Variables

Overview

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

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

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

Action	VAR	VAR RETAIN	VAR GLOBAL PERSISTENT RETAIN
Online change to application program	X	X	X
Online change modifying the boot application ⁽¹⁾	–	X	X
Stop	X	X	X
Power cycle	–	X	X
Reset warm	–	X ⁽²⁾	X
Reset cold	–	–	X
Reset origin	–	–	–
Download of application program ⁽³⁾	–	–	X

X The value is maintained.
– The value is reinitialized.


(1) Retain variable values are maintained if an online change modifies only the code part of the boot application (for example, `a:=a+1; => a:=a+2;`). In all other cases, retain variables are reinitialized.

(2) For more details on VAR RETAIN, refer to Effects of the Reset warm Command ([see page 65](#)).

(3) If the application is downloaded via SD card, any existing persistent variables used by the application are reinitialized. If the application is downloaded using SoMachine, however, existing persistent variables maintain their values. In both cases, if the downloaded application contains the same persistent variables as the existing application, the existing retain variables maintain their values.

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?

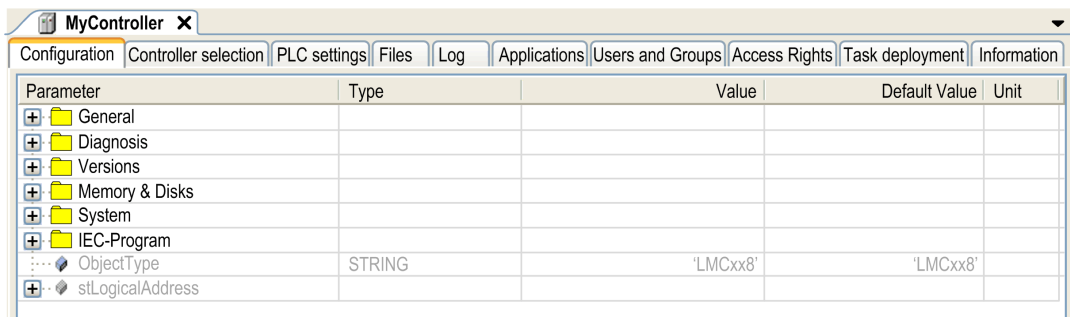
This chapter contains the following topics:

Topic	Page
Controller Parameters	74
Configuration Parameters	76
Controller Selection	86
PLC Settings	88

Controller Parameters

Controller Parameters

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



Tabs Description

Tab	Description	Restriction
Configuration	Access and configuration of the controller parameters.	–
Controller selection <i>(see page 86)</i>	<p>Manages the connection between the PC and the controller:</p> <ul style="list-style-type: none"> ● 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. <p>The controller list is detected through NetManage or through the Active Path based on the communication settings. To access the Communication settings, click Project → Project Settings... in the menu bar. For more information, refer to the SoMachine Programming Guide (<i>Communication Settings</i>).</p>	Online mode only
Applications	Presents the application running on the controller and allows removing the application from the controller.	Online mode only
Files	File management between the PC and the controller.	Online mode only
Log	View the controller log file.	Online mode only

Tab	Description	Restriction
PLC settings <i>(see page 88)</i>	Configuration of: <ul style="list-style-type: none"> ● application for I/O handling ● I/O behavior in stop ● bus cycle options 	–
Task deployment	Displays a list of I/Os and their assignments to tasks.	After compilation only
Users and Groups	The Users and Groups tab is provided for devices supporting online user management. It allows setting up users and access-rights groups and assigning them access rights to control the access on SoMachine projects and devices in online mode. For more details, refer to the SoMachine Programming Guide.	–
Access Rights	The Access Rights tab is provided for devices supporting online user management. It serves to grant or deny the currently defined user groups certain permissions, thus defining the access rights for users on files or objects (for example, an application) on the controller during runtime. For more details, refer to the SoMachine Programming Guide.	–
Information	Displays general information about the device (name, description, provider, version, image).	–

Configuration Parameters

Overview

This illustration presents the **Configuration** tab:

MyController X					
Configuration Controller selection PLC settings Files Log Applications Users and Groups Access Rights Task deployment Information					
Parameter	Type	Value	Default Value	Unit	
General					
Name	STRING(40)	"	"		
AutoRun	Enumeration of BOOL	no / 0	no / 0		
IP_SubNetMask	STRING(15)	'255.255.0.0'	'255.255.0.0'		
IP_Address	STRING(15)	'192.168.100.1'	'192.168.100.1'		
IP_Gateway	STRING(15)	'0.0.0.0'	'0.0.0.0'		
EthernetAdr	STRING(19)	"	"		
MsgFilter	DWORD(0..65535)	16#0000FFFF	16#0000FFFF		
ControllerReset	Enumeration of BOOL	true / 1	true / 1		
IOReset	Enumeration of DINT	after download or prg. reset / 2	after download or prg. reset / 2		
Diagnosis					
DiagClass	DINT				
DiagCode	DINT				
DiagSource					
DiagMsg	STRING(39)	"	"		
DiagExtMsg	STRING(14)	"	"		
MsgEntries	DINT				
FastTimer	UDINT			µs	
Timer1	DINT			ms	
Timer10	DINT			10 ms	
CycleLoad	DINT			%	
RTBReadRes	DINT			µs	
RTBWriteRes	DINT			µs	
PowerOK	Enumeration of BOOL	false / 0	false / 0		
SetRealTimeClock	DT	0	0		
RealTimeClock	DT	0	0		
BatteryLowWarningDelay	REAL(0..50)	0.0	0.0	h	
Versions					
FW_Version	STRING(28)	"	"		
ControllerType	STRING(254)	"	"		
ControllerType1	STRING(254)	"	"		
HW_Code	STRING(20)	"	"		
SerialNumber	STRING(20)	"	"		
Memory & Disks					
RamDiskSize	DINT(128..4096)	1024	1024	KByte	
RamDiskFree	DINT			KByte	
Diskfree	DINT			byte	
Memoryfree	DINT			byte	
System					
Systemticks	DINT(10..20000)	4000	4000		
EnableLoadEff	Enumeration of BOOL	off / 0	off / 0		
AvailableLoad	DINT	0	0	%	
AvailableLoadPeriod	DINT(1..2000)	100	100	ms	
RemoteCommunicationAccess	Enumeration of DINT	read/write/save / 2	read/write/save / 2		
ActivateFatalCrashReaction	Enumeration of BOOL	yes / 1	yes / 1		
IEC-Program					

NOTE: The parameters are also accessible in the application (`ObjectName.ParameterName`, for example `MyController.AvailableLoad`).

NOTE: The parameters are also accessible through communication protocols.

Parameters Description

This table describes the controller parameters for diagnostic and configuration:

Parameter	Access	Param . type	Data type	Value	Default value	Description
General						
Name	R/W(*)	EF	STRING	"	"	Symbolic name of the configuration object.
AutoRun	R/W	ER	BOOL Enum	no / 0 yes / 1	no / 0	<ul style="list-style-type: none"> ● 0 = Autorun is not activated. ● 1 = The program starts automatically after controller booting.
IP_SubNetMask	R	AF	STRING	255.255.0.0	255.255.0.0	Displays IP subnet mask.
IP_Address	R	AF	STRING	192.168.100.1	192.168.100.1	Displays IP address.
IP_Gateway	R	AF	STRING	0.0.0.0	0.0.0.0	Displays gateway address.
EthernetAddr	R	AF	STRING	"	"	Displays device-specific Ethernet address.
MsgFilter	R/W	EF	DWORD	0...FFFFh	FFFFh	<p>Configures the message logger filtering.</p> <p>You can distinguish 16 classes of message:</p> <ul style="list-style-type: none"> ● Bit 0: General system messages. ● Bit 1: Diagnostic messages. ● Bit 2: Program system function block. ● Bit 3: Fieldbus-specific information. ● Bit 4...11: Not used. ● Bit 12: Extended system messages. ● Bit 13-14: Not used.

Parameter	Access	Param . type	Data type	Value	Default value	Description
ControllerReset	R/W(*)	ED	BOOL Enum	FALSE / 0 TRUE / 1	TRUE / 1	To distinguish between a program reset and a controller reset. This parameter is set to 1 when the controller is reset. The parameter can be set to 0 using an application program. In this way, you can distinguish between a program reset and a controller reset. <ul style="list-style-type: none"> ● 0 = Program reset. ● 1 = Controller reset.
IOReset	R/W	EF	DINT Enum	0...3	2	Defines the reset modes of I/O areas: <ul style="list-style-type: none"> ● 0 = No reset. ● 1 = Reset after download. ● 2 = Reset after download or program reset. ● 3 = Reset after download, program reset, or program stop.
Diagnosis						
DiagClass	R	AD	DINT	-	-	Displays the diagnostic class (<i>see page 262</i>) in decimal code.
DiagCode	R	AD	DINT	-	-	Displays the diagnostic code (<i>see page 262</i>) in decimal code.
DiagSource	R	AD	ST_LogicalAddress	-	-	Specifies the source of the diagnostic (stLogicalAddress type)
DiagMsg	R	AD	STRING	-	-	Displays the diagnostic text.
DiagExtMsg	R	AD	STRING	-	-	Displays the extended diagnostic message.
MsgEntries	R	AF	DINT	-	-	Number of entries in the message logger.
FastTimer	R	AF	UDINT	-	-	Displays the timer in μ s. The timer is derived from the TSC (TimeStampCounter) of the CPU. The TSC is a 64-bit counter that runs with the CPU cycle. The counter runs endlessly from 0 to $2^{31}-1$. The end value corresponds to approximately 71 min.

Parameter	Access	Param . type	Data type	Value	Default value	Description
Timer1	R	AF	DINT	-	-	Displays the time since system startup in ms. After reaching the maximum ($2^{31}-1$ ms, that is, approximately, 24.8 days), it starts counting upwards from 0. The counter is automatically started after a reset. The resolution of <code>Timer1</code> is the <code>CycleTime</code> of the Sercos drive bus.
Timer10	R	AF	DINT	-	-	Displays the time since system startup in 10 ms steps.
CycleLoad	R	AF	DINT	-	-	Indicates the utilization in % of the real-time cycle (<code>CycleLoad</code> Parameter (<i>see page 84</i>)).
RTBReadRes	R	AD	DINT	-	-	Displays the real-time read reserve in μ s.
RTBWriteRes	R	AD	DINT	-	-	Displays the real-time write reserve in μ s.
PowerOK	R	AF	BOOL Enum	FALSE / 0 TRUE / 1	FALSE / 0	Indicates power supply undervoltage: <ul style="list-style-type: none"> ● 0 = Supply voltage < 18 V. ● 1 = Supply voltage > 18 V.
SetRealTime-Clock	R/W(*)	EF	DT	-	-	The controller RTC is set when the <code>SetRealTimeClock</code> parameter is written. The parameter only displays the time the clock was set or the time of the last boot when the hardware clock has been installed and is functioning properly.
RealTimeClock	R	AF	DT	-	-	Displays date and time of the software clock, it is set automatically after controller boot from the real-time hardware clock (running with battery when the controller is turned off).

Parameter	Access	Param . type	Data type	Value	Default value	Description
BatteryLow-WarningDelay	R/W	EF	REAL	0...50	0.0	Delay for low battery message in hours. If the capacity of the battery goes below a minimum value, the diagnostic message 8037 Battery low is triggered. If the diagnostic message is acknowledged without replacing the battery, the diagnostic message will be triggered after the defined delay in this parameter. The delay of the diagnostic message is not continued when the system is reset.
Versions						
FW_Version	R	AK	STRING	-	-	Displays the firmware version of the controller and the creation date.
Controller-Type	R	AK	STRING	-	-	Displays various hardware information: <ul style="list-style-type: none"> ● Controller type: Name of the controller type. ● AX: Maximum number of axes according to cycle time. ● RAM: Main memory expansion in Mbytes. ● NVRAM: NVRAM expansion in Kbytes. ● Disks: SD card size in Mbytes.
Controller-Type1	R	AK	STRING	-	-	Displays various hardware information: <ul style="list-style-type: none"> ● PFPGA version. ● CPU version. ● Bios version. ● SFPGA version.
HW_Code	R	AK	STRING	-	-	Displays the hardware code of the controller.
SerialNumber	R	AK	STRING	-	-	Displays the serial number of the controller.

Parameter	Access	Param . type	Data type	Value	Default value	Description
Memory & Disks						
RamDiskSize	R/W	ER	DINT	128...4096	1024	Defines the size of the RamDisk in Kbytes. When switching on the controller, a RamDisk with the identifier ram0: is generated in the main memory. The system uses the RamDisk as a temporary memory for data when reading off the message logger.
RamDiskFree	R	AF	DINT	-	-	Displays the free memory space of the RamDisk.
Diskfree	R	AF	DINT	-	-	Displays the free memory space of the SD card.
Memoryfree	R	AF	DINT	-	-	Displays the free memory space of the RAM system memory.
System						
Systemticks	R/W	ER	DINT	10...20000	4000	This can be used to set the system cycle of the controller. NOTE: Consult the Schneider Electric application department when attempting to change this parameter.
EnableLoadEff	R/W(*)	EF	BOOL Enum	off / 0 on / 1	off / 0	This parameter starts and stops the effective CPU load measurement: <ul style="list-style-type: none"> ● 0 = Effective CPU load measurement stopped. ● 1 = Effective CPU load measurement started.
AvailableLoad	R	AD	DINT	0...100	0	Displays the available CPU time in %.
Available-LoadPeriod	R/W	EF	DINT	1...2000	100	Defines the measurement period for the effective CPU load measurement in ms. The remaining calculation time is calculated for this period.

Parameter	Access	Param . type	Data type	Value	Default value	Description
RemoteCommunicationAccess	R/W	ER	DINT Enum	read only / 0 read/write / 1 read/write/ save / 2	read/write/ save / 2	<p>You can change various communication parameters of the controller using the NetManage tool.</p> <p>If this is not desired, you can prevent it using the parameter RemoteCommunicationAccess:</p> <ul style="list-style-type: none"> ● 0 = The values are displayed in the NetManage tool, but the controller does not permit any changes to the communication parameters. ● 1 = The controller temporarily changes its communication parameters. ● 2 = The controller changes its communication parameters and immediately saves them on the SD card (thus keeping this setting even when being restarted). Saving is optional and can be selected in the NetManage tool.
ActivateFatalCrashReaction	R/W	ED	BOOL Enum	no / 0 yes / 1	yes / 1	Activates/deactivates the specific behavior when the controller stops responding.
IEC-Program						
IECRetainFree	R	AF	DINT	-	-	Displays the free memory in the retain area.
ProjectDate	R	AD	DT	-	-	Displays the project date.
ProjectName	R	AD	STRING	-	-	Displays the project name.
ProjectTitle	R	AD	STRING	-	-	Displays the project title.
ProjectVersion	R	AD	STRING	-	-	Displays the project version.
ProjectAuthor	R	AD	STRING	-	-	Displays the project author.
ProjectDescription	R	AD	STRING	-	-	Displays the project description.

Parameter	Access	Param . type	Data type	Value	Default value	Description
ProgrammingSystem	R	AD	STRING	-	-	Displays the programming system version (PSV) and the version of the device description (TSV) that was used to create the program.
OnlineChangeCounter	R	AD	DINT	-	0	The value indicates how many online change updates of the program sequences have been made since the last program download. Using this value, certain initialization steps can be called in the program after an online change.
IECProgramStateSet	R/W	EF	DINT Enum	stop / 0 start / 1	stop / 0	Starts / stops the IEC program and presents the current state of the IEC program.
-						
ObjectType	R	AD	STRING	LMCxx8	LMCxx8	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	Logical address of the controller parameters. stLogicalAddress = STRUCT (udiType, udiInstance, udiParameterId)
udiType	R	-	UDINT	-	-	
udiInstance	R	-	UDINT	-	-	
udiParameterId	R	-	UDINT	-	-	

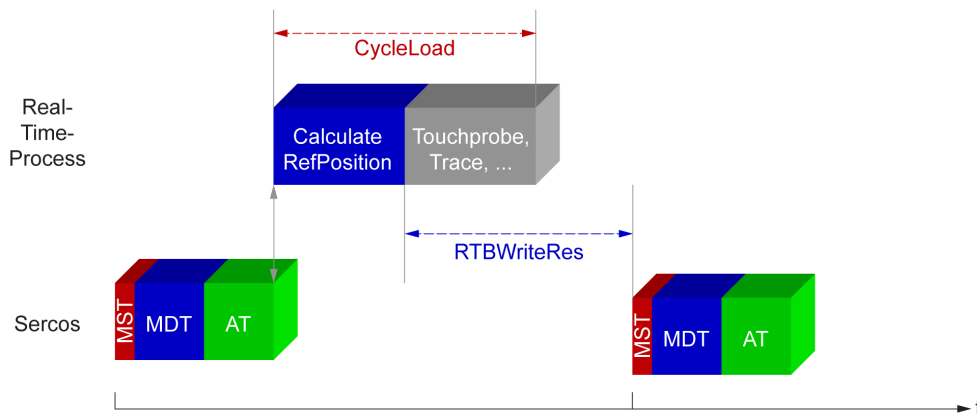
(*) For more information on the parameter access rights, refer to Parameter Types ([see page 27](#)).

CycleLoad Parameter

The parameter `CycleLoad` indicates the utilized capacity (in %) of the controller by the real-time process. The value of the parameter should not exceed a mean of 40 to 50% (brief peaks are tolerated). The remaining 50-60% are then available for the other system functions such as the fieldbus server, network, and the program. Diagnostic message 8511 CPU time overflow is sent when 100% is reached.

The `CycleLoad` parameter is a simple and clear variable for evaluating the system load.

This illustration presents the relationship between the parameters `CycleLoad` and `RTBWriteRes`:



The real-time process (RTP) is the most important system task. It is responsible for executing all real-time tasks at the correct time. Real-time processing is triggered by the Sercos real-time bus during each bus cycle.

`CycleLoad` is performed in 2 main steps:

Step	Description
1	<ul style="list-style-type: none"> ● Preparation of the cycle. ● Initialization of measuring variables and monitoring. ● Acceptance of the real-time data provided by the most recent drive telegrams (ATs) of the Sercos slaves. ● Processing of all master encoders such as virtual master encoders and physical master encoders. ● Processing of all slave encoders. ● All axes: <ul style="list-style-type: none"> ○ Diagnostic status. ○ Status machine of the drives. ○ Real-time job preparation. ○ POS and CAM generators (master and slave curves) ... ○ New reference values are now available. ○ The data is transmitted in the master data telegram (MDT) in the next cycle. ● The reference values are transferred for transmission in the next cycle.
2	<p>The remaining real-time functions are executed, such as:</p> <ul style="list-style-type: none"> ● Touchprobe. ● Trace, and so on...

NOTE: Certain externally event-controlled tasks and also high-priority tasks can adversely affect the execution of the real-time process. Monitoring the variables `CycleLoad`, `RTBReadRes` and `RTBWriteRes` allows the evaluation of the dynamic behavior.

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 process the communication settings of the controller.

Controller Selection Toolbar

The toolbar contains the following buttons:



Label	Button	Description
1	Optical	Click this button to cause the selected controller to indicate an optical signal: It flashes a control LED quickly. This can help you to identify the respective controller if many controllers are used. The function stops on a second click or after about 30 seconds.
2	Optical and acoustical	Not supported.
3	Update	Click this button to refresh the list of controllers. A request is sent to the controllers in the network. Any controller that responds to the request is listed with the current values. Pre-existing entries of controllers are updated with every new request. Controllers that are already in the list but that do not respond to a new request are not deleted. They are marked as inactive by a red cross being added to the controller icon. The Update button corresponds to the Refresh list command that is provided in the context menu if you right-click a controller in the list. To refresh the information of a selected controller, the context menu provides the command Refresh this controller . This command requests more detailed information from the selected controller. NOTE: The Refresh this controller command can also refresh the information of other controllers.

Label	Button	Description
4	Remove inactive controllers from list	<p>Controllers that do not respond to a network scan are marked as inactive in the list. This is indicated by a red cross being added to the controller icon. Click this button to remove all controllers marked as inactive controllers simultaneously from the list.</p> <p>NOTE: Because of network issues, a controller can be marked as inactive even if not.</p> <p>The context menu that opens if you right-click a controller in the list provides 2 other commands for removing controllers:</p> <ul style="list-style-type: none"> • The Remove selected controller from list command allows you to remove only the selected controller from the list. • The Remove all controllers from list command allows you to remove all controllers simultaneously from the list.
5	New favorite	<p>You can use Favorites to adjust the selection of controllers to your personal requirements. This can help you to keep track of many controllers in the network.</p> <p>A Favorite describes a collection of controllers that are recognized by a unique identifier.</p> <p>Click a favorite button (such as Favorite 0) to select or deselect it. If you have not selected a favorite, all detected controllers are visible.</p> <p>You can also access Favorites via the context menu. It opens upon right-clicking a controller in the list.</p> <p>Move the cursor over a favorite button in the toolbar to view the associated controllers as a tooltip.</p>
6	Favorite x	

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

Process Communication Settings

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

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

- Without the **Save settings permanently** option:
Configure the communication parameters and click **OK**. These settings are immediately taken into account and are not kept if the controller is reset.
- With the **Save settings permanently** option:
You can also activate the **Save settings permanently** option before you click **OK**. Once this option is activated, the configured Ethernet parameters are stored on the SD card. After reset of the controller, the configured Ethernet parameters from the SD card will be active.

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, 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: <ul style="list-style-type: none"> ● Keep current values ● Set all outputs to default ● Execute program
	Update all variables in all devices If this option is activated, then for all devices of the current controller configuration all I/O variables will get updated in each cycle of the bus cycle task. This corresponds to the option Always update variables , which can be set separately for each device in the I/O Mapping dialog.
Bus cycle options	<p>Bus cycle task</p> <p>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.</p> <p>NOTE: <unspecified> means that the task is in "slowest cyclic task" mode.</p>

Element		Description
Additional settings	Generate force variables for IO mapping	Not used.
	Enable Diagnosis for devices	Not used.

Chapter 9

Embedded Inputs and Outputs Configuration

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Embedded I/O Configuration	92
Master Encoder Input Configuration	100

Embedded I/O Configuration

Introduction

The Modicon LMC078 Motion Controller provides:

- 12 embedded inputs:
 - 8 digital inputs: **DI_0...DI_7**
 - 4 advanced digital inputs (touchprobe and interrupt): **ADI_0...ADI_3**
- 8 embedded outputs:
 - 7 digital outputs: **DQ_0...DQ_6**
 - 1 digital output configurable as watchdog output: **DQ_WD**

Digital Input Group Configuration

To configure the digital input group, double-click the **DIG_DigitalIn** node in the **Devices tree**.

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)				Object name (EF)[0x0013]
Bit0_7	USINT				Bit 0 to 7 (AF)[0x0001]
Bit8_11	USINT				Bit 8 to 11 (AF)[0x0014]
ObjectType	STRING	'D_ING2'	'D_ING2'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]
udiType	UDINT	0	0		
udiInstance	UDINT	0	0		
udiParameterId	UDINT	0	0		

This table describes the different parameters:

Parameter	Access	Param. type	Data type	Description
Name	R/W ^(*)	EF	STRING(40)	Symbolic name of the configuration object.
Bit0_7	R	AF	USINT	Value of the digital inputs DI_0...DI_7 , each bit is assigned to an input. Bit x = value of input DI_x
Bit8_11	R	AF	USINT	Value of the advanced digital inputs ADI_0...ADI_3 , each bit is assigned to an advanced input. Bit x = value of input ADI_x
ObjectType	R	AD	STRING	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	Logical address of the input group.

^(*) For more information on the parameter access rights, refer to Parameter Types (*see page 27*).

Digital Input Configuration

To configure a digital input, double-click the **DI_x** node in the **Devices tree**.

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0018]
Value	Enumeration of BOOL				Actual value (AD)[0x0001]
FilterTime	UDINT(90..4294967)	100	100	µs	inputfiltertime (in 1µs steps) (EF)[0x001A]
ObjectType	STRING	'D_IN5'	'D_IN5'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]
udiType	UDINT	0	0		
udiInstance	UDINT	0	0		
udiParameterId	UDINT	0	0		

This table describes the input parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Value	R	AD	BOOL Enum	L / 0 H / 1	-	Value of the digital input.
FilterTime	R/W	EF	UDINT	90...4294967	100	Filter time of the input in µs.
ObjectType	R	AD	STRING	D_IN5	D_IN5	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	Logical address of the inputs.

(*) For more information on the parameter access rights, refer to Parameter Types ([see page 27](#)).

Advanced Digital Input Configuration

To configure an advanced digital input, double-click the **ADI_x** node in the **Devices tree**.

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0018]
Value	Enumeration of BOOL				Actual value (AD)[0x0001]
FilterTime	UDINT(90..4294967)	100	100	µs	inputfiltertime (in 1 µs steps) (EF)[0x001A]
CaptureState	Enumeration of DINT	inactive / 0	inactive / 0		State of capture function (AD)[0x0005]
SensorDelay	LREAL(-100..100)	0.0	0.0	msec	Sensor delay (EF)[0x0004]
Counter	UDINT	0	0		counter for input (EF)[0x000C]
Enable	Enumeration of BOOL	off / 0	off / 0		Enable interrupt for input (EF)[0x000D]
ExtEventEdge	Enumeration of DINT	positive / 1	positive / 1		active edge for input (EF)[0x001B]
ObjectType	STRING	'D_IN62'	'D_IN62'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]
udiType	UDINT	0	0		
udiInstance	UDINT	0	0		
udiParameterId	UDINT	0	0		

This table describes the advanced input parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Value	R	AD	BOOL Enum	L / 0 H / 1	-	Value of the advanced digital input.
FilterTime	R/W	EF	UDINT	90...4294967	100	Filter time of the input in µs.
CaptureState	R	AD	DINT Enum	inactive / 0 active / 1 captured / 2 overflow / 3 disabled / 4 not ready / 5 virtual / 6	inactive / 0	State of capture function.
SensorDelay	R/W	EF	LREAL	-100...100	0	Sensor delay in ms.
Counter	R	EF	UDINT	-	0	Counter of input.
Enable	R/W(*)	EF	BOOL Enum	off / 0 on / 1	off / 0	Enables the interrupt function of the input.
ExtEventEdge	R/W	EF	DINT Enum	negative / 0 positive / 1 negative and positive / 2	positive / 1	Defines the active edge of the input.

Parameter	Access	Param. type	Data type	Value	Default value	Description
ObjectType	R	AD	STRING	D_IN62	D_IN62	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	Logical address of the advanced inputs.

(*) For more information on the parameter access rights, refer to Parameter Types (*see page 27*).

Digital Output Group Configuration

To configure the digital output group, double-click the **DQG_DigitalOut** node in the **Devices tree**.

The screenshot shows a configuration window titled "DQG_DigitalOut" with a "Configuration" tab. It contains a table with the following data:

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0008]
Bit0_7	USINT(0..255)	2#00000000	2#00000000		Bit 0 to 7 (EF)[0x0001]
DiagMask	UINT	2#11111111	2#11111111		DiagnosisMask (EF)[0x0004]
OpenloadDiagMsk	UINT	2#11111111	2#11111111		DiagnosisMask (EF)[0x0005]
OverloadDiagMsk	UINT	2#11111111	2#11111111		DiagnosisMask (EF)[0x0006]
ObjectType	STRING	'D_OUTG5'	'D_OUTG5'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]
udiType	UDINT	0	0		
udiInstance	UDINT	0	0		
udiParameterId	UDINT	0	0		

This table describes the different parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Bit0_7	R/W(*)	EF	USINT	0	0	Value of the digital outputs DQ_0...DQ_7 , each bit is assigned to an output. Bit x = value of output DQ_x
DiagMask	R/W	EF	UINT	2#11111111	2#11111111	Enables the diagnostic message 8788 Wiring error for each output, each bit is assigned to an output. Bit x = 0, the output x is not monitored and the diagnostic message is not displayed.

Parameter	Access	Param. type	Data type	Value	Default value	Description
OpenloadDiagMask	R/W	EF	UINT	2#11111111	2#11111111	Enables the diagnostic message 8788 Wiring error / Openload for each output, each bit is assigned to an output. Bit x = 0 , the output x is not monitored and the diagnostic message is not displayed.
OverloadDiagMask	R/W	EF	UINT	2#11111111	2#11111111	Enables the diagnostic message 8788 Wiring error / Overload for each output, each bit is assigned to an output. Bit x = 0 , the output x is not monitored and the diagnostic message is not displayed.
ObjectType	R	AD	STRING	D_OUTG5	D_OUTG5	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	Logical address of the output group.

(*) For more information on the parameter access rights, refer to Parameter Types (*see page 27*).

Digital Output Configuration

To configure a digital output, double-click the **DQ_x** node in the **Devices tree**.

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0007]
Value	Enumeration of BOOL	L / 0	L / 0		Actual value (ED)[0x0001]
Status	Enumeration of DINT	default / 0	default / 0		Status (AF)[0x0002]
EnableDiagMsg	Enumeration of BOOL	H / 1	H / 1		Enable Diagnosis text (EF)[0x0003]
OpenloadDiagMsg	Enumeration of BOOL	H / 1	H / 1		Enable openload diagnosis text (EF)[0x0004]
OverloadDiagMsg	Enumeration of BOOL	H / 1	H / 1		Enable overload diagnosis text (EF)[0x0005]
ObjectType	STRING	'D_OUT1'	'D_OUT1'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]
udiType	UDINT	0	0		
udiInstance	UDINT	0	0		
udiParameterId	UDINT	0	0		

This table describes the output parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Value	R/W(*)	ED	BOOL Enum	L / 0 H / 1	L / 0	Displays the value of the digital output.
Status	R	AF	DINT Enum	default / 0 openload / 1 overload / 2	default / 0	Displays the status of the digital output: <ul style="list-style-type: none"> ● 0: default state ● 1: no load available ● 2: there is a short circuit
EnableDiagMsg	R/W(*)	EF	BOOL Enum	L / 0 H / 1	H / 1	0 = The output does not report the diagnostic message 8788 Wiring error. 1 = The output reports the diagnostic message 8788 Wiring error.
OpenloadDiagMsg	R/W(*)	EF	BOOL Enum	L / 0 H / 1	H / 1	0 = The output does not report the diagnostic message 8788 Wiring error / Openload. 1 = The output reports the diagnostic message 8788 Wiring error / Openload.
OverloadDiagMsg	R/W(*)	EF	BOOL Enum	L / 0 H / 1	H / 1	0 = The output does not report the diagnostic message 8788 Wiring error / Overload. 1 = The output reports the diagnostic message 8788 Wiring error / Overload.
ObjectType	R	AD	STRING	D_OUT1	D_OUT1	Object type.
stLogicalAddress	R	AD	ST_Logical Address	-	-	Logical address of the outputs.

(*) For more information on the parameter access rights, refer to Parameter Types ([see page 27](#)).

Watchdog Output Configuration

To configure the watchdog output, double-click the **DQ_WD** node in the **Devices tree**.

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0007]
Value	Enumeration of BOOL	L / 0	L / 0		Actual value (ED)[0x0001]
Status	Enumeration of DINT	default / 0	default / 0		Status (AF)[0x0002]
EnableDiagMsg	Enumeration of BOOL	H / 1	H / 1		Enable Diagnosis text (EF)[0x0003]
OpenloadDiagMsg	Enumeration of BOOL	H / 1	H / 1		Enable openload diagnosis text (EF)[0x0004]
OverloadDiagMsg	Enumeration of BOOL	H / 1	H / 1		Enable overload diagnosis text (EF)[0x0005]
WDOutEnable	Enumeration of BOOL	Off / 0	Off / 0		output is watchdog controlled (ED)[0x0008]
ObjectType	STRING	'D_OUT2'	'D_OUT2'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]
udiType	UDINT	0	0		
udiInstance	UDINT	0	0		
udiParameterId	UDINT	0	0		

This table describes the watchdog output parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Value	R/W(*)	ED	BOOL Enum	L / 0 H / 1	L / 0	Displays the value of the watchdog output.
Status	R	AF	DINT Enum	default / 0 openload / 1 overload / 2	default / 0	Displays the status of the watchdog output: <ul style="list-style-type: none"> ● 0: default state ● 1: no load available ● 2: there is a short circuit
EnableDiagMsg	R/W(*)	EF	BOOL Enum	L / 0 H / 1	H / 1	0 = The output does not report the diagnostic message 8788 Wiring error. 1 = The output reports the diagnostic message 8788 Wiring error.
OpenloadDiagMsg	R/W(*)	EF	BOOL Enum	L / 0 H / 1	H / 1	0 = The output does not report the diagnostic message 8788 Wiring error / Openload. 1 = The output reports the diagnostic message 8788 Wiring error / Openload.

Parameter	Access	Param. type	Data type	Value	Default value	Description
OverloadDiagMsg	R/W ^(*)	EF	BOOL Enum	L / 0 H / 1	H / 1	0 = The output does not report the diagnostic message 8788 Wiring error / Overload. 1 = The output reports the diagnostic message 8788 Wiring error / Overload.
WDOutEnable	R/W	ED	BOOL Enum	off / 0 on / 1	off / 0	0 = The watchdog output is deactivated. 1 = The watchdog output is activated and controlled by the system watchdog (<i>see page 45</i>).
ObjectType	R	AD	STRING	D_OUT2	D_OUT2	Object type.
stLogicalAddress	R	AD	ST_Logical Address	-	-	Logical address of the watchdog output.

^(*) For more information on the parameter access rights, refer to Parameter Types (*see page 27*).

Master Encoder Input Configuration

Introduction

The controller has a specific hardware encoder interface that can support:

- Incremental encoder (RS422)
- Absolute encoder (SinCos Hiperface)

The goal of this function is to connect an encoder to acquire a position that is synchronous with the Sercos real-time bus. This acquired position can then be used as the master axis for motion drives on Sercos.

The master encoder configuration is split in two parts:

- The encoder node that supports the hardware configuration.
- The **SoftMotion_Encoder** node that supports the scaling configuration.

Add an Encoder

To add an encoder to your controller, select **Incremental Encoder Input** or **SinCos Encoder Input** in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on your controller node.

Result: The encoder is added to your controller as a new subnode and a **SoftMotion_Encoder** node is added as a subnode of your encoder.

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*)

Incremental Encoder Configuration

To configure the incremental encoder, double-click the encoder node in the **Devices tree**:

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0014]
Enable	Enumeration of BOOL	on / 1	on / 1		Enable (EF)[0x0001]
State	Enumeration of DINT	not ready / 0	not ready / 0		incremental input state (AD)[0x000E]
Filter	DINT(0..1024)	0	0	ms	Filter (EF)[0x0004]
CheckOff	Enumeration of BOOL	no / 0	no / 0		Encoder Check Off (EF)[0x0005]
ZeroTrackCheckOff	Enumeration of BOOL	no / 0	no / 0		Zero Track Check Off (EF)[0x0013]
ZeroTrackStart	Enumeration of BOOL	false / 0	false / 0		Zero Track detection start (EF)[0x0009]
ZeroTrackDetected	Enumeration of BOOL	false / 0	false / 0		Zero Track detected (AD)[0x000A]
DiagClass	DINT				Diagnosis class (AD)[0x0077]
DiagCode	DINT				Diagnosis code (AD)[0x000F]
DiagSource					Diagnosis source (AD)[0x0078]
DiagMsg	STRING(39)	"	"		Diagnosis text (AD)[0x0011]
DiagExtMsg	STRING(14)	"	"		Extended diagnosis message (AD)[0x0079]
ObjectType	STRING	'INC_IN2'	'INC_IN2'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]

This table describes the incremental encoder parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Enable	R/W	EF	BOOL Enum	off / 0 on / 1	on / 1	Enables the encoder: <ul style="list-style-type: none"> ● 0 = The encoder is not processed by the real-time process. ● 1 = The encoder is processed by the real-time process.
State	R	AD	DINT Enum	not ready / 0 initialization / 1 no sync / 2 ready / 3	not ready / 0	Displays the availability and the validity of the encoder position data: <ul style="list-style-type: none"> ● 0 = The encoder or encoder processing is not ready. ● 1 = Encoder processing is initialized. ● 2 = Encoder processing does not run synchronously with the RTP. ● 3 = The velocity and position values of the encoder are valid.
Filter	R/W	EF	DINT	0...1024	0	Filtering value in ms. The filter is a low-pass filter and affects the actual velocity. If the filter is set too high, it can lead to oscillation of the system depending on the velocity path.
CheckOff	R/W	EF	BOOL Enum	no / 0 yes / 1	no / 0	Disables the encoder monitoring: <ul style="list-style-type: none"> ● 0 = Encoder verification active. ● 1 = Encoder monitoring not active (track and cable monitoring).

Parameter	Access	Param. type	Data type	Value	Default value	Description
ZeroTrack-CheckOff	R/W	EF	BOOL Enum	no / 0 yes / 1	no / 0	Disables the zero track monitoring of the encoder: <ul style="list-style-type: none"> ● 0 = Zero track monitoring active. ● 1 = Zero track monitoring not active.
ZeroTrack-Start	R/W(*)	EF	BOOL Enum	FALSE / 0 TRUE / 1	FALSE / 0	A rising edge of this parameter enables the zero track detection: <ul style="list-style-type: none"> ● 0 = Zero track detection disabled. ● 1 = Zero track detection enabled.
ZeroTrack-Detected	R	AD	BOOL Enum	FALSE / 0 TRUE / 1	FALSE / 0	When zero track detection is enabled, this parameter indicates the zero track detection: <ul style="list-style-type: none"> ● 0 = Zero track not detected. ● 1 = Zero track detected. When a zero track is detected, the position of the encoder is set to 0.
DiagClass	R	AD	DINT	-	-	Displays the diagnostic class (<i>see page 262</i>) in decimal code.
DiagCode	R	AD	DINT	-	-	Displays the diagnostic code (<i>see page 262</i>) in decimal code.
DiagSource	R	AD	ST_LogicalAddress	-	-	Logical address of the diagnostic source.
DiagMsg	R	AD	STRING	-	-	Diagnostic message.
DiagExtMsg	R	AD	STRING	-	-	Extended diagnostic message.
ObjectType	R	AD	STRING	INC_IN2	INC_IN2	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	Logical address of the incremental encoder.

(*) For more information on the parameter access rights, refer to Parameter Types (*see page 27*).

Hiperface (SinCos) Encoder Configuration

To configure the Hiperface encoder, double-click the encoder node in the **Devices tree**:

Parameter	Type	Value	Default Value	Unit	Description
Name	STRING(40)	"	"		Object name (EF)[0x0028]
Enable	Enumeration of BOOL	on / 1	on / 1		Enable (EF)[0x0001]
Filter	DINT(0..1024)	0	0	ms	Filter (EF)[0x0003]
CheckOff	Enumeration of BOOL	no / 0	no / 0		Encoder Check Off (EF)[0x0004]
EncoderType	DINT				Encoder type 22-SRS 27-SRM 02-SCS 07-SCM (AK)[0x0008]
State	Enumeration of DINT	not ready / 0	not ready / 0		Phys. master encoder state (AD)[0x0024]
SignalQuality	UDINT			%	Quality of the analog position signals (AF)[0x0020]
SignalQualityLimit	UDINT(0..100)	50	50	%	Errorlimit of the analog position signals (EF)[0x0021]
DiagClass	DINT				Diagnosis class (AD)[0x0077]
DiagCode	DINT				Diagnosis code (AD)[0x0025]
DiagSource					Diagnosis source (AD)[0x0079]
DiagMsg	STRING(39)	"	"		Diagnosis text (AD)[0x0027]
DiagExtMsg	STRING(14)	"	"		Extended diagnosis message (AD)[0x0078]
ObjectType	STRING	'P_ENC2'	'P_ENC2'		Object type (AD)[0x10000001]
stLogicalAddress					extended logic address (AD)[0x10000003]

This table describes the Hiperface encoder parameters:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Name	R/W(*)	EF	STRING(40)	"	"	Symbolic name of the configuration object.
Enable	R/W	EF	BOOL Enum	off / 0 on / 1	on / 1	Enables the encoder: <ul style="list-style-type: none"> 0 = The encoder is not processed by the real-time process. 1 = The encoder is processed by the real-time process.
Filter	R/W	EF	DINT	0...1024	0	Filtering value in ms. The filter is a low-pass filter and affects the actual velocity. If the filter is set too high, it can lead to oscillation of the system depending on the velocity path.

Parameter	Access	Param. type	Data type	Value	Default value	Description
CheckOff	R/W	EF	BOOL Enum	no / 0 yes / 1	no / 0	Disables the encoder monitoring: <ul style="list-style-type: none"> ● 0 = Encoder monitoring active. ● 1 = Encoder monitoring not active. The diagnostic message 8601 Master Encoder signal out of range is disabled.
Encoder-Type	R	AK	DINT	-	-	Displays the supported SinCos Hiperface encoder type (read at controller boot): <ul style="list-style-type: none"> ● 00h: value not yet read ● 02h: SCS 60/70 ● 07h: SCM 60/70 multi turn ● 22h: SRS 50/60/64 and SCK 25/35/40/45/53 single turn ● 27h: SRM 50/60/64 and motor 25/35/40/45/53 multi turn ● 32h: SKS 36 single turn ● 37h: SKM 36 multi turn ● 42h: SEK 52 single turn

Parameter	Access	Param. type	Data type	Value	Default value	Description
State	R	AD	DINT Enum	not ready / 0 initialization / 1 no sync / 2 get type / 3 ready / 10 read position / 11 write position / 12 read error code / 13	not ready / 0	Displays the availability and the validity of the encoder position data: <ul style="list-style-type: none"> ● 0: The encoder or encoder processing is not ready. ● 1: Encoder processing is initialized. ● 2: Encoder processing does not run synchronously with the RTP. ● 3: The encoder type is read out. ● 4..9: Invalid state. ● 10: The velocity and position values of the encoder are valid. ● 11: The absolute position is read from the encoder. ● 12: The absolute position is written into the encoder. ● 13: The error code is read from the encoder.
Signal-Quality	R	AF	UDINT	-	-	Describes the signal quality of the analog sine and cosine tracks of the encoder (in %). The sine and cosine signals must correspond to the following formula: Sine signal ² + cosine signal ² = 1 (100%) This parameter represents this formula normalized to 100%.
Signal-QualityLimit	R	EF	UDINT	50	50	This parameter determines at which value of SignalQuality the diagnostic code 8601 Master encoder signal out of the range is issued.

Parameter	Access	Param. type	Data type	Value	Default value	Description
DiagClass	R	AD	DINT	-	-	Displays the diagnostic class (<i>see page 262</i>) in decimal code.
DiagCode	R	AD	DINT	-	-	Displays the diagnostic code (<i>see page 262</i>) in decimal code.
DiagSource	R	AD	ST_LogicalAddress	-	-	Logical address of the diagnostic source.
DiagMsg	R	AD	STRING	-	-	Diagnostic message.
DiagExtMsg	R	AD	STRING	-	-	Extended diagnostic message.
ObjectType	R	AD	STRING	P_ENC2	P_ENC2	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	Logical address of the Hiperface encoder.

SoftMotion Encoder Configuration

To configure the SoftMotion encoder, double-click the **SoftMotion_Encoder** node in the **Devices tree**.

The configuration of SoftMotion encoder is described in the SoMachine online help, chapter *Programming with SoMachine / SoftMotion / SoftMotion Device Editor*.

Chapter 10

Communication Modules

Introduction

This chapter describes how to add and configure a communication module.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
10.1	PROFIBUS DP Slave Module Configuration	108
10.2	EtherNet/IP Adapter Configuration	119
10.3	Ethernet/IP Scanner Configuration	130

Section 10.1

PROFIBUS DP Slave Module Configuration

Introduction

This section describes the configuration of the VW3E704000000 PROFIBUS DP module.

What Is in This Section?

This section contains the following topics:

Topic	Page
Add a PROFIBUS DP Slave Module	109
PROFIBUS DP Slave Module Configuration	111
Acyclic Data Exchange	116

Add a PROFIBUS DP Slave Module

Overview

With the PROFIBUS protocol, the data is exchanged according to the master/slave principle. Only the master can initialize communication. The slaves respond to requests from masters. Several masters can coexist on the same bus. In this case, the slave I/O can be read by all the masters. However, a single master has write access to the outputs. The number of data items exchanged is defined during the configuration.

There are 2 types of exchange services supported by this module:

- I/O cyclic frame exchanges
- Acyclic data exchanges with PROFIBUS DPV1 function

Add a PROFIBUS DP Slave Module

Select the **PROFIBUS-DPV1-Slave** module in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on your controller node.

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

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

Add Virtual I/O Devices

Below a PROFIBUS DP slave module you can add one or several virtual I/O devices.

The PROFIBUS DP slave module is an intermediate between the PROFIBUS master and the controller, and data is exchanged by using virtual I/O devices that you define when configuring the module. The virtual devices are not physical I/O modules, but are logical input and output objects within the module that you can then map to memory within the controller. These input and output objects are read from and written to by the PROFIBUS master. In turn, the module reads and writes this data to I/O memory locations in the controller so that you can use the data within your application program.

Select the virtual I/O devices in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the **PROFIBUS-DPV1-Slave** node.

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

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

The virtual I/O devices you define within the module can be either input or output, and can vary in size as defined by the table:

Name	Number of I/O	Format
X byte input (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	Byte
X byte input con (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	
X byte output (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	
X byte output con (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	
X word input (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	Word
X word input con (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	
X word output (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	
X word output con (0x**)	X= 1,2,3,4,8,12,16,20,32 or 64	

Once you have defined these virtual input and/or output devices within the module, you can then map these devices to memory locations within the controller. The type of memory objects you map these virtual I/O devices to depends on the type of exchange you define between the master and the slave.

PROFIBUS DP Slave Module Configuration

PROFIBUS DP Slave Module Configuration

In the **Devices tree**, double-click **PBS_Slave (PROFIBUS-DPV1-Slave)**:

Parameter	Type	Value	Default Value	Unit
InitParameters				
BusAddr	BYTE(2..126)	2	2	
BaudRate	Enumeration of BYTE	Baudrate_AUTO / 15	Baudrate_AUTO / 15	
Connector	Enumeration of INT	Front / 1	Front / 1	
not in use	Enumeration of BYTE	NOT_IN_USE	NOT_IN_USE	
WdgEnabled	BOOL	TRUE	TRUE	
WdgTime	UDINT(20..65535)	100	100	
DPV1Enable	BOOL	TRUE	TRUE	
SyncSupported	BOOL	TRUE	TRUE	
FreezeSupported	BOOL	TRUE	TRUE	
FailsafeSupported	BOOL	TRUE	TRUE	
NoAddrChangeSupported	BOOL	TRUE	TRUE	
Internal				
Info				
Diag				

Parameters in black Read/write access

Parameters in gray Read-only access

The following parameters are provided in the **PROFIBUS Configuration** tab:

Parameter	Type	Value	Default value	Description
InitParameters				
BusAddr	BYTE	2...126	2	PROFIBUS DP slave address.
BaudRate	BYTE Enum	0...15	15	PROFIBUS transmission rate: <ul style="list-style-type: none"> ● 0 = 9600 Baud ● 1 = 19.2 KBAud ● 2 = 93.75 KBAud ● 3 = 187.5 KBAud ● 4 = 500 KBAud ● 6 = 1500 KBAud ● 7 = 3 MBAud ● 8 = 6 MBAud ● 9 = 12 MBAud ● 10 = 31.25 KBAud ● 11 = 45.45 KBAud ● 15 = Auto, the transmission rate set on the master is recognized by the slave.

Parameter	Type	Value	Default value	Description
WdgEnabled	BOOL	TRUE FALSE	TRUE	Enables the watchdog: <ul style="list-style-type: none"> ● TRUE = The watchdog is activated. The value of the parameter <code>WdgTime</code> is used as watchdog timeout. ● FALSE = The watchdog is deactivated. The value of the parameter <code>WdgTime</code> is ignored.
WdgTime	UDINT	20...65535	100	Determines after which time a slave is detected as incommunicative by the watchdog. The time is given in ms.
DPV1Enable	BOOL	TRUE FALSE	TRUE	Enables the DPV1 functions for acyclic communication: <ul style="list-style-type: none"> ● TRUE = The DPV1 mode is activated. ● FALSE = The DPV1 mode is deactivated.
SyncSupported	BOOL	TRUE FALSE	TRUE	Enables the sync mode that supports the sync command: <ul style="list-style-type: none"> ● TRUE = The sync mode is activated. ● FALSE = The sync mode is deactivated.
FreezeSupported	BOOL	TRUE FALSE	TRUE	Enables the freeze mode that supports the freeze command: <ul style="list-style-type: none"> ● TRUE = The freeze mode is activated. ● FALSE = The freeze mode is deactivated.
FailsafeSupported	BOOL	TRUE FALSE	TRUE	Enables a proprietary "failsafe" mode if supported by the device: <ul style="list-style-type: none"> ● TRUE = The mode is activated. ● FALSE = The mode is deactivated.
Internal				
Reserved				
Info				
IdentNumber	WORD	3424	3424	Displays the identification number of the PROFIBUS DP slave module.
VendorName	STRING	'Schneider Electric'	'Schneider Electric'	Displays the vendor name of the PROFIBUS DP slave module.
ModelName	STRING	'PROFIBUS DPV1 slave'	'PROFIBUS DPV1 slave'	Displays the model name of the PROFIBUS DP slave module.
DriverInstance	DWORD	0	0	Displays the identifier for this driver instance.

Parameter	Type	Value	Default value	Description
Diag / ChannelCommonStatusBlock				
CommState	UDINT Enum	UNKNOWN / 0 NOT_ CONFIGURED / 1 STOP / 2 IDLE / 3 OPERATE / 4	UNKNOWN / 0	Displays the current network status of the communication channel: <ul style="list-style-type: none"> ● 0 = Indeterminable. ● 1 = Not configured. ● 2 = Stopped. ● 3 = Idle. ● 4 = Operational.
CommError	UDINT Enum	Diagnostic Codes <i>(see page 114)</i>	SUCCESS / 0x0	Displays the current diagnostic code of the communication channel.
ErrorCount	UDINT	-	0	Displays the total number of errors detected since the last power-up or the last restart.
Diag / ChannelExtendedStatusBlock				
Baudrate	UDINT Enum	0...15	Baudrate_ AUTO / 15	Displays the applied baud rate. The parameter displays the baud rate used by the master if the value AUTO has been set in the initialization parameter of the slave. PROFIBUS transmission rate: <ul style="list-style-type: none"> ● 0 = 9600 Baud ● 1 = 19.2 KBaud ● 2 = 93.75 KBaud ● 3 = 187.5 KBaud ● 4 = 500 KBaud ● 6 = 1500 KBaud ● 7 = 3 MBaud ● 8 = 6 MBaud ● 9 = 12 MBaud ● 10 = 31.25 KBaud ● 11 = 45.45 KBaud ● 15 = Auto, the master has no connection to the slave.

Diagnostic Codes

No error has been detected:

Value	Meaning
SUCCESS / 0x0	No error detected.

Runtime error has been detected:

Value	Meaning
WATCHDOG_TIMEOUT / 0xC000000C	The watchdog time has been exceeded.

Initialization errors have been detected:

Value	Meaning
INIT_FAULT / 0xC0000100	The initialization was not successful.
DATABASE_ACCESS_FAILED / 0xC0000101	Access to data memory was not successful.

Configuration errors have been detected:

Value	Meaning
NOT_CONFIGURED / 0xC0000119	The module is not configured.
CONFIGURATION_FAULT / 0xC0000120	A configuration error has been detected.
INCONSISTENT_DATA_SET / 0xC0000121	Inconsistent set data have been detected.
DATA_SET_MISMATCH / 0xC0000122	A mismatch of set data has been detected.
INSUFFICIENT_LICENSE / 0xC0000123	An insufficient license has been detected.
PARAMETER_ERROR / 0xC0000124	A parameter error has been detected.
INVALID_NETWORK_ADDRESS / 0xC0000125	The network address is not correct.
SECURITY_MEMORY / 0xC0000126	The security memory is not available.

Network errors have been detected:

Value	Meaning
COMM_NETWORK_FAULT / 0xC0000126	A network communication error has been detected.
COMM_CONNECTION_CLOSED / 0xC0000141	The communication connection has been closed.
COMM_CONNECTION_TIMEOUT / 0xC0000142	A communication connection timeout has been detected.
COMM_DUPLICATE_NODE / 0xC0000144	A duplicate node has been detected.
COMM_CABLE_DISCONNECT / 0xC0000145	A disconnected cable has been detected.
PROFIBUS_CONNECTION_TIMEOUT / 0xC009002E	A PROFIBUS connection timeout has been detected.

Acyclic Data Exchange

Registering for Callback

It is possible to register for a callback in the case of a noncyclical inquiry. If a noncyclical query of the master is received by the PROFIBUS DPV1 slave driver, the query is first made to registered data areas.

Then, all registered user function block (FB) instances are called up which have been registered with the PROFIBUS DPV1 slave by using the interfaces `IF_AsyncRead` and `IF_AsyncWrite`. At the same time, the state of the automatic inquiry processing is transferred as well, so that possible problems can be responded to. The current status is then written into parameter `iq_stError`.

To call up the `AsyncRead` method in the case of an incoming read query, an instance of the same type as the function block that implements the `IF_AsyncRead` interface has to be registered with the PROFIBUS DPV1 slave.

If the `AsyncRead` method of this function block instance is no longer to be called up when a noncyclical read inquiry is received, the registration of the instance has to be removed.

NOTE: Use the method `IsRegisteredAsyncRead` to verify whether a certain function block instance is already registered as callback with the PROFIBUS DPV1 slave.

A list of all relevant methods can be found here after.

NOTE: The `IoDrvPROFIBUSDPV1Slave` library is added to the library manager when you add the module. This library contains the interfaces, methods, and function blocks for managing the PROFIBUS DP module.

Relevant Methods for Callback Registration

This table lists the relevant methods for the registration of the PROFIBUS DP interface:

Method	Description
<code>IsRegisteredAsyncAlarmAck</code>	Returns whether the transferred function block is already registered for the callback upon arrival of an alarm acknowledge.
<code>IsRegisteredAsyncRead</code>	Returns whether the transferred function block is already registered for the callback upon arrival of a noncyclical read inquiry.
<code>IsRegisteredAsyncWrite</code>	Returns whether the transferred function block is already registered for the callback upon arrival of a noncyclical read inquiry.
<code>RegisterAsyncAlarmAck</code>	Registers the transferred function block instance for the callback with the PROFIBUS slave upon arrival of an alarm acknowledge.
<code>RegisterAsyncRead</code>	Registers the transferred function block instance for the callback with the PROFIBUS slave upon arrival of a noncyclical read inquiry.
<code>RegisterAsyncWrite</code>	Registers the transferred function block instance for the callback with the PROFIBUS slave upon arrival of a noncyclical write inquiry.

Method	Description
UnregisterAsyncAlarmAck	Unregisters the transferred function block instance for the callback with the PROFIBUS slave upon arrival of an alarm acknowledge.
UnregisterAsyncRead	Unregisters the transferred function block instance for the callback with the PROFIBUS slave upon arrival of a noncyclical read inquiry.
UnregisterAsyncWrite	Unregisters the transferred function block instance for the callback with the PROFIBUS slave upon arrival of a noncyclical write inquiry.

Registering Noncyclical Data Areas

Data areas can be registered in the PROFIBUS DPV1 slave. Noncyclical queries (`AsyncRead`, `AsyncWrite`) to registered data areas are then automatically processed by the PROFIBUS DPV1 slave. A slot and an index are required in order to address a noncyclical data area.

In order to create a noncyclical module, the structure `ST_PROFIBUSDPV1AsyncDataModule` is used. A slot and an index have to be entered into this structure as well as a pointer to a data area and the length of the data at this address.

Ensure that the pointer `pbyData` points to a memory area that actually exists during runtime and is not deleted. For this purpose, for example, declare the array as a variable in the program. The data module structures do not have to be available permanently since the contents of the structure are copied when registering the data module.

A list of all relevant methods can be found here after.

Relevant Methods for Data Area Registration

This table lists the relevant methods for the registration of the data areas of the `IF_PROFIBUS_DPV1_Slave` interface:

Method	Description
<code>IsRegisteredDataModule</code>	Verifies whether the transferred data module is registered. The return values are: <ul style="list-style-type: none"> ● 0: No data module is registered on a certain slot and index. ● 1: A data module is registered on a certain slot and index. ● 2: The slot is invalid. ● 3: The index is invalid.
<code>RegisterAsyncDataModule</code>	Registers the transferred data module with a slot and an index. The return values are: <ul style="list-style-type: none"> ● 0: The data module is registered. ● 1: The slot is invalid. ● 2: The index is invalid. ● 3: The pointer to the data area is 0. ● 4: The length is invalid. ● 5: The data module already exists.
<code>UnregisterAllAsyncDataModules</code>	Removes registration of all registered data modules.
<code>UnregisterAsyncDataModule</code>	Indicates why the transferred data module is unregistered. The return values are: <ul style="list-style-type: none"> ● 0: The data module has been unregistered. ● 1: The slot is invalid. ● 2: The index is invalid. ● 3: The pointer to the data area is 0. ● 4: The data module is not registered.
<code>GetAsyncDataModule</code>	The data of a defined registered data module is read and returned. The return values are: <ul style="list-style-type: none"> ● 0: The data module has been read out successfully. ● 1: The slot is invalid. ● 2: The index is invalid. ● 3: No data module is registered with this slot and index.

Section 10.2

EtherNet/IP Adapter Configuration

Introduction

This section describes how to configure the EtherNet/IP adapter service of the VW3E70410000 communication module.

What Is in This Section?

This section contains the following topics:

Topic	Page
EtherNet/IP Adapter Configuration	120
Cyclic Data Exchange	124
Acyclic Data Exchange	125

EtherNet/IP Adapter Configuration

Introduction

This section describes the configuration of the EtherNet/IP adapter service.

The EtherNet/IP adapter supports the following exchange services:

- Cyclic data exchanges (*see page 124*)
- Acyclic data exchanges (*see page 125*)

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

Adding the EtherNet/IP Adapter

To add the EtherNet/IP adapter to your controller, select **EtherNet-IP-Adapter** in the **Hardware Catalog**, drag it to the **Devices tree** and drop it on your controller node.

NOTE: The **InputArea** and **OutputArea** nodes are added to the EtherNet/IP adapter node. These two I/O modules are used for cyclical communication.

Ethernet/IP Adapter Configuration

To access the EtherNet/IP adapter parameters, double-click **EtherNet-IP-Adapter** in the **Devices tree**:

Parameter	Type	Value	Default Value	Unit
Internal				
InitParameters				
Connector	Enumeration of INT	Front / 1	Front / 1	
not in use	Enumeration of BYTE	NOT_IN_USE	NOT_IN_USE	
IPAddressConfig	Enumeration of BYTE	manual / 0	manual / 0	
IPAddress	STRING(15)	'10.128.234.33'	'10.128.234.33'	
IPSubnetMask	STRING(15)	'255.255.240.0'	'255.255.240.0'	
IPGateway	STRING(15)	'10.128.224.10'	'10.128.224.10'	
EthernetAddress	STRING(13)	"	"	
EthernetConfig	Enumeration of BYTE	Auto-Negotiation / 0	Auto-Negotiation / 0	
InputLength	UDINT(1..504)	504	504	
OutputLength	UDINT(1..504)	504	504	
WdgEnabled	BOOL	FALSE	FALSE	
WdgTime	UDINT	100	100	
Info				
Diag				
ChannelCommonStatusBlock				
CommCOS	UDINT	2#0000_0000	2#0000_0000	
CommState	Enumeration of UDINT	UNKNOWN / 0	UNKNOWN / 0	
CommError	Enumeration of UDINT		SUCCESS / 0x0	
Version	UINT	0	0	
WdgTime	UINT	0	0	
Reserved0	ARRAY [0..1] OF UINT	[2(0)]	[2(0)]	
WdgTimeHost	UDINT	0	0	
ErrorCount	UDINT	0	0	

Parameters in black Read/write access

Parameters in gray Read-only access

The following parameters are provided in the **EtherNet/IP Configuration** tab:

Parameter	Type	Value	Default value	Description
Internal				
Reserved				

Parameter	Type	Value	Default value	Description
InitParameters				
IPAddressConfig	BYTE Enum	manual / 0 enable BOOTP / 1 enable DHCP / 2	manual / 0	Defines how the IP address of the module is set: <ul style="list-style-type: none"> ● 0 = Manually entered IP address is used. ● 1 = IP address is determined from the bootstrap protocol. ● 2 = IP address is determined from the DHCP protocol.
IPAddress	STRING	'10.128.234.33'	'10.128.234.33'	Specifies the IP address of the EtherNet/IP adapter.
IPSubnetMask	STRING	'255.255.240.0'	'255.255.240.0'	Specifies the subnet mask of the EtherNet/IP adapter.
IPGateway	STRING	'10.128.224.10'	'10.128.224.10'	Specifies the gateway address of the EtherNet/IP adapter.
EthernetAddress	STRING	"	"	Displays the MAC address of the EtherNet/IP adapter.
EthernetConfig	BYTE Enum	Auto-negotiation / 0 Full Duplex / 100 Mbit/s / 1 Full Duplex / 10 Mbit/s / 2 Half Duplex / 100 Mbit/s / 3 Half Duplex / 10 Mbit/s / 4	Auto-negotiation / 0	Displays the Ethernet configuration of the EtherNet/IP adapter: <ul style="list-style-type: none"> ● 0 = If this value is set, the device independently negotiates the connection parameters with the remote hub or the switch. ● 1 = The device works at 100 Mbit/s and in full duplex. ● 2 = The device works at 10 Mbit/s and in full duplex. ● 3 = The device works at 100 Mbit/s and in half duplex. ● 4 = The device works at 10 Mbit/s and in half duplex.
InputLength	UDINT	1...504	504	Defines the size of the input data of the cyclical data transfer in bytes (InputArea module in the Devices tree).
OutputLength	UDINT	1...504	504	Defines the size of the output data of the cyclical data transfer in bytes (OutputArea module in the Devices tree).

Parameter	Type	Value	Default value	Description
WdgEnabled	BOOL	TRUE FALSE	FALSE	Enables the watchdog: <ul style="list-style-type: none"> ● TRUE = The watchdog is activated. The value of the parameter <code>WdgTime</code> is used as watchdog timeout. ● FALSE = The watchdog is deactivated. The value of the parameter <code>WdgTime</code> is ignored.
WdgTime	UDINT	20...65535	100	Determines after which time a slave is detected as incommunicative by the watchdog. The time is given in ms.
Info				
IdentNumber	WORD	0	0	Displays the identification number of the EtherNet/IP adapter.
VendorName	STRING	'Schneider Electric'	'Schneider Electric'	Displays the vendor name of the EtherNet/IP adapter.
ModelName	STRING	'EtherNet/IP-Adapter'	'EtherNet/IP-Adapter'	Displays the model name of the EtherNet/IP adapter.
DriverInstance	DWORD	0	0	Displays the identifier for this driver instance.
Diag / ChannelCommonStatusBlock				
CommState	UDINT Enum	UNKNOWN / 0 NOT_CONFIGURED / 1 STOP / 2 IDLE / 3 OPERATE / 4	UNKNOWN / 0	Displays the current network status of the communication channel: <ul style="list-style-type: none"> ● 0 = Indeterminable. ● 1 = Not configured. ● 2 = Stopped. ● 3 = Idle. ● 4 = Operational.
CommError	UDINT Enum	Diagnostics Codes (<i>see page 114</i>)	SUCCESS / 0x0	Displays the current diagnostic code of the communication channel.
ErrorCount	UDINT	-	0	Displays the total number of errors detected since the last power-up or the last restart.

Cyclic Data Exchange

Instance ID of the Input/Output Areas

The input and output areas have the following instance ID:

Element	Instance ID	Size (bytes)	Description
Input assembly (InputArea)	101	0...504	Command word of master controller outputs (%QW)
Output assembly (OutputArea)	100	0...504	State of master controller inputs (%IW)

The number of exchanged data depends on the parameters `InputLength` and `OutLength` configured in the Ethernet/IP Configuration (*see page 121*).

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

I/O Modules of the EtherNet/IP Adapter

The **InputArea** and **OutputArea** nodes are added to the EtherNet/IP adapter node when you add the module. These two I/O modules are used for cyclical communication.

The data length of the 2 inserted I/O modules corresponds to the maximum data length for cyclical communication (504 bytes).

If, for example, `InputLength` is set to 50 and `OutputLength` is set to 20, the first 50 bytes in the I/O module **InputArea** and the first 20 bytes in the I/O module **OutputArea** are cyclically exchanged with the scanner.

The remaining 454 (504-50) input bytes and 484 (504-20) output bytes in the I/O modules are not used.

EtherNet/IP Modules I/O Mapping

Double-click the **InputArea** or **OutputArea** node in the **Devices tree**

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

For further generic descriptions, refer to I/O Mapping Tab Description (*see SoMachine, Programming Guide*).

Acyclic Data Exchange

Registering for Callback

With the EtherNet/IP adapter it is possible to register for a callback in the case of a noncyclical inquiry. If a noncyclical inquiry of the scanner is received by the EtherNet/IP adapter driver, the inquiry is first passed on to registered data areas.

Then, all registered user function block (FB) instances are called up which have been registered with the EtherNet/IP adapter by using the interfaces `IF_EIPEventHandler_AsyncGetAttributeAll`, `IF_EIPEventHandler_AsyncGetAttributeSingle` and `IF_EIPEventHandler_AsyncSetAttributeSingle`. At the same time, the state of the automatic inquiry processing is transferred as well, so that possible problems can be responded to. The current status is then written into parameter `iq_udiError`.

NOTE: The `RegisterAsyncClass()` method has to be called up in order to obtain the callbacks of a certain class in the callback method.

NOTE: If noncyclical data areas are registered, this method is automatically executed internally. In this case, you do not need to call up the `RegisterAsyncClass()` method manually.

A list of all relevant methods can be found here after.

NOTE: The `IoDrvEtherNetIPAdapter` library is added to the library manager when you add the module. This library contains the interfaces, methods, and function blocks for managing the EtherNet/IP module.

Relevant Methods for Callback Registration

This table lists the relevant methods for the registration of the data areas of the `IF_EtherNet-IP_Adapter` interface:

Method	Description
<code>IsRegisteredAsyncGetAttributeAll</code>	Returns whether the transferred function block is already registered for the callback upon arrival of a <code>GetAttributeAll</code> inquiry.
<code>IsRegisteredAsyncGetAttributeSingle</code>	Returns whether the transferred function block is already registered for the callback upon arrival of a <code>GetAttributeSingle</code> inquiry.
<code>IsRegisteredAsyncSetAttributeSingle</code>	Returns whether the transferred function block is already registered for the callback upon arrival of a <code>SetAttributeSingle</code> inquiry.
<code>RegisterAsyncGetAttributeAll</code>	Registers the transferred function block for the callback in the case of <code>GetAttributeAll</code> inquiries.
<code>RegisterAsyncGetAttributeSingle</code>	Registers the transferred function block for the callback in the case of <code>GetAttributeSingle</code> inquiries.
<code>RegisterAsyncSetAttributeSingle</code>	Registers the transferred function block for the callback in the case of <code>SetAttributeSingle</code> inquiries.

Method	Description
<code>UnregisterAsyncGetAttributeAll</code>	Unregisters the transferred function block for the callback in the case of <code>GetAttributeAll</code> inquiries.
<code>UnregisterAsyncGetAttributeSingle</code>	Unregisters the transferred function block for the callback in the case of <code>GetAttributeSingle</code> inquiries.
<code>UnregisterAsyncSetAttributeSingle</code>	Unregisters the transferred function block for the callback in the case of <code>SetAttributeSingle</code> inquiries.
<code>RegisterAsyncClass</code>	<p>Enables the callbacks for the transferred <code>ClassId</code>. Only when the callbacks for a certain <code>ClassId</code> have been activated, the code registered for the callbacks is called up.</p> <p>The return values are:</p> <ul style="list-style-type: none"> ● 0: The <code>ClassId</code> has been registered successfully. ● 1: The <code>ClassId</code> is already registered. ● : The <code>ClassId</code> is invalid. ● 3: The inquiry <code>RegisterAsyncClass</code> request cannot be sent. ● 4: The inquiry <code>RegisterAsyncClass</code> request was not correct.

Registering Noncyclical Data Areas

Data areas for noncyclical inquiries can be registered with the EtherNet/IP adapter. Noncyclical inquiries (`Get_Attribute_All`, `Get_Attribute_Single` and `Set_Attribute_Single`) to registered data areas are then automatically processed by the EtherNet/IP adapter. A noncyclical data area is addressed by using a `ClassId`, an `InstanceId` and an `AttributeId`.

For this purpose, an attribute of type `ST_EtherNetIPAttribute` has to be created. This attribute has to contain the following program contents: an `Id`, a length, and a pointer to a data area (for example, an array).

Ensure that the area to which the pointer of an attribute points actually exists during operation and is not deleted. For this purpose, for example, declare the array as a variable in the program.

By using the interface `IF_EtherNetIPInstance`, the attribute is then registered with an instance of the function block `FB_EtherNetIPInstance`

NOTE: Ensure that the instances of the function block `FB_EtherNetIPInstance` as well as the attributes are available permanently and not accidentally deleted during operation (for example, local variables of a method are deleted upon completion of the method).

In order to register the instance with the EtherNet/IP adapter, the created noncyclical instance has to contain a `ClassId` and an `InstanceId`.

These can be set using the methods `SetClassId` and `SetInstanceId`.

The instances are registered by using the `IF_EtherNetIP_Adapter` interface of the Ethernet/IP adapter.

A list of all relevant methods can be found here after.

Relevant Methods for Data Area Registration

This table lists the relevant methods for the registration of the data areas of the `IF_EtherNetIP_Adapter` and `IF_EtherNetIPInstance` interfaces:

Method	Description
IF_EtherNetIP_Adapter interface	
<code>RegisterInstance</code>	Registers the instance with the EtherNet/IP adapter. The return values are: <ul style="list-style-type: none"> ● 0: Instance is registered. ● 1: Instance is already registered. ● 2: The instance is invalid. ● 3: Internal error.
<code>UnregisterAllInstances</code>	Unregisters all instances that have been registered with the EtherNet/IP adapter. The return values are: <ul style="list-style-type: none"> ● 0: All registrations have been removed. ● 1: Internal error. ● 2: There are no registered instances. ● 3: Internal error.
<code>UnregisterInstance</code>	Unregisters a defined instance in the EtherNet/IP adapter. The return values are: <ul style="list-style-type: none"> ● 0: The registration has been made void. ● <code>CmpErrors.Errors.ERR_NO_OBJECT</code>: The last instance was not found. ● <code>CmpErrors.Errors.ERR_INVALID_HANDLE</code>: Internal error.
IF_EtherNetIPInterface interface	
<code>SetClassId</code>	Assigns a <code>ClassId</code> to an instance of the function block <code>FB_EtherNetIPInstance</code> . The return values are: <ul style="list-style-type: none"> ● 0: The <code>ClassId</code> is valid and set. ● 1: The <code>ClassId</code> is not between <code>Gc_EtherNetIPClass_MinID</code> and <code>Gc_EtherNetIPClass_MaxID</code>.
<code>SetInstanceId</code>	Assigns a <code>InstanceId</code> to an instance of the function block <code>FB_EtherNetIPInstance</code> . The return values are: <ul style="list-style-type: none"> ● 0: The <code>InstanceId</code> is valid and set. ● 1: The <code>ClassId</code> is not between 0 and <code>Gc_EtherNetIPClass_MaxID</code>.

Method	Description
AddAttribute	Adds the transferred attribute to the instance of the function block <code>FB_EtherNetIPInstance</code> . The return values are: <ul style="list-style-type: none"> ● 0: The attribute has been added. ● 1: The <code>AttributeId</code> is already used. ● 2: The attribute is invalid.
AddAttributeArray	Splits an array up into several parts that have the size of <code>i_byAttributeLength</code> and adds every part as attribute to the instance of the function block <code>FB_EtherNetIPInstance</code> . The attributes IDs are numbered consecutively. The <code>o_byStartAttributeId</code> returns the <code>AttributeId</code> of the first added attribute. The length of the last added attribute may be less than the transfer parameter <code>i_byAttributeLength</code> . The return values are: <ul style="list-style-type: none"> ● 0: All attributes have been added and no errors were detected. ● 1: Not enough available space in the instance.
RemoveAttribute	Removes an attribute with a defined ID. The return values are: <ul style="list-style-type: none"> ● 0: The attribute was removed successfully. ● <code>CmpErrors.Errors.ERR_INVALIDID</code>: The attribute was not found.
RemoveAllAttributes	Removes all attributes within the instance and returns the number of removed attributes.
GetClassId	Returns the <code>ClassId</code> .
GetInstanceId	Returns the <code>InstanceId</code> .
GetAttribute	Returns the data of the transferred attribute ID in the in/out parameter <code>iq_stAttribute</code> . Return value <code>CmpErrors.Errors.ERR_INVALIDID</code> : The attribute was not found.
GetAttributeCount	Returns the number of attributes.
GetInstanceFreeSize	Returns the size of the free space of the instance.
IsValid	Indicates whether the instance information is valid. For this purpose, <code>ClassID</code> and <code>InstanceID</code> have to be set.

Method	Description
GetFirstAttribute	Returns the data of the first attribute of the instance in the in/out parameter <code>iq_stAttribute</code> . The return values are: <ul style="list-style-type: none">● 0: Attributes have been found.● 1: No attributes have been found.
GetNextAttribute	Returns the data of the next attribute of the instance in the in/out parameter <code>iq_stAttribute</code> . The return values are: <ul style="list-style-type: none">● 0: Attributes have been found.● 1: No attributes have been found.

Section 10.3

Ethernet/IP Scanner Configuration

Introduction

This section describes how to configure the EtherNet/IP Scanner service of the VW3E704100000 Ethernet communication module.

What Is in This Section?

This section contains the following topics:

Topic	Page
Presentation	131
Supported Devices	132
EtherNet/IP Scanner Configuration	134
EtherNet/IP Scanner I/O Mapping	136
EtherNet/IP Scanner Status and Diagnostics	137
Target Device Declaration	139
Target Settings	141
Connection Configuration	143
Device Replacement with User Parameters	159
EtherNet/IP I/O Mapping	163

Presentation

EtherNet/IP Overview

EtherNet/IP is the name given to the Common Industrial Protocol (CIP)-based protocol implemented over standard Ethernet. For further information about CIP, refer to the www.odva.org website.

EtherNet/IP uses an Originator/Target architecture for data exchange:

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

Communication between EtherNet/IP originators and targets is accomplished using EtherNet/IP connections.

EtherNet/IP Scanner Features

The EtherNet/IP Scanner on the LMC078 Motion Controller is an originator device that establishes connections to, and exchanges configuration information and input/output data with, target devices. For example, the scanner might indicate to a target device how often the device should transmit its input data and how often it expects output data from the scanner.

The following table presents the features of the EtherNet/IP Scanner service for the LMC078 Motion Controller:

Feature	Description
Performance	Up to 64 EtherNet/IP target devices managed by the controller, monitored within a timeslot of 10 ms
Number of connections	1...64
Number of input words	0...5712. Maximum of 504 for each target device
Number of output words	0...5712. Maximum of 504 for each target device
I/O communications	EtherNet/IP I/O scanner service Function block for configuration and data transfer Originator/Target
Other services	EDS file management

Supported Devices

Supported Devices

This table presents the target devices supported by the EtherNet/IP Scanner:

Device name		TVDA	Key features
Predefined devices	Altivar 32	X	Libraries, predefined connections, predefined data exchanges
	Altivar 320	X	Libraries, predefined connections, predefined data exchanges
	Altivar 340	X	Libraries, predefined connections, predefined data exchanges
	Altivar 6••	X	Libraries, predefined connections, predefined data exchanges
	Altivar 71	X	Libraries, predefined connections, predefined data exchanges
	Altivar 9••	X	Libraries, predefined connections, predefined data exchanges
	Lexium 32 M	X	Libraries, predefined connections, predefined data exchanges
	Lexium ILA	X	Libraries, predefined connections, predefined data exchanges
	Lexium ILE	X	Libraries, predefined connections, predefined data exchanges
	Lexium ILS	X	Libraries, predefined connections, predefined data exchanges
	OsiSense XG	X	Predefined connections, predefined data exchanges
	OsiSense XUW	X	Predefined connections, predefined data exchanges
	XPSMCM	X	Predefined connections, predefined data exchanges
Other devices	Device provided with EDS file ⁽¹⁾	-	User parameters, predefined connections
	Generic slave device ⁽²⁾	-	User parameters
<p>(1) An EDS file provides, among other things, predefined connections to facilitate network integration. (2) A generic slave device is used in SoMachine to add EtherNet/IP devices that do not have predefined connections, such as speed drives, sensors, or other controllers.</p>			

Key Features

This table presents the key features:

Key features	Description
Libraries	Functions and function blocks (dedicated to the device) available for use by the application. Refer to the Motion Control Library Guide (<i>see page 9</i>).
Predefined connections	Used to set up cyclic data exchanges. Select one of the proposed connections containing the relevant information. For more information, refer to Connection Configuration (<i>see page 143</i>).
Predefined data exchanges	The cyclic data exchanges are set automatically: one predefined connection is automatically selected when you add the device to the project.
User parameters	Parameters that are sent automatically to the device at power-up. These parameters are used when replacing devices that do not support FDR.

TVDA

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

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

For more details, refer to TVDA Device Module Library, Function Template Library Guide (*see TVDA Device Module Library, Function Template Library Guide*).

EtherNet/IP Scanner Configuration

Adding the EtherNet/IP Scanner

To add the EtherNet/IP scanner to your controller, select **Ethernet IP Scanner board** in the **Hardware Catalog**, drag it to the **Devices tree** and drop it on your controller node.

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

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

EtherNet/IP Scanner Configuration

To configure the EtherNet/IP Scanner, double-click the **EtherNet IP Scanner** node in the **Devices tree**:

The screenshot shows the configuration window for the EtherNet_IP_Scanner. It has a title bar with a close button and a tab labeled 'EtherNet_IP_Scanner x'. Below the title bar are four tabs: 'Scanner settings' (selected), 'EtherNet/IP I/O Mapping', 'Status', and 'Information'. The main content area is divided into several sections:

- Address Settings:**
 - Use static IP-Address
 - IP Address: 192 . 168 . 0 . 1
 - Subnet Mask: 255 . 255 . 255 . 0
 - Gateway Address: 0 . 0 . 0 . 0
 - Obtain IP-Address automatically
 - BOOTP
 - DHCP
- Ethernet Settings:**
 - Speed & Duplex: Auto-negotiation (dropdown menu)
- Options:**
 - Auto-reestablish connections

The following parameters are provided in the **Scanner Settings** tab:

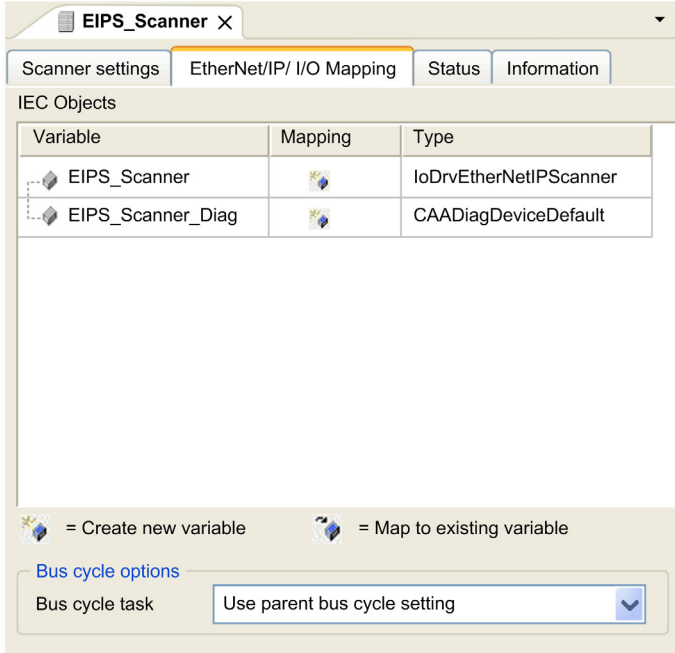
Parameter	Default value	Description
IP Address	192.168.0.1	Specify the IP address of the EtherNet/IP Scanner.
Subnet Mask	255.255.255.0	Specify the subnet mask of the EtherNet/IP Scanner.
Gateway Address	0.0.0.0	Specify the gateway address of the EtherNet/IP Scanner.
Speed&Duplex	Auto-negotiation	<p>Select the data transmission direction and speed of the EtherNet/IP Scanner:</p> <p>Auto-negotiation. The scanner independently negotiates the connection parameters with the remote hub or switch.</p> <p>Full Duplex /100 Mbit/s. The scanner works at 100 Mbit/s and in full duplex.</p> <p>Full Duplex /10 Mbit/s. The scanner works at 10 Mbit/s and in full duplex.</p> <p>Half Duplex /100 Mbit/s. The scanner works at 100 Mbit/s and in half duplex.</p> <p>Half Duplex /10 Mbit/s. The scanner works at 10 Mbit/s and in half duplex.</p>

EtherNet/IP Scanner I/O Mapping

Configuring the EtherNet/IP Scanner I/O Mapping

I/O bus configuration allows you to select the task that controls EtherNet/IP cyclic data exchanges.

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

Step	Action
1	In the Devices tree , double-click EtherNet IP Scanner . Result: The configuration window is displayed.
2	Select the EtherNet/IP I/O Mapping tab. 
3	The Bus cycle task parameter defines the task responsible for refreshing the I/O images (%QB, %IB). These I/O images correspond to the EtherNet/IP requests sent to the target devices and the health bits. Select the Bus cycle task to use: <ul style="list-style-type: none"> ● Use parent bus cycle setting (the default). Use the task specified in the PLC Settings (see page 88) tab of the controller. ● MAST. Use the MAST task (see page 49). ● Motion. Use the Motion task (see page 42).

EtherNet/IP Scanner Status and Diagnostics

Introduction

In online mode, the **Status** tab of the EtherNet/IP Scanner provides monitoring and diagnostics information for the EtherNet/IP Scanner and connected devices.

Displaying Monitoring and Diagnostics Information

Step	Action																														
1	In the Devices tree , double-click EtherNet IP Scanner .																														
2	<p>Select the Status tab. Result: The Last Diagnostic Message window is displayed:</p> <p>EtherNet/IP : Diagnostic message available</p> <p>Last diagnostic message: Acknowledge</p> <table border="1"> <tr> <td>MasterDiag</td> <td></td> <td>EthernetIP master diagnostic information</td> </tr> <tr> <td> + CommunicationCOS</td> <td>143</td> <td></td> </tr> <tr> <td> ... CommunicationState</td> <td>4</td> <td>0 = Unknown, 1 = Not Configured, 2 = Stop, 3 = Idle, 4 Operate</td> </tr> <tr> <td> ... Version</td> <td>1</td> <td>Version number of diagnosis structure</td> </tr> <tr> <td> ... Watchdog</td> <td>0</td> <td>Configured Watchdog Timeout</td> </tr> <tr> <td> ... ErrorCount</td> <td>1</td> <td>Total number of detected errors since startup</td> </tr> <tr> <td> ... SlaveState</td> <td>1</td> <td>0 = Unknown, 1 = OK, 2 = Failed, 3 = Warning</td> </tr> <tr> <td> ... NumConfigSlaves</td> <td>6</td> <td>Number of configured slaves</td> </tr> <tr> <td> ... NumActiveSlaves</td> <td>5</td> <td>Number of active slaves</td> </tr> <tr> <td> ... NumDiagSlaves</td> <td>1</td> <td>Number of slaves, reporting diagnostic issues.</td> </tr> </table>	MasterDiag		EthernetIP master diagnostic information	+ CommunicationCOS	143		... CommunicationState	4	0 = Unknown, 1 = Not Configured, 2 = Stop, 3 = Idle, 4 Operate	... Version	1	Version number of diagnosis structure	... Watchdog	0	Configured Watchdog Timeout	... ErrorCount	1	Total number of detected errors since startup	... SlaveState	1	0 = Unknown, 1 = OK, 2 = Failed, 3 = Warning	... NumConfigSlaves	6	Number of configured slaves	... NumActiveSlaves	5	Number of active slaves	... NumDiagSlaves	1	Number of slaves, reporting diagnostic issues.
MasterDiag		EthernetIP master diagnostic information																													
+ CommunicationCOS	143																														
... CommunicationState	4	0 = Unknown, 1 = Not Configured, 2 = Stop, 3 = Idle, 4 Operate																													
... Version	1	Version number of diagnosis structure																													
... Watchdog	0	Configured Watchdog Timeout																													
... ErrorCount	1	Total number of detected errors since startup																													
... SlaveState	1	0 = Unknown, 1 = OK, 2 = Failed, 3 = Warning																													
... NumConfigSlaves	6	Number of configured slaves																													
... NumActiveSlaves	5	Number of active slaves																													
... NumDiagSlaves	1	Number of slaves, reporting diagnostic issues.																													

Diagnostics Information

The **CommunicationCOS** field is a binary-coded decimal value:

Bit	Description when the bit is set to 1
0	Communication ready
1	Configuration performed correctly
2	Protocol open for communication
3	Configuration locked
4	New configuration
5	Communication restart requested
6	Communication restart enabled
7	Direct Memory Access enabled
8...31	Not used

For example, for a value of **CommunicationCOS** of 143 (Operational State), the following bits are set to 1:

- Bit 0 (Communication ready)
- Bit 1 (Configuration performed correctly)
- Bit 2 (Protocol open for communication)
- Bit 3 (Configuration locked)
- Bit 7 (Direct Memory Access enabled)

Target Device Declaration

Overview

This section describes how to add a target device on the EtherNet/IP Scanner.

The available target devices are listed in Supported Devices (*see page 132*).

Automatic Settings

During each target device declaration, SoMachine automatically:

- Sets the network settings (IP address, subnet mask, gateway address) in accordance with the EtherNet/IP Scanner IP settings (*see page 134*).
- For predefined devices, creates predefined data exchanges.

Adding a Target Device

To add a target device on the **Ethernet-IP-Scanner** node, select the device in the **Hardware Catalog**, drag it to the **Devices tree**, and drop it on the **Ethernet-IP-Scanner** node.

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

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

Depending on the target device you add to your project, some libraries may be loaded automatically. Refer to the Motion Control Library Guide (*see page 9*) for the available function blocks.

Adding a Target Device from a Template

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

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

To add a device from a template, proceed as follows:

Step	Action
1	In the Hardware Catalog , select the Device Template check box.
2	Select the device in the Hardware Catalog , drag it to the Devices tree , and drop it on the Ethernet-IP-Scanner node.

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

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

Adding a Target Device from an EDS File

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

To add a device with an EDS file, proceed as follows:

Step	Action
1	In the SoMachine menu, select Tools → Device Repository .
2	Click Install to open the Device description files dialog box.
3	Select EDS and DCF files in the file type list.
4	Select the EDS file.
5	Click OK to close the dialog box.
6	Click Close to close the Install Device Description dialog box.
7	Select the Ethernet-IP-Scanner and click the Plus button. Select the newly added target device and click Add Device . For more details, refer to Using the Contextual Menu or Plus Button (<i>see SoMachine, Programming Guide</i>)

Target Settings

Overview

Once a target device has been added in the EtherNet/IP Scanner, use the **Target Settings** tab of the device to edit the network settings of the device.

Target Device Settings

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

The screenshot shows the 'Target settings' tab for a device named 'Altivar_71'. The interface includes several sections:

- Address Settings:**
 - IP Address: 192 . 168 . 0 . 2
 - BOOTP
 - MAC Address: 00 : 00 : 00 : 00 : 00 : 00
 - Store IP-Address
- Electronic Keying:**
 - Check Device Type: 2
 - Check Vendor Code: 243
 - Check Product Code: 5
 - Check Major Revision: 1
 - Check Minor Revision: 1
 - Restore default values button
- Protocol on the fieldbus:**
 - Protocol used by the device: EtherNet/IP
 - Text: This is the protocol used between the logic controller and the device.

Address Settings

Target devices added to the EtherNet/IP Scanner must have a fixed IP address.

Enter the IP address of the device in the **IP Address** field.

If replacing a device:

Step	Action
1	Install the new device.
2	In the device, configure the network settings (IP address, subnet mask, and gateway address).
3	Configure the parameters in the device directly or using SoMachine.
4	Power up the device and launch the application.

Electronic Keying

Electronic Keying signatures are used to identify the device.

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

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

If the device values are not the same as the application values, the logic controller no longer communicates with the device. This can be monitored within your application via diagnostic information (*see page 137*) in order to take appropriate actions within the context of your machine.

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

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

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

Connection Configuration

Connection Overview

To access an EtherNet/IP device, it is necessary to configure connections. A connection allows the exchange of blocks of data combined into assemblies.

The starting and stopping of connections is managed by the controller.

Assemblies

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

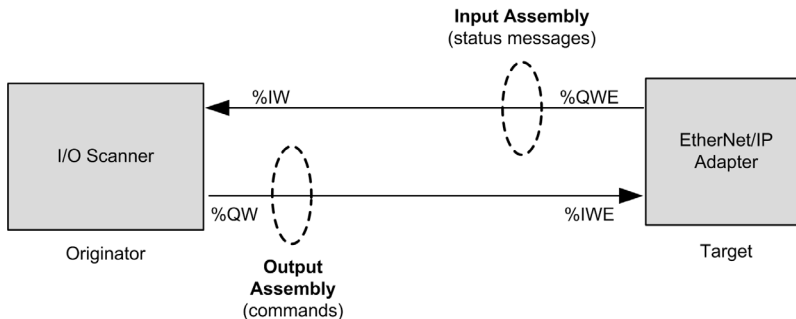
Data (attributes) from different objects can be combined into a single object to allow data to be sent or received over a single connection.

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

Assembly objects are structured into classes, instances, and attributes:

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

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



The EtherNet/IP configuration parameters are defined as:

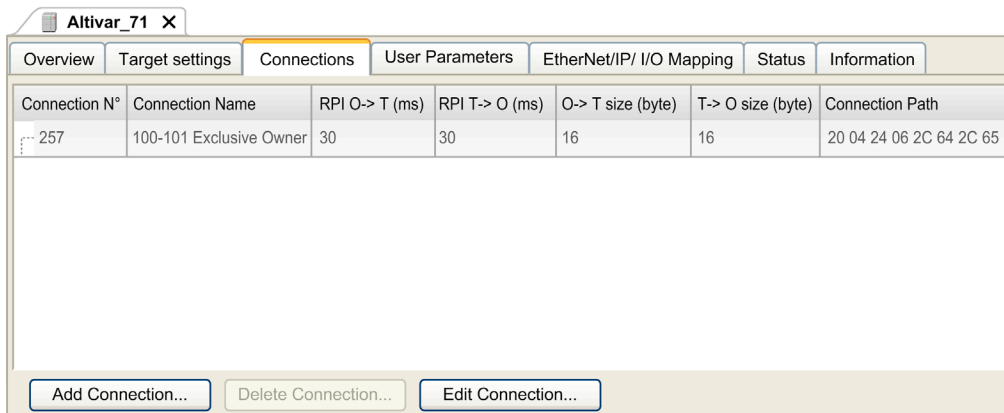
- **Instance:** Number referencing the assembly.
- **Size:** Number of channels of an assembly.

The memory size of each channel is 2 bytes, which store the value of $\%IB_x$ or $\%QB_x$ objects, where x is the channel number.

For example, if the **Size** of the **Output Assembly** is 20, there are 20 input channels ($IB_0 \dots IB_{19}$) addressing $\%IB_y \dots \%IB_{(y+20-1)}$, where y is the first available channel for the assembly.

Configuring Device Connections

To create and configure connections, double-click an EtherNet/IP target device in the **Devices tree** and select the **Connections** tab:



Column	Comment
Connection N°	The connection number is unique. It is automatically assigned by SoMachine.
Connection Name	The default connection name is generated automatically by SoMachine. For predefined connections (<i>see page 132</i>), this name cannot be edited. For other connections, the default connection name can be edited on the Edit Connect window (<i>see page 154</i>).
RPI O->T (ms)	Requested Packet Interval: The time period between cyclic data transmissions requested by the EtherNet/IP Scanner (O --> T) or by the target device (T --> O).
RPI T->O (ms)	
O->T size (byte)	Number of bytes to exchange between the Originator (O) and the Target (T).
T->O size (byte)	
Config#1 size (byte)	Size of the first set of configuration parameters. Only displayed for devices with configurable parameters.
Config#2 size (byte)	Size of the second set of configuration parameters. Only displayed for device with configurable parameters.
Connection Path	Coded transcription of the other connection parameters.

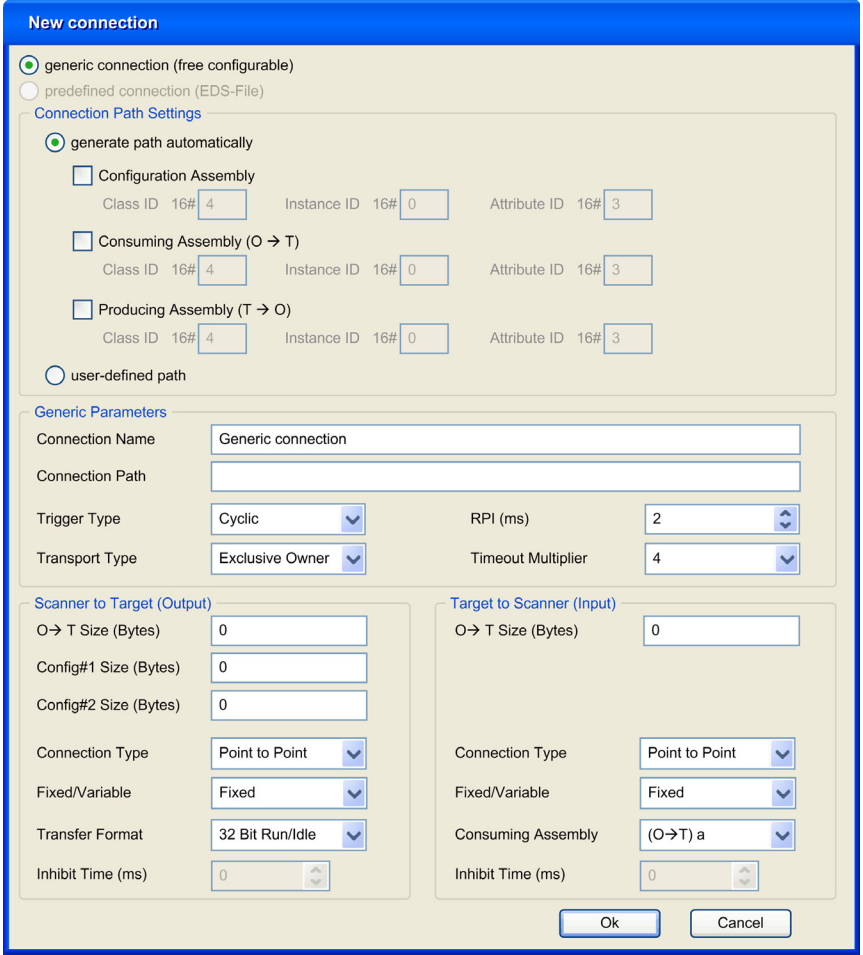
To create a connection, click **Add Connection**. See “Adding an EtherNet/IP Connection” that follows.

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

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

Adding an EtherNet/IP Connection

To create and configure a connection, proceed as follows:

Step	Action
1	In the Devices tree , double-click an EtherNet/IP target device.
2	Select the Connections tab.
3	Click Add Connection .
4	<p>Select generic connection (free configurable):</p>  <p>(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation.</p>

Step	Action
5	Select generate path automatically and configure the Configuration Assembly : <ul style="list-style-type: none"> ● Class ID (4 by default): Class identifier⁽¹⁾ ● Instance ID: Instance identifier⁽¹⁾ ● Attribute ID (3 by default): Attribute identifier⁽¹⁾
6	Configure the Consuming Assembly (O --> T) : <ul style="list-style-type: none"> ● Class ID (4 by default): Class identifier⁽¹⁾ ● Instance ID: Instance identifier⁽¹⁾ ● Attribute ID (3 by default): Attribute identifier⁽¹⁾
7	Configure the Producing Assembly (T --> O) : <ul style="list-style-type: none"> ● Class ID (4 by default): Class identifier⁽¹⁾ ● Instance ID: Instance identifier⁽¹⁾ ● Attribute ID (3 by default): Attribute identifier⁽¹⁾

⁽¹⁾ The Class ID, Instance ID, and Attribute ID can be found in the device documentation.

Step	Action
8	<p>Configure the Generic Parameters:</p> <ul style="list-style-type: none"> ● Connection Name. The default connection name is generated automatically by SoMachine. For generic connections, the default name “generic connection” can be edited. ● Connection Path. Coded transcription of the physical link object. Can be edited for generic connections. ● Trigger Type. Select how the exchange of data is initiated: <ul style="list-style-type: none"> ○ Cyclic: Endpoints exchange data at regular, predetermined time intervals. ○ Change of state: Endpoints only exchange data when the data changes. To keep the connection from timing out if no changes occur, the data is also exchanged at the background cyclic interval (see RPI below). When Change of state is selected, the Inhibit Time (ms) fields of the scanner-to-target and target-to-scanner connection properties are enabled. ○ Application. The exchange of data is triggered by an application. ● Transport Type: <ul style="list-style-type: none"> ○ Exclusive Owner: This is a bidirectional connection to an output connection point (typically an Assembly Object), where the data of this assembly can only be controlled by one scanner. There may be a connection to an input assembly; this data is being sent to the scanner. ○ Listen Only: The scanner receives input data from the target device and produces a Heartbeat to the target device. There is no output data. A Listen Only connection can only be attached to an existing Exclusive Owner or Input Only connection. If this underlying connection stops, then the Listen Only connection is also stopped or timed out. ○ Input Only: The scanner receives input data from the target device and produces a Heartbeat to the target device. There is no output data. ● RPI (ms) Requested Packet Interval. The time period between cyclic data transmissions requested by the scanner. Target devices always provide a minimum RPI, whereas in the controller the goal is to have the highest RPI in order to not overload the system. Each time a device is added to the EtherNet/IP fieldbus, or each time an RPI value is modified, verify the level of controller resources used by the target devices. The device RPI may be specified in the device documentation, but is usually provided as part of the EDS file delivered with the device. ● Select the Timeout Multiplier: 4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512. The selected value is multiplied by the RPI value to obtain the connection time-out value.
(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation.	

Step	Action
9	Configure Scanner to Target (Output) -specific parameters: <ul style="list-style-type: none"> ● O → T Size (Bytes): Number of bytes to transfer. ● Config#1 Size (Bytes): Number of parameters in the first set of configuration parameters. ● Config#2 Size (Bytes): Number of parameters in the second set of configuration parameters. ● Connection Type. Connection type to use: <ul style="list-style-type: none"> ○ Multicast. Connection is between the scanner and multiple target devices. ○ Point to Point. Connection is between the scanner and a single target device. ● Fixed/Variable. Whether the amount of data transmitted is always the same (Fixed) or only the exact amount of buffered data is transmitted (Variable). ● Transfer Format. The real-time data format to use on the connection: <ul style="list-style-type: none"> ○ 32 Bit Run/Idle. A 32-bit packet header includes run/idle notification. ○ pure Data. No run/idle notification. ○ Heartbeat. No run/idle notification. ○ Idle with zero length. Zero-length data format indicates idle. For details, refer to the OVDA website . <ul style="list-style-type: none"> ● Inhibit Time (ms): Minimum period of time between 2 data exchanges. Accessible only if Trigger Type is Change of state. The value must be a multiple of 2 ms. The maximum value is the RPI (ms) value, up to a maximum possible value of 254 ms.
10	Configure Target to Scanner (Input) -specific parameters: <ul style="list-style-type: none"> ● T → O Size (Bytes): Number of bytes to transfer. ● Connection Type. Connection type to use: <ul style="list-style-type: none"> ○ Multicast. Connection is between the target and multiple scanners. ○ Point to Point. Connection is between the target and a single scanner. ● Fixed/Variable. Whether the amount of data transmitted is always the same (Fixed) or only the exact amount of buffered data is transmitted (Variable). ● Consuming Assembly (O → T) ● Inhibit Time (ms): Minimum period of time between 2 data exchanges. Accessible if Trigger Type is Change of state. The value must be a multiple of 2 ms. The maximum value is the RPI (ms) value, up to a maximum possible value of 254 ms.
11	Click OK .
(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation.	

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

NOTE: Due to the **O → T Size (Bytes)** and **T → O Size (Bytes)** limitations and the maximum input/output words ([see page 131](#)) of the scanner, verify the scanner resources load after adding a connection.

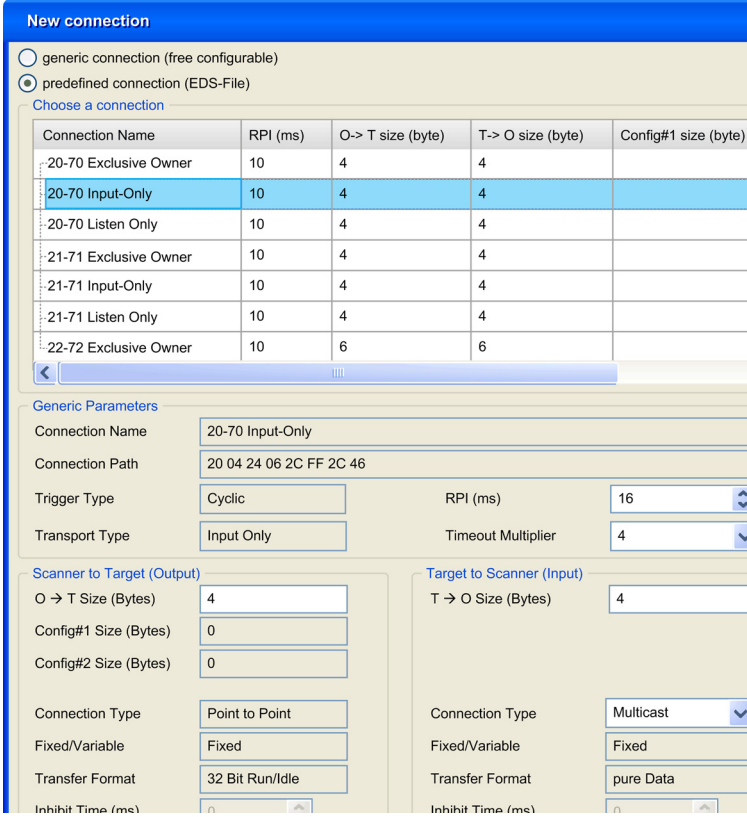
Add a Predefined Connection

Predefined connections (refer to Supported Devices (*see page 131*)) are available for:

- Predefined devices
- Other devices that are delivered with an EDS file.

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

To add a predefined connection, proceed as follows:

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

Step	Action
6	Configure the Generic Parameters : <ul style="list-style-type: none"> ● RPI (ms) Requested Packet Interval. The period of time between cyclic data transmissions requested by the scanner. The default value is defined in the EDS. ● Select the Timeout Multiplier: 4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512. The selected value is multiplied by the RPI value to obtain the connection time-out.
7	Configure the Scanner to Target (Output) : <ul style="list-style-type: none"> ● O --> T Size (Bytes): Number of bytes to transfer.
8	Configure the Target to Scanner (Input) : <ul style="list-style-type: none"> ● T --> O Size (Bytes): Number of bytes to transfer (Number of channels of the assembly) ● Select the Connection Type: Multicast (the default) if the connection is between the scanner and multiple target devices, or Point-to-Point if the connection is between the scanner and a single target device. Only editable for certain types of Transport Type.
9	Click OK . Result : The connection is added to the Connections tab.

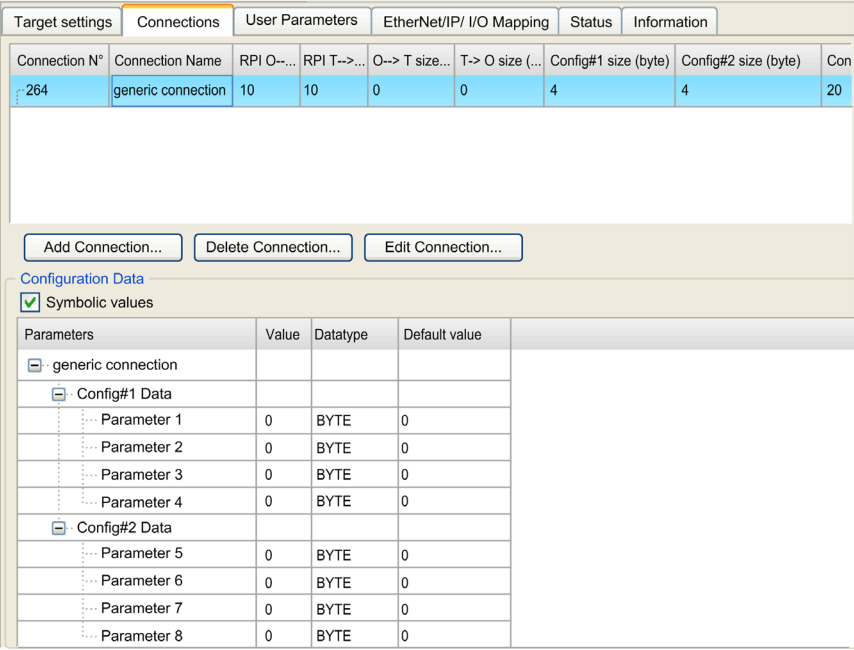
Configuring a Configuration Assembly

Some devices support a configuration assembly.

A configuration assembly is a single request, sent when the EtherNet/IP Scanner starts up, that sends all configuration parameters to the target device.

To configure a configuration assembly, proceed as follows:

Step	Action
1	In the Devices tree , double-click an EtherNet/IP device.
2	Select the Connections tab.
3	Select an existing connection and click Edit Connection .
4	Select generic connection (free-configurable) .
5	Select Configuration Assembly .
6	Configure the Configuration Assembly : <ul style="list-style-type: none"> ● Class ID (4 by default): Class identifier⁽¹⁾ ● Instance ID: Instance identifier⁽¹⁾ ● Attribute ID (3 by default): Attribute identifier⁽¹⁾
7	Configure the Scanner to Target (Output) : <ul style="list-style-type: none"> ● Config#1 Size (Bytes): Number of parameters in the first set of configuration parameters. ● Config#2 Size (Bytes): Number of parameters in the second set of configuration parameters.

Step	Action																																																
8	<p>Click OK.</p> <p>Result: The configuration parameters are displayed in the Connections tab:</p>  <p>Configuration Data</p> <input checked="" type="checkbox"/> Symbolic values <table border="1" data-bbox="381 581 879 906"> <thead> <tr> <th>Parameters</th> <th>Value</th> <th>Datatype</th> <th>Default value</th> </tr> </thead> <tbody> <tr> <td>- generic connection</td> <td></td> <td></td> <td></td> </tr> <tr> <td>- Config#1 Data</td> <td></td> <td></td> <td></td> </tr> <tr> <td> Parameter 1</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td> Parameter 2</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td> Parameter 3</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td> Parameter 4</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td>- Config#2 Data</td> <td></td> <td></td> <td></td> </tr> <tr> <td> Parameter 5</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td> Parameter 6</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td> Parameter 7</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> <tr> <td> Parameter 8</td> <td>0</td> <td>BYTE</td> <td>0</td> </tr> </tbody> </table>	Parameters	Value	Datatype	Default value	- generic connection				- Config#1 Data				Parameter 1	0	BYTE	0	Parameter 2	0	BYTE	0	Parameter 3	0	BYTE	0	Parameter 4	0	BYTE	0	- Config#2 Data				Parameter 5	0	BYTE	0	Parameter 6	0	BYTE	0	Parameter 7	0	BYTE	0	Parameter 8	0	BYTE	0
Parameters	Value	Datatype	Default value																																														
- generic connection																																																	
- Config#1 Data																																																	
Parameter 1	0	BYTE	0																																														
Parameter 2	0	BYTE	0																																														
Parameter 3	0	BYTE	0																																														
Parameter 4	0	BYTE	0																																														
- Config#2 Data																																																	
Parameter 5	0	BYTE	0																																														
Parameter 6	0	BYTE	0																																														
Parameter 7	0	BYTE	0																																														
Parameter 8	0	BYTE	0																																														
9	<p>Double-click in the Value column to set the configuration parameter values.</p>																																																
<p>(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information (see page 158).</p>																																																	

Editing Predefined Connections

To edit a predefined connection, select the connection in the **Connections** tab and click **Edit Connection**:

Parameter	Description
Generic Parameters	
RPI (ms)	RPI (ms) Requested Packet Interval. The time period between cyclic data transmissions requested by the scanner. The device always provides a minimum RPI, whereas in the controller the goal is to have the highest RPI in order to not overload the system. Each time a device is added to the EtherNet/IP fieldbus, or each time an RPI value is modified, verify the resources used by the target devices. The device RPI may be specified in the device documentation but is usually provided as part of the EDS file delivered with the device.
Timeout Multiplier	Select the Timeout Multiplier : 4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512. The selected value is multiplied by the RPI value to obtain the connection time-out value.

Parameter	Description
Scanner to Target (Output)	
O --> T Size (Bytes)	Size of channel for an assembly. The memory size of each channel is 2 bytes that stores the value of %IWx or %QWx object, where x is the channel number.
Target to Scanner (Input)	
T --> O Size (Bytes)	T --> O Size (Bytes): Number of bytes to transfer (Number of channels of the assembly) The memory size of each channel is 2 bytes that stores the value of %IWx or %QWx object, where x is the channel number.
Connection Type	Connection type of the request: <ul style="list-style-type: none"> ● Multicast (the default) if the connection is between the scanner and multiple target devices ● Point-to-Point if the connection is between the scanner and a single target device Only editable for certain types of Transport Type .
Inhibit Time (ms)	Minimum period time between 2 data exchanges. Accessible if Trigger Type is Change of state . Inhibit Time maximum value is RPI and is limited to 254 ms.

Editing Generic Connections

To edit a generic connection, select the connection in the **Connections** tab and click **Edit Connection**:

Edit connection

Connection Path Settings

generate path automatically

Configuration Assembly
 Class ID 16# Instance ID 16# Attribute ID 16#

Consuming Assembly (O → T)
 Class ID 16# Instance ID 16# Attribute ID 16#

Producing Assembly (T → O)
 Class ID 16# Instance ID 16# Attribute ID 16#

user-defined path

Generic Parameters

Connection Name:

Connection Path:

Trigger Type: RPI (ms):

Transport Type: Timeout Multiplier:

Scanner to Target (Output)

O → T Size (Bytes):

Config#1 Size (Bytes):

Config#2 Size (Bytes):

Connection Type:

Fixed/Variable:

Transfer Format:

Inhibit Time (ms):

Target to Scanner (Input)

T → O Size (Bytes):

Connection Type:

Fixed/Variable:

Transfer Format:

Inhibit Time (ms):

Proceed as follows:

Parameter	Values	Description	
Connection Path Settings			
Generate path automatically	Yes/No	Enables you to configure the parameters of the assemblies.	
	Configuration assembly	True/False	Enables you to configure a configuration assembly (<i>see page 158</i>).
	Class ID	2 bytes (04h by default)	Class identifier ⁽¹⁾
	Instance ID	2 bytes (0 by default)	Instance identifier ⁽¹⁾
	Attribute ID	2 bytes (03h by default)	Attribute identifier ⁽¹⁾
	Consuming Assembly (O → T)		
	Class ID	2 bytes (04h by default)	Class identifier ⁽¹⁾
	Instance ID	2 bytes (0 by default)	Instance identifier ⁽¹⁾
	Attribute ID	2 bytes (03h by default)	Attribute identifier ⁽¹⁾
	Producing Assembly (T → O)		
	Class ID	2 bytes (04h by default)	Class identifier ⁽¹⁾
Instance ID	2 bytes (0 by default)	Instance identifier ⁽¹⁾	
Attribute ID	2 bytes (03h by default)	Attribute identifier ⁽¹⁾	
User-defined path	Yes/No	Disable the Generate path automatically area and enable the Connection Path field.	
Generic Parameters			
Connection Name	Text string	Type the name of the generic connection. The default value is generic connection .	
Connection Path	Array of bytes	Coded transcription of the physical link object if generate path automatically is selected. Editable if user-defined path is selected.	
⁽¹⁾ The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information (<i>see page 158</i>).			

Parameter	Values	Description
Trigger Type	<ul style="list-style-type: none"> ● Cyclic (default) ● Change of state ● Application 	<p>Select how the exchange of data is initiated:</p> <ul style="list-style-type: none"> ● Cyclic: Endpoints exchange data at regular, predetermined time intervals. ● Change of state: Endpoints only exchange data when the data changes. To keep the connection from timing out, the data is also exchanged at the background cyclic interval (see RPI below) if no changes occur. When Change of state is selected, the Inhibit Time (ms) fields of the scanner-to-target and target-to-scanner connection properties are enabled. ● Application. The exchange of data is triggered by the application.
Transport Type	<ul style="list-style-type: none"> ● Exclusive Owner (default) ● Redundant Owner 	<p>Exclusive Owner: This is a bidirectional connection to an output connection point (typically an Assembly Object), where the data of this assembly can only be controlled by one Scanner. There may be a connection to an input assembly; this data is being sent to the scanner. If the input data length is zero, then this direction becomes a Heartbeat connection.</p> <p>Redundant Owner. Allows multiple separate originator applications to each establish an independent, identical connection to the transport of a target device.</p>
RPI (ms)	In ms (10 ms by default)	<p>Requested Packet Interval. The time period between cyclic data transmissions requested by the scanner.</p> <p>The device RPI may be specified in the device documentation. Usually, however, this information is provided as part as the EDS file delivered with the device.</p>
Timeout Multiplier	4 (default) / 8 / 16 / 32 / 64 / 128 / 256 / 512	Scanner timeout is managed on a per-connection basic by multiplying the RPI and timeout multiplier values.
<p>⁽¹⁾ The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information (see page 158).</p>		

Parameter	Values	Description
Scanner to Target (Output)		
O --> T Size (Bytes)	0 to XX => device specific	Size of channel for an assembly. The memory size of each channel is 2 bytes, which store the value of %IW _x or %QW _x objects, where x is the channel number.
Config#1 Size (Bytes)	0 to XX => device specific	Accessible if connection path contains a configuration assembly. Number of parameters (1 byte) to transfer. The configuration values are sent to the device at the scanner start.
Config#2 Size (Bytes)	0 to XX => device specific	
Connection Type	Point to Point	Connection type of the request
Fixed/Variable	Fixed	The request length is fixed.
Transfer format	<ul style="list-style-type: none"> ● 32 bit Run-idle (by default) ● pure Data ● Heartbeat 	Transfer format of the request. For more information, refer to ODVA website .
Inhibit Time	0 ms	Minimum period time between 2 data exchanges.
Target to Scanner (Input)		
T --> O Size (Bytes)	0 to XX => device specific	Size of channel of an assembly. The memory size of each channel is 2 bytes that stores the value of %IW _x or %QW _x object, where x is the channel number.
Connection Type	<ul style="list-style-type: none"> ● Multicast (default) ● Point to Point 	Connection type of the request
Fixed/Variable	Fixed	The request length is fixed.
Transfer format	<ul style="list-style-type: none"> ● 32 Bit Run Idle ● pure Data (by default) ● Heartbeat ● Idle with zero length 	Transfer format of the request. For more information, refer to ODVA website .
Inhibit Time (ms)	In multiples of 2 ms (2 ms by default)	Minimum period time between 2 data exchanges. Accessible if Trigger Type is Change of state . Inhibit Time maximum value is RPI and is limited to 254 ms.
(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find Assembly Information (see page 158).		

How to Find Assembly Information

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

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

1. Class ID

The "Assembly object" Class ID is contained in the device documentation and is normally equal to 04h.

2. Instance ID

Select the assembly instance, depending on the application and on the type of device. The selection of the assembly instance will induce a dedicated state machine in the device:

- **Configuration assembly:** Supported by few devices; verify in the device documentation which assembly instance is supported.
- **Consuming assembly:** sometimes referred to as "device output" in the device documentation (from the device point of view).
- **Producing assembly:** sometimes referred to as "device input" in the device documentation (from the device point of view).

3. Attribute ID

Search for the attribute to read. This corresponds to the data buffer exchanged during the connection.

The attribute property must have write access for the producing assembly and read access for the consuming assembly.

The attribute ID is the same for the two assemblies. Its value is contained in the device documentation and is normally equal to 03h. It matches an attribute whose access is *Get/Set*. The name is often "data", and the type of data "Array of byte".

Device Replacement with User Parameters

Overview

You can configure **User Parameters** that are sent to the device to facilitate device replacement just before the scanner connection is started after:

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

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

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

For maintenance details, refer to Apply the Correct Device Configuration.

User Parameters

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

Line	Name	Class...	Instance...	Attribute...	Value	Bitlength	Abort if error	Jump to line if error	Next line	Comment
1	Profile	8B	1	2	2	16	<input type="checkbox"/>	<input type="checkbox"/>	0	
2	Ref. 1 Channel	8B	1	E	169	16	<input type="checkbox"/>	<input type="checkbox"/>	0	
3	Cmd Channel 1	8B	1	18	30	16	<input type="checkbox"/>	<input type="checkbox"/>	0	
4	Cmd Switching	8B	1	16	98	16	<input type="checkbox"/>	<input type="checkbox"/>	0	
5	Ref. 2 Switching	8B	1	C	222	16	<input type="checkbox"/>	<input type="checkbox"/>	0	
6	OMA3	AE	1	4	3204	16	<input type="checkbox"/>	<input type="checkbox"/>	0	

Column	Description
Line	Line number. Indicates the order of the parameters loaded to the device.
Name	Name of the parameter.
Class	Class ID ⁽¹⁾ of the class corresponding to the object.
Instance	Instance ID ⁽¹⁾ of the instance corresponding to the object.
Attribute	Attribute ID ⁽¹⁾ of the attribute corresponding to the object.
Value	Value of the parameter. Double-click the value to modify it. If applicable, a list opens containing possible values.
⁽¹⁾ The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find User Parameter Information (see page 162).	

Column	Description
Bitlength	Number of bits of the parameter. Automatically changed depending of the parameter datatype selected.
Abort if error	If selected, when an error is detected, the transmission of the parameters is aborted.
Jump to line if error	If selected, when an error is detected, the program resumes with the line specified in the Next line column. A block can thus be skipped during the initialization, or a return can be defined. NOTE: A return can lead to an endless loop if it is never possible to write a certain parameter.
Next line	Double-click to enter the line to jump to (if Jump to line if error is selected).
Comment	Double-click to enter a comment.
(1) The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find User Parameter Information (see page 162).	

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

Creating or Configuring User Parameters

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

Fields	Description
Name	Name of the parameter.
Class	Class ID ⁽¹⁾ of the class corresponding to the type of object.
Instance	Instance ID ⁽¹⁾ of the instance corresponding to an implementation of a class.
Attribute	Attribute ID ⁽¹⁾ of the attribute corresponding to a characteristic of an instance.
Datatype	List containing the possible data type.
Bitlength	Number of bits of the parameter. Automatically changed depending on the selected Datatype .
Value	Value of the parameter.

⁽¹⁾ The Class ID, Instance ID, and Attribute ID can be found in the device documentation. Refer to How To Find User Parameter Information ([see page 162](#)).

How To Find User Parameter Information

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

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

A user parameter always has the following numeric properties:

- Class, or Class ID, usually expressed as an hexadecimal value
- Instance, or Instance ID, usually expressed as an hexadecimal value
- Attribute, or Attribute ID, usually expressed as an hexadecimal value

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

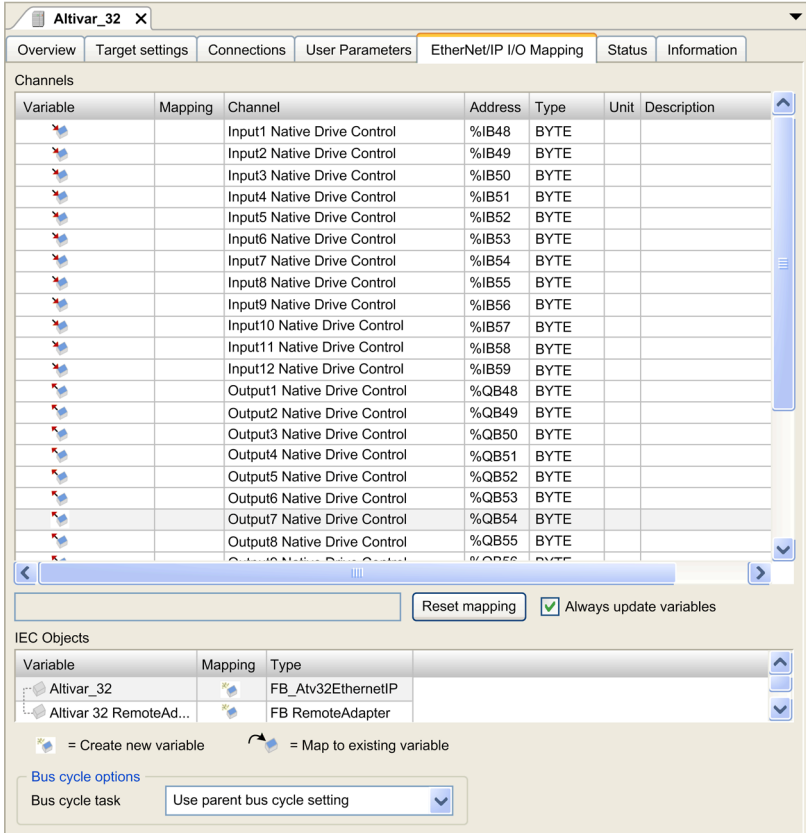
EtherNet/IP I/O Mapping

Overview

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

Configure an EtherNet/IP Target Device I/O Mapping

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

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

Step	Action
3	Double-click in a cell of the Variable column to open a text field. Enter the name of a variable or click the browse button [...] and chose a variable with the Input Assistant .
4	IEC Objects . This part of the tab lists IEC objects of the target device that can be accessed by the application (for example, in order to restart a bus or to poll information).
5	The Bus cycle task parameter defines the task responsible for refreshing the I/O images (%IB and %QB). These I/O images correspond to the EtherNet/IP request sent to the EtherNet/IP target device and the health bits. Select the Bus cycle task in the list: <ul style="list-style-type: none"> ● Use parent bus cycle setting (by default). Use the task specified in the PLC Settings <i>(see page 88)</i> tab of the controller. ● MAST. Use the MAST task <i>(see page 49)</i>. ● Motion. Use the Motion task <i>(see page 42)</i>.

Chapter 11

Ethernet Configuration

Introduction

This chapter describes how to configure the Ethernet network interface of the Modicon LMC078 Motion Controller.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	Ethernet Services	166
11.2	Firewall Configuration	184

Section 11.1

Ethernet Services

What Is in This Section?

This section contains the following topics:

Topic	Page
Presentation	167
IP Address Configuration	169
Modbus TCP Client/Server	174
FTP Server	176
FTP Client	178
LMC078 Motion Controller as an IOScanner Slave Device on Modbus TCP	179

Presentation

Ethernet Services

The controller supports the following services:

- Modbus TCP Server (*see page 174*)
- Modbus TCP Client (*see page 174*)
- Web visualization (*see page 168*)
- FTP Server (*see page 176*)
- FTP Client (*see page 178*)
- Controller as an IOScanner Slave Device on Modbus TCP (*see page 179*)
- IEC VAR ACCESS (*see page 168*)

Ethernet Protocol

The controller supports the following protocols:

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

Connections

This table presents the maximum number of connections:

Connection Type	Maximum Number of Connections
Modbus Server	8
Modbus Client	2
Ethernet/IP Target	64
FTP Server	4
Web visualization	10

Each server based on TCP manages its own set of connections.

When a client tries to open a connection that exceeds the poll size, the controller closes the oldest connection.

If all connections are busy (exchange in progress) when a client tries to open a new one, the new connection is denied.

All server connections stay open as long as the controller stays in operational states (RUN, STOP, HALT).

All server connections are closed when leaving or entering operational states (RUN, STOP, HALT), except in case of power outage (because the controller does not have time to close the connections).

Services Available

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

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

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

Web Visualization

The Web visualization function is described in the SoMachine online help, chapter *Programming with SoMachine / Visualization*.

IP Address Configuration

Introduction

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

- address assignment by DHCP server
- address assignment by BOOTP server
- fixed IP address

The IP address can be changed dynamically:

- via the Controller Selection (*see SoMachine, Programming Guide*) tab in SoMachine.
- via the **changeIPAddress** function block (*see page 251*).

NOTE: If the attempted addressing method is unsuccessful, the controller will start using a default IP address (*see page 172*).

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

WARNING

UNINTENDED EQUIPMENT OPERATION

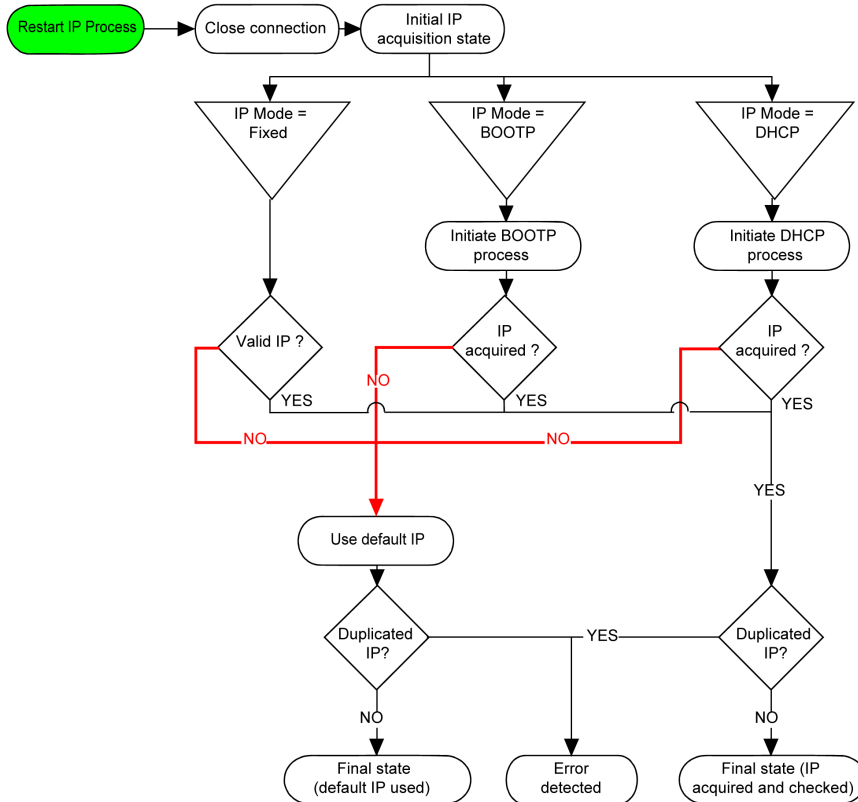
- Verify that there is only one master controller configured on the network or remote link.
- Verify that all devices have unique addresses.
- Obtain your IP address from your system administrator.
- Confirm that the IP address of the device is unique before placing the system into service.
- Do not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

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

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

Address Management

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



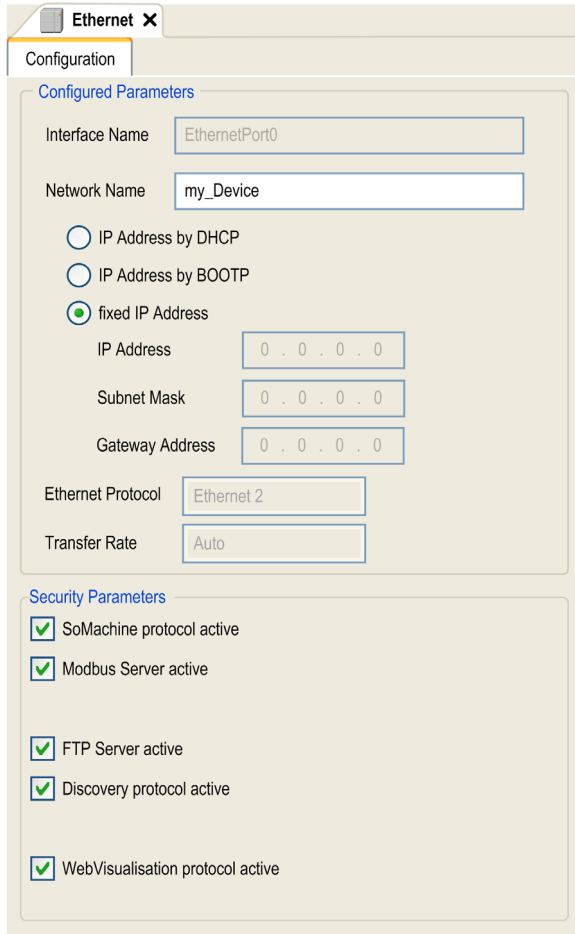
NOTE: If a device programmed to use the DHCP or BOOTP addressing methods is unable to contact its respective server, the controller uses the default IP address. It will, however, constantly repeat its request.

The IP process restarts in the following cases:

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

Ethernet Configuration

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



Ethernet x

Configuration

Configured Parameters

Interface Name: EthernetPort0

Network Name: my_Device

IP Address by DHCP
 IP Address by BOOTP
 fixed IP Address

IP Address: 0 . 0 . 0 . 0

Subnet Mask: 0 . 0 . 0 . 0

Gateway Address: 0 . 0 . 0 . 0

Ethernet Protocol: Ethernet 2

Transfer Rate: Auto

Security Parameters

SoMachine protocol active
 Modbus Server active
 FTP Server active
 Discovery protocol active
 WebVisualisation protocol active

The configured parameters are explained as below:

Configured Parameters	Description
Interface Name	Name of the network link.
Network Name	Used as device name to retrieve IP address through DHCP, maximum 16 characters.
IP Address by DHCP	IP address is obtained via DHCP.
IP Address by BOOTP	IP address is obtained via BOOTP.

Configured Parameters	Description
Fixed IP Address	To configure a fixed IP address, use the Controller Selection (<i>see page 87</i>) tab of the Controller Device Editor .
Ethernet Protocol	Protocol type used: Ethernet2
Transfer Rate	Transfer rate and direction on the bus are automatically configured.

Default IP Address

The IP address by default is 190.201.100.100.

The default subnet mask is 255.255.255.0.

Address Classes

The IP address is linked:

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

An IP address is always coded using 4 bytes.

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

The different IP address classes are defined in this table:

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

Subnet Mask

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

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

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

Example of a subnet address:

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

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

Gateway Address

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

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

Security Parameters

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

Nodename

The **Nodename** is used to identify a device during a network scan. Each device in your network must have a unique **Nodename**.

The **Nodename** of an LMC078 motion controller is the controller name as it appears in the **Devices tree**.

Modbus TCP Client/Server

Introduction

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

The Modicon LMC078 Motion Controller implements both client and server services so that it can initiate communications to other controllers and I/O devices, and to respond to requests from other controllers, SCADA, HMIs, and other devices.

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

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

Modbus TCP Client

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

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

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

Modbus TCP Server

The Modbus server supports the Modbus requests:

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

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

NOTE: The embedded Modbus server only ensures time-consistency for a single word (2 bytes). If your application requires time-consistency for more than 1 word, add and configure (*see page 179*) a **Modbus TCP Slave Device** so that the contents of the %IW and %QW buffers are time-consistent in the associated IEC task (MAST by default).

Diagnostic Request

This table contains the data selection code list:

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

FTP Server

Introduction

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

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

WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

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

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

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

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

FTP Access

Access to the FTP server is controlled by User Rights when they are enabled in the controller. For more information, refer to **Users and Groups** Tab Description (*see page 74*).

If User Rights are not activated in the controller, you are prompted for a user name and password. The default user name is USER and the default password is also USER.

NOTE: It is preferable to use the User Rights to help protect your controller as a whole.

For reasons of security for your installation, you must immediately change the default password upon first login if User Rights are not enabled in the controller. The password can be changed with the function FC_UserChangePassword (*see Modicon LMC078 Motion Controller, System Functions and Variables, PLCSystem Library Guide*).

WARNING

UNAUTHORIZED DATA ACCESS

- Immediately change the default password to a new, secure password.
- Do not distribute the password to unauthorized or otherwise unqualified personnel.
- Disable the FTP to prevent any unwanted or unauthorized access to data in your application.

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

NOTE: A secure password is one that has not been shared or distributed to any unauthorized personnel and does not contain any personal or otherwise obvious information. Further, a mix of upper and lower case letters and numbers offer greater security. You should choose a password length of at least 7 characters.

NOTE: The only way to gain access to a controller that has user access-rights enabled and for which you do not have the password(s) is by performing an Update Firmware operation. In addition, you may clear the User Rights in the controller by running a script (for more information, refer to SoMachine Programming Guide). This effectively removes the existing application from the controller memory, but restores the ability to access the controller.

If you have not enabled User Rights and if you have lost or forgotten the password, you will need to connect directly to the controller with SoMachine and perform a reset origin to reestablish the default password. After doing so, set up a new, secure password.

Files Access

See File Organization (*see page 33*).

FTP Client

Introduction

The FtpRemoteFileHandling library provides the following FTP client functionalities for remote file handling:

- Reading files
- Writing files
- Deleting files
- Listing content of remote directories
- Adding directories
- Removing directories

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

WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

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

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

For further information, refer to FtpRemoteFileHandling Library Guide.

LMC078 Motion Controller as an IScanner Slave Device on Modbus TCP

Overview

This section describes the configuration of the LMC078 Motion Controller as a **Modbus TCP Slave Device**.

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

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

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

The TCP slave device can define a privileged Modbus client application, whose connection is not forcefully closed (embedded Modbus connections may be closed when more than 8 connections are needed).

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

Adding a Modbus TCP Slave Device

To configure your LMC078 Motion Controller as a Modbus TCP slave device:

Step	Action
1	Select the Ethernet Network node and click the Plus button. Result: The Add Device dialog window is displayed.
2	Choose ModbusTCP Slave Device .
3	Click Add Device
4	Click Close .

Modbus TCP Configuration

To configure the **Modbus TCP Slave Device**, double-click **Ethernet Network** → **ModbusTCP Slave Device** in the **Devices tree**.

This dialog box appears:

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

Modbus TCP Slave Device I/O Mapping Tab

The I/Os are mapped to Modbus registers from the master perspective as follows:

- %IW's are mapped from register 0 to n-1 and are R/W (n = Holding register quantity, each %IW register is 2 bytes).
- %QW's are mapped from register n to n+m -1 and are read only (m = Input registers quantity, each %QW register is 2 bytes).

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

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

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

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

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

Modbus TCP

Modbus TCP Slave Device I/O Mapping

Information

Channels

Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
		Inputs	%IW0	ARRAY [0...9] OF...			Modbus Holding...
Application.POU.tata		Inputs[0]	%IW0	WORD			
iwModbusTCT_Sla...		Inputs[1]	%IW1	WORD			
iwModbusTCT_Sla...		Inputs[2]	%IW2	WORD			
iwModbusTCT_Sla...		Inputs[3]	%IW3	WORD			
iwModbusTCT_Sla...		Inputs[4]	%IW4	WORD			
iwModbusTCT_Sla...		Inputs[5]	%IW5	WORD			
iwModbusTCT_Sla...		Inputs[6]	%IW6	WORD			
iwModbusTCT_Sla...		Inputs[7]	%IW7	WORD			
iwModbusTCT_Sla...		Inputs[8]	%IW8	WORD			
iwModbusTCT_Sla...		Inputs[9]	%IW9	WORD			
		Outputs	%QW0	ARRAY [0...9] OF...			Modbus Input Re...
qwModbusTCP_SI...		Outputs[0]	%QW0	WORD			
qwModbusTCP_SI...		Outputs[1]	%QW1	WORD			
qwModbusTCP_SI...		Outputs[2]	%QW2	WORD			
qwModbusTCP_SI...		Outputs[3]	%QW3	WORD			
qwModbusTCP_SI...		Outputs[4]	%QW4	WORD			
qwModbusTCP_SI...		Outputs[5]	%QW5	WORD			
qwModbusTCP_SI...		Outputs[6]	%QW6	WORD			
qwModbusTCP_SI...		Outputs[7]	%QW7	WORD			
qwModbusTCP_SI...		Outputs[8]	%QW8	WORD			
qwModbusTCP_SI...		Outputs[9]	%QW9	WORD			

Always update variables

IEC Objects

Variable	Mapping	Type
Modbus TCP_Slave_De		IoDrvModbusTCPSlave

= Create new variable = Map to existing variable

Bus cycle options

Bus cycle task Use parent bus cycle setting

Channel	Type	Description
Input	IW0	WORD

	IWx	WORD
Output	QW0	WORD

	QWy	WORD

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

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

NOTE: The **Modbus TCP Slave Device** refreshes the %IW and %QW registers as a single time-consistent unit, synchronized with the IEC tasks (MAST task by default). By contrast, the embedded Modbus TCP server only ensures time-consistency for 1 word (2 bytes). If your application requires time-consistency for more than 1 word (2 bytes), use the **Modbus TCP Slave Device**.

Bus Cycle Options

Select the **Bus cycle task** to use:

- **Use parent bus cycle setting** (the default),
- **MAST**
- **An existing task of the project**

NOTE: There is a corresponding **Bus cycle task** parameter in the I/O mapping editor of the device that contains the **Modbus TCP Slave Device**. This parameter defines the task responsible for refreshing the %IW and %QW registers.

Section 11.2

Firewall Configuration

Introduction

This section describes how to configure the firewall of the Modicon LMC078 Motion Controller.

What Is in This Section?

This section contains the following topics:

Topic	Page
Introduction	185
Firewall Behavior	187
Firewall Script Commands	188
Script Files	192

Introduction

Firewall Presentation

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

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

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

WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

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

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

Firewall Configuration

There are two ways to manage the controller firewall configuration:

- Static configuration,
- Application settings.

Script file is used in the static configuration.

Static Configuration

The static configuration is loaded at the controller boot.

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

Application Settings

Refer to Ethernet Configuration (*see page 171*).

Firewall Behavior

Introduction

The firewall configuration depends on the action done on the controller and the initial configuration state. There are 4 possible initial states:

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

No Default Script File

If...	Then ...
Boot of the controller	Firewall is not configured. No protection is activated.
Download application	Firewall is configured according to the application settings.

Default Script File Present

If...	Then ...
Boot of the controller	Firewall is configured according to the default script file.
Download application	The whole configuration of the application is ignored. Firewall is configured according to the default script file.

Incorrect Default Script File Present

If...	Then ...
Boot of the controller	Firewall is not configured. No protection is activated
Download application	Firewall is configured according to the application settings.

Application Settings with No Default Script File

If...	Then ...
Boot of the controller	Firewall is configured according to the application settings.
Download application	The whole configuration of the previous application is deleted. Firewall is configured according to the new application settings.

Firewall Script Commands

Overview

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

Script File Syntax

The syntax of script files is described in Script Syntax Guidelines ([see page 192](#)).

General Firewall Commands

The following commands are available to manage the Ethernet firewall of the LMC078 Motion Controller:

Command	Description
<code>FireWall Enable</code>	Blocks the frames from the Ethernet interfaces. If no specific IP address is authorized, it is not possible to communicate on the Ethernet interfaces. NOTE: By default, when the firewall is enabled, the frames are rejected.
<code>FireWall Disable</code>	IP addresses are allowed access to the controller on the Ethernet interfaces.
<code>FireWall Eth1 Default Allow</code>	Frames are accepted by the controller.
<code>FireWall Eth1 Default Reject</code>	Frames are rejected by the controller. NOTE: By default, if this line is not present, it corresponds to the command <code>FireWall Eth1 Default Reject</code> .

Specific Firewall Commands

The following commands are available to configure firewall rules for specific ports and addresses:

Command	Range	Description
<code>Firewall Eth1 Allow IP*</code>	<code>* = 0...255</code>	Frames from the specified IP address are allowed on all port numbers and port types.
<code>Firewall Eth1 Reject IP*</code>	<code>* = 0...255</code>	Frames from the specified IP address are rejected on all port numbers and port types.
<code>Firewall Eth1 Allow IPs* to*</code>	<code>* = 0...255</code>	Frames from the IP addresses in the specified range are allowed for all port numbers and port types.
<code>Firewall Eth1 Reject IPs* to*</code>	<code>* = 0...255</code>	Frames from the IP addresses in the specified range are rejected for all port numbers and port types.

Command	Range	Description
Firewall Eth1 Allow port_type port Y	Y = (destination port numbers <i>(see page 191)</i>)	Frames with the specified destination port number are allowed.
Firewall Eth1 Reject port_type port Y	Y = (destination port numbers <i>(see page 191)</i>)	Frames with the specified destination port number are allowed.
Firewall Eth1 Allow port_type ports Y1 to Y2	Y = (destination port numbers <i>(see page 191)</i>)	Frames with a destination port number in the specified range are allowed.
Firewall Eth1 Reject port_type ports Y1 to Y2	Y = (destination port numbers <i>(see page 191)</i>)	Frames with a destination port number in the specified range are rejected.
Firewall Eth1 Allow IP on port_type port Y	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from the specified IP address and with the specified destination port number are allowed.
Firewall Eth1 Reject IP on port_type port Y	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from the specified IP address and with the specified destination port number are rejected.
Firewall Eth1 Allow IP on port_type ports Y1 to Y2	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from the specified IP address and with a destination port number in the specified range are allowed.
Firewall Eth1 Reject IP on port_type ports Y1 to Y2	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from the specified IP address and with a destination port number in the specified range are rejected.
Firewall Eth1 Allow IPs •1.1.1.1 to •2.2.2.2 on port_type port Y	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from an IP address in the specified range and with the specified destination port number are rejected.
Firewall Eth1 Reject IPs •1.1.1.1 to •2.2.2.2 on port_type port Y	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from an IP address in the specified range and with the specified destination port number are rejected.
Firewall Eth1 Allow IPs •1.1.1.1 to •2.2.2.2 on port_type ports Y1 to Y2	• = 0...255 Y = (destination port numbers <i>(see page 191)</i>)	Frames from an IP address in the specified range and with a destination port number in the specified range are allowed.

Command	Range	Description
Firewall Eth1 Reject IPs •1.1.1.1 to •2.2.2.2 on port_type ports Y1 to Y2	• = 0...255 Y = (destination port numbers (<i>see page 191</i>))	Frames from an IP address in the specified range and with a destination port number in the specified range are rejected.
Firewall Eth1 Allow MAC ••:••:••:••:••:••	• = 0...F	Frames from the specified MAC address ••:••:••:~••:~•• are allowed.
Firewall Eth1 Reject MAC ••:~••:~••:~••:~••	• = 0...F	Frames with the specified MAC address ••:~••:~••:~••:~•• are rejected.

Script Example

```

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

```

Used Ports

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

Script Files

Overview

The following describes how to write script files to configure the Ethernet firewall (*see page 188*).

Script Syntax Guidelines

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

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

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

The syntax is not case-sensitive.

If the syntax is not respected in the script file, the script file is not executed. This means, for example, that the firewall configuration remains in the previous state.

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

Chapter 12

CANopen Configuration

Introduction

This chapter describes how to configure the CAN interface offered within the Controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
CANopen Interface Configuration	194
CANopen Master Configuration	195
CANopen Slave Configuration	197

CANopen Interface Configuration

Adding the CAN Bus Node

To add the **CAN_Layer** node to your controller, select **CANbus** in the **Hardware Catalog**, drag it to the **Devices tree** and drop it on your controller node.

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

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

CAN Bus Configuration

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

Step	Action
1	In the Devices tree , double-click CAN_Layer .
2	Configure the baud rate (by default: 250000 bits/s).

Adding a CANopen Manager

The controller supports the following CANopen managers:

- **CANopen_Manager** for the CAN port configured as CANopen master
- **CAN_Local_Device** for the CAN port configured as CANopen slave

To add a CANopen manager to your controller, select in the **Hardware Catalog**:

- For a CANopen master: **CANopen_Manager**
- For a CANopen slave: **CAN_Local_Device**

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

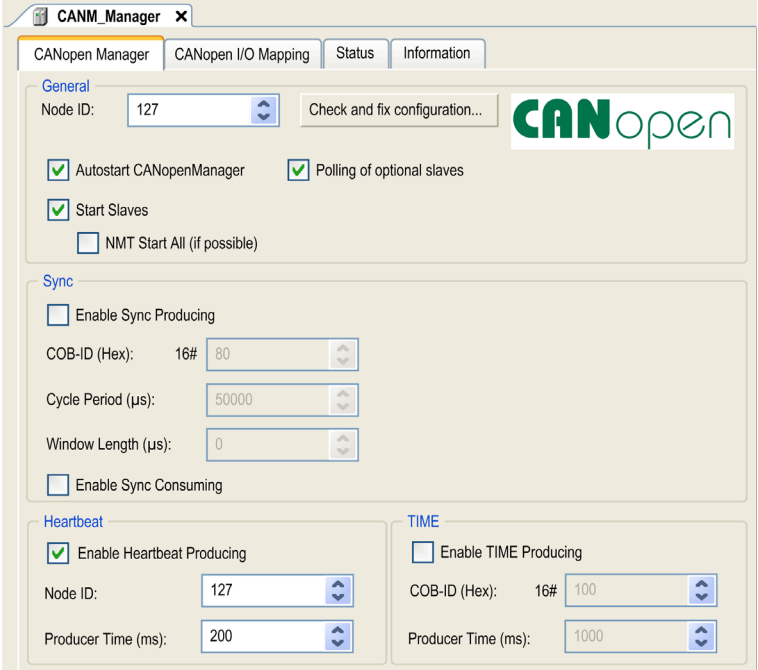
For more information on adding a device to your project, refer to:

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

CANopen Master Configuration

CANopen Manager Configuration

To configure the **CANopen_Manager**, proceed as follows:

Step	Action
1	<p>Double-click CANopen_Manager in the Devices tree. Result: The CANopen Manager configuration window appears:</p> 
2	<p>For more information about CANopen manager configuration, refer to the SoMachine online help, chapter <i>Programming with SoMachine / Device Editors / CANopen Manager</i>.</p>

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 LMC078 Motion Controller CANopen master has the following operating limits:

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

WARNING

UNINTENDED EQUIPMENT OPERATION

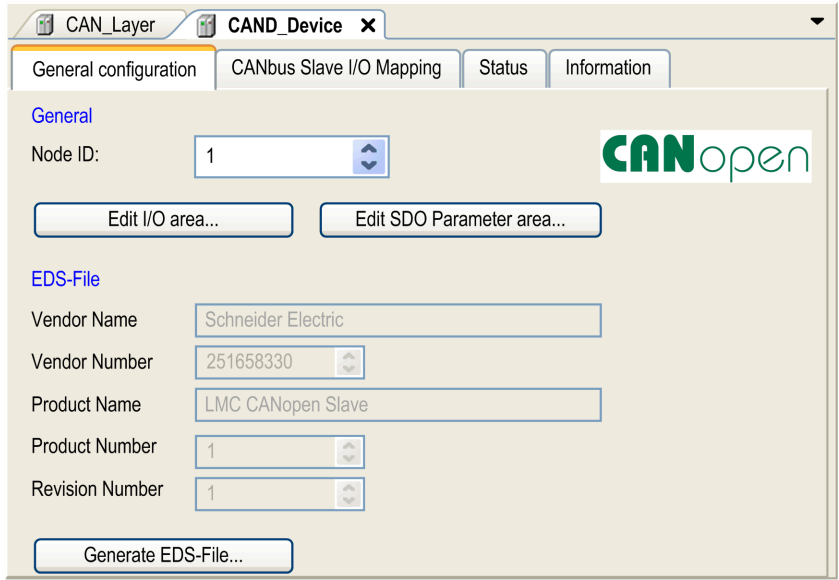
- Do not connect more than 63 CANopen slave devices to the controller.
- Program your application to use 252 or fewer Transmit PDO (TPDO).
- Program your application to use 252 or fewer Receive PDO (RPDO).

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

CANopen Slave Configuration

CANopen Slave Configuration

To configure the controller as a CANopen slave, proceed as follows:

Step	Action
1	Select the controller node in the Devices tree , and either click the green plus button of the node, or right-click the node and select Add Device from the context menu. Result: The Add Device dialog box opens.
2	In the Add Device dialog box, select CANbus and click the Add Device button. Result: The device is added to the controller.
3	Click the Close button in the Add Device dialog box.
4	Select the CAN_Layer node in the Devices tree , and either click the green plus button of the node, or right-click the node and select Add Device from the context menu.
5	In the Add Device dialog box, select CAN Local_Device and click the Add Device button. Result: The device is added to the CAN_Layer node.
6	Click the Close button in the Add Device dialog box.
7	Double-click CAND_Device in the Devices tree . Result: The CAND_Device configuration window appears: 
8	The configuration of the CANopen manager is described in the SoMachine online help, chapter <i>Programming with SoMachine / Device Editors / CANbus Slave Device</i> .

Chapter 13

Sercos Configuration

Introduction

This chapter describes how to configure the Sercos interface of the Modicon LMC078 Motion Controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Overview of the Sercos Standard	200
Sercos Interface Configuration	203
Sercos Devices	208
Device Addressing Editor	209
Lexium LXM32S Drive Configuration	213
TM5NS31 Sercos Interface Module	216
Sercos Error Codes	217

Overview of the Sercos Standard

Introduction

The Sercos interface is a standardized interface (IEC 6149) for real-time communication between controllers, drives, and I/O devices.

It describes the internationally standardized digital interface for communication between a control unit and associated servo drives networked together to form a motion control system. It defines standardization of operating data, parameters, and scaling for machines with multiple drives that can be operated in torque, velocity, or position interface operation modes.

The main features of the Sercos interface are:

- Ring topology (redundancy)
- Master / slave system
- Baud rate 100 MBaud
- Minimum synchronization time of 1 ms (8 axes), 2 ms (16 axes), or 4 ms (24 axes)
- Synchronization (jitter < 1 μ s)

Data Exchange

Communication with Sercos interface is divided into two types:

- Cyclical communication with telegrams:
The cyclical communication is used for exchanging real-time data (for example, position) and is executed once in every communication cycle (`CycleTime`). Certain specified data are transferred from the controller to all drives and from all drives to the controller in every cycle.
- Non-cyclical communication with function blocks of the LMC078_Sercos3 library
(*see page 271*).

Non-cyclical communication is used to exchange data such as parameters for configuring communication, the drive parameters, status, and so on, where time is not a critical factor. The controller controls non-cyclical communication. All of the parameters in the system can be contacted using this channel, even parameters that are configured cyclically.

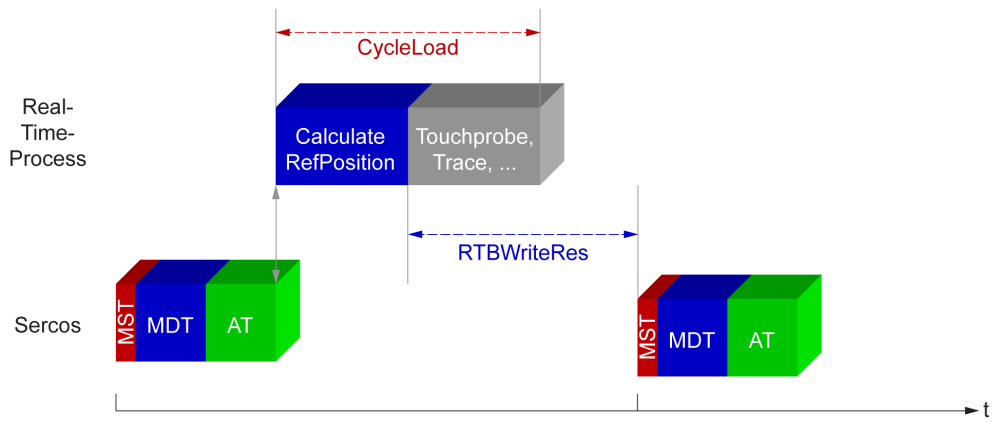
NOTE: The two types of communication can be used simultaneously.

Cyclical Data Exchange

The exchange of information between the motion controller (Sercos master) and the servo drives (slaves), is accomplished via a message structure known as a telegram. There are three telegrams defined by IEC 61491:

- MST (Master Synchronization Telegram): An MST telegram is broadcast by the master at the beginning of each transmission cycle to synchronize the timing of the cycle.
- MDT (Master Data Telegram): An MDT telegram is sent by the master once during each transmission cycle to transmit data (command values) to the servo drives (slaves).
- AT (Acknowledge Telegram): AT telegrams are sent by the slaves to the master (feedback values).

This illustration presents the 3 types of telegrams:



This topic discusses only the MDT and AT telegram types. A general telegram structure is shown below:

Telegram Delimiter	Address Field	Configurable data field	Frame Check Sequence	Telegram Delimiter
--------------------	---------------	-------------------------	----------------------	--------------------

The administrative segment of the telegram, which includes the telegram delimiter (start), address field, frame check sequence, and telegram delimiter (end), are required for the transmission of all telegrams.

Within the telegram, real-time data (operation data) is transmitted in the configurable data field during each communication cycle. The specification of this data is provided by an identification number (IDN).

Sercos enables the processing cycle of the controller to be synchronized with data exchange and the control cycle in the drive. Therefore, there is no interference between these individual cycles, and the control loops have a minimum, constant dead time. Moreover, the new reference value goes into effect in all drives at the same time and all bus slaves record the measured values that they forward to the controller as actual values at the same time.

IDN Description

IEC 61491 assigns identification numbers (IDNs) to all the operation data in a Sercos drive. Operation data includes parameters, interface procedure commands, and command and feedback values.

There are two categories of IDNs available:

- Standard IDNs (S): They are defined by the Sercos standard IEC 61491. Standard IDNs, if supported by a Sercos drive, behave the same, irrespective of the drive manufacturer.
- Proprietary IDNs (P): They are reserved for product-specific data that can be defined by the manufacturers of control units and servo drives.

NOTE: For complete, detailed information on Sercos IDNs that are implemented in the LXM32S drives, refer to the Lexium LXM32S Product Manual.

The IDNs are normally 16-bit or 32-bit binary parameters of a Sercos drive. IDNs are identified in the following way:

- S-0-0047.0.0 standard IDN: Position command value
- P-0-3017.0.12 manufacturer-defined IDN: Current limitation

The -0- refers to parameter sets. Many Sercos drives (including the Lexium) do not support parameter sets.

Sercos Interface Configuration

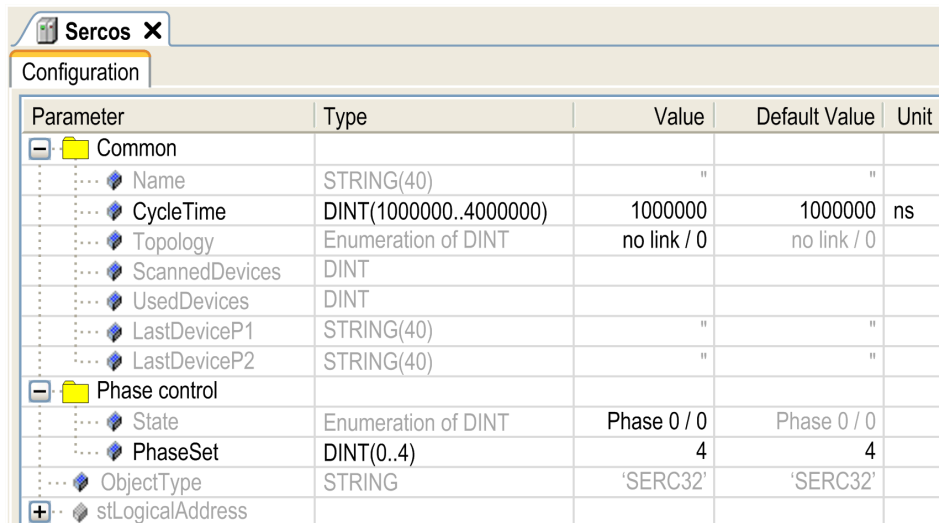
Introduction

The Sercos configuration window allows you to configure and view the Sercos interface parameters.

Sercos Interface Configuration

To access the Sercos configuration window, double-click the **SERCOSIII** node in the **Devices tree**.

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



Parameter	Type	Value	Default Value	Unit
[-] Common				
... Name	STRING(40)	"	"	
... CycleTime	DINT(1000000..4000000)	1000000	1000000	ns
... Topology	Enumeration of DINT	no link / 0	no link / 0	
... ScannedDevices	DINT			
... UsedDevices	DINT			
... LastDeviceP1	STRING(40)	"	"	
... LastDeviceP2	STRING(40)	"	"	
[-] Phase control				
... State	Enumeration of DINT	Phase 0 / 0	Phase 0 / 0	
... PhaseSet	DINT(0..4)	4	4	
... ObjectType	STRING	'SERC32'	'SERC32'	
[+] ... stLogicalAddress				

This table describes the parameters of the Sercos interface:

Parameter	Access	Param. type	Data type	Value	Default value	Description
Common						
Name	R/W(*)	EF	STRING	"	"	Symbolic name of the configuration object.
CycleTime	R/W	ER	DINT	1000000 2000000 4000000	1000000	Defines the Sercos bus cycle time in ns. The CycleTime can be set to either 1 ms, 2 ms or 4 ms. If the CycleTime is changed, the controller must be reset.

Parameter	Access	Param. type	Data type	Value	Default value	Description
Topology	R	AF	DINT Enum	no link / 0 line P1 / 1 line P2 / 2 double line / 3 ring / 4 defect ring / 8	no link / 0	Describes the topology of the Sercos system: <ul style="list-style-type: none"> ● 0 = No Sercos device connected. ● 1 = All Sercos devices are connected to port 1. ● 2 = All Sercos devices are connected to port 2. ● 3 = Sercos devices are connected to port 1 and port 2. ● 4 = The connection from port 1 to port 2 is closed (the ring is closed). ● 8 = Invalid ring topology, topology switchover not yet completed.
ScannedDevices	R	AD	DINT	-	-	Number of physically scanned Sercos devices.
UsedDevices	R	AD	DINT	-	-	Number of configured and physically assigned Sercos devices.
LastDeviceP1	R	AD	STRING	"	"	Name of the last physical Sercos device on port 1.
LastDeviceP2	R	AD	STRING	"	"	Name of the last physical Sercos device on port 2.

Parameter	Access	Param. type	Data type	Value	Default value	Description
Phase control						
State	R	AD	DINT Enum	Phase 0 / 0 Phase 1 / 1 Phase 2 / 2 Phase 3 / 3 Phase 4 / 4 Firmware download / 5 Phase 6 / 6 Bus scan / 7 Reinit Sercos / 8 Init / 10 Error / 11 Continuous light / 12 Zero bit stream / 13	Phase 0 / 0	Displays the Sercos system state: <ul style="list-style-type: none"> ● 0 = Configuration and startup. ● 1 = Normal operation, Sercos phase 1. ● 2 = Normal operation, Sercos phase 2. ● 3 = Normal operation, Sercos phase 3. ● 4 = Normal operation, Sercos phase 4. ● 5 = Firmware download in progress. ● 6 = Reserved ● 7 = Bus scan in progress. ● 8 = Sercos reinitialization in progress. ● 10 = Occurs only briefly during the booting of the system. ● 11 = A Sercos error has been detected in the operating phase. ● 12 = Not applicable (for optical connection). ● 13 = Not applicable (for optical connection).

Parameter	Access	Param. type	Data type	Value	Default value	Description
PhaseSet	R/W	EF	DINT	0...4	4	<p>You can use this parameter to preset the required communication phase of the Sercos bus: ⁽¹⁾</p> <ul style="list-style-type: none"> ● 0...3 = The tasks of the program are started simultaneously with the Sercos run-up. ● 4 = The program tasks start is delayed until: <ul style="list-style-type: none"> ○ The Sercos run-up has reached phase 4 (State = 4). ○ The Sercos run-up is canceled by a start-up error (State = 11).
—						
ObjectType	R	AD	STRING	SERC32	SERC32	Object type.
stLogicalAddress	R	AD	ST_LogicalAddress	-	-	<p>Logical address of the Sercos parameters.</p> <p>LogicalAddress = STRUCT (udiType, udiInstance, udiParameterId)</p>

(*) For more information on the parameter access rights, refer to Parameter Types ([see page 27](#)).

(1) Once the 'PhaseSet' parameter indicates the active exchange of real-time date, a reinitialization of the axes is required for the synchronization of position.

NOTICE

POSITION LOSS DUE TO THE SERCOS BUS PHASE CHANGE

Programmatically ensure the reinitialization or homing of the motion system when first arriving at Sercos phase 4.

Failure to follow these instructions can result in equipment damage.

Sercos Operating Phases

This table describes the Sercos operating phases (phases 0..4):

Operating phase	Description
Phase 0	Verify whether the number of connected devices remains constant and which topology is used. Then, a switchover to phase 1 is effected with the determined topology.
Phase 1	Verify whether the slaves can be contacted. To do this, all of the configured slaves are briefly addressed. If all configured bus slaves in the ring can be contacted, communication phase 2 becomes active.
Phase 2	The master exchanges important communication parameters and data on general device properties with each slave sequentially. This defines and fixes the configuration of the cyclical channel. If all bus slaves are configured, communication phase 3 becomes active.
Phase 3	Parameter phase: The master exchange parameters with the slaves. All slaves can be addressed at once. Real-time data is not available yet.
Phase 4	Operating phase: Cyclical exchange of real-time data (cyclical communication). Any number of parameters can be read and written using the service channel (non-cyclical communication).

Sercos Devices

Introduction

The Modicon LMC078 Motion Controller supports the following Sercos devices:

- Lexium LXM32S drives
- TM5NS31 Sercos interface modules
- Third-party Sercos devices

Adding a Sercos Device

There are two ways to add Sercos devices:

- Using the Device Addressing Editor (*see page 209*).
- Using the method described below.

To add a Sercos device, select **Lexium 32 S** or **TM5NS31 Interface** in the Hardware Catalog, drag it to the **Devices tree**, and drop it on the **SERCOSIII** node.

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

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

Adding a Third-Party Sercos Device

You can also add Sercos Devices from Third-Party Vendors (*see SoMachine, Programming Guide*).

NOTE: Depending on the third-party vendor, it may be necessary to set the parameter `Producer-CycleTime` of the Sercos device to a value deviating from the default value (1 ms). For detailed information, refer to the specification in the device documentation provided by the third-party vendor.

Device Addressing Editor

Introduction

The **Device Addressing** editor supports the following functions:

- Scan of the devices connected to the Sercos bus
- Add new Sercos devices
- Define identification parameters of the Sercos devices

To access to the **Device Addressing** tool, select the **Tools tree** tab and double-click **Device Addressing**:

Color legend

- Login possible without any errors
- Unable to login without errors
- No scanned device allocated
- Login possible without any error (however differences in irrelevant values are marked)
- Login possible without any errors (but no scanned device assigned)

Adopt values of all assigned devices

Designation	Description
Devices in the PLC Configuration	<p>The left part of the editor window displays the LXM32S drives and the TM5NS31 interface modules of the Sercos configuration in the controller project.</p> <p>NOTE: If the Sercos objects are assigned automatically, they are listed in the order of their topological address, starting with the lowest value. If Sercos objects from a previous scan are already listed, any further Sercos objects are added at the end of the list, in the order of the topological address, starting with the lowest value.</p>
Operation Mode (see page 212)	<p>The operating mode is used to determine the way this device operates.</p> <p>NOTE: If you change a value in this column while pressing and holding the shift key, all values of this column are set to this value.</p>

Designation	Description
Start Sercos scan	Click this button to start searching for real LXM32S drives and the TM5NS31 interface modules that are connected to the Sercos bus. To perform the search: <ul style="list-style-type: none"> • The Sercos bus is switched to phase 0 (<i>see page 207</i>). • All applications are stopped. • All diagnostic messages must be confirmed.
Scanned devices	After performing a scan, the right part of the editor window displays the LXM32S drives and the TM5NS31 interface modules that are connected to the Sercos bus. The column header presents the number of devices scanned and the number of devices assigned automatically. All devices that were automatically assigned are highlighted in a non-white color. You can manually change the assignment later on by using a drop-down list in the right-most column.
<--	Click this button to apply the values of the Sercos device assigned in this row.
Adopt values of all assigned devices	After having assigned all LXM32S drives and TM5NS31 interface modules, click this button to apply the device data of the scanned Sercos devices in the assigned objects in the Sercos node of the controller configuration.
Add	Click this button to add some new devices to the Sercos node of the controller configuration.

NOTE: The functions of the **Device addressing** editor are only available in offline mode.

The color coding of a row indicates whether a login can be performed for the specific Sercos object in the controller configuration and, if so, whether a login is possible.

The colors have the following meanings:

Color	Description
Green	Login can be performed with this object.
Red	Login cannot be performed with this object.
White	No Sercos device is assigned to this object.
Yellow	Login to this object is uncertain. There are deviations in values of the assigned Sercos device. The mismatched values are highlighted in bold .
Pink	Login to this object is uncertain. However, no Sercos device was assigned after performing the Sercos scan ([Start Sercos scan]) even though the Operating mode is set to Real .


Sercos Scan

After scanning (**Start SERCOS scan** button]), the program attempts to assign Sercos objects from the controller configuration to the devices connected to the Sercos bus by using the topological address. All devices that were automatically assigned are highlighted in a non-white color.

The column header presents the number of devices scanned and the number of devices assigned automatically:

Scanned devices	
2 scanned, 2 assigned devices	
<--	1 LXM32S 2324035233 LEXIUM_SERVO 2
<--	2 TM5NS31 B37C0170632 Modular IO Device 3

You can manually change the automatic assignment later on by using a drop-down list in the right-most column:

- Click the  button in the row that you want to change. The drop-down list displays all Sercos devices of this type that have not been assigned yet.
- Select the desired device from the list.

NOTE: You can use the empty row at the bottom of the list to reset an assignment.

Each row of the selection list contains a short description of the Sercos device.

The values are separated by a vertical bar ("|") and correspond to the following parameters:

- TopologyAddress
- ObjectType
- SerialNumber
- ConfiguredApplicationType
- SercosAddress

You have two possibilities to adopt the parameter values:

- You can apply the values of an assigned device by clicking the <-- button.
- You can apply the values of all assigned devices by clicking the **Adopt values of all assigned devices**.

As you apply these values, the program writes the values of the assigned and scanned Sercos devices that are required for commissioning into the related Sercos objects in the controller configuration.

NOTE: After the values have been applied, the corresponding row is highlighted in green.

Operation Mode

The **Operation mode** is used to determine the way a Sercos device operates.

Select the desired operating mode in the drop-down list in the **Operation mode** column:

Operation mode	Description
Virtual	The Sercos device does not exist physically.
Real	The Sercos device must exist physically.
Deactivated	The Sercos device is not in use. It may exist physically.
Optional	The Sercos device may exist physically, but this is not a prerequisite.

NOTE: If you change a value in this column while pressing and holding the shift key, all values of this column are set to this value.

Add Devices Manually

In the **Device Addressing** editor, you can add manually some new devices to the controller configuration.

To add devices manually, proceed as follows:

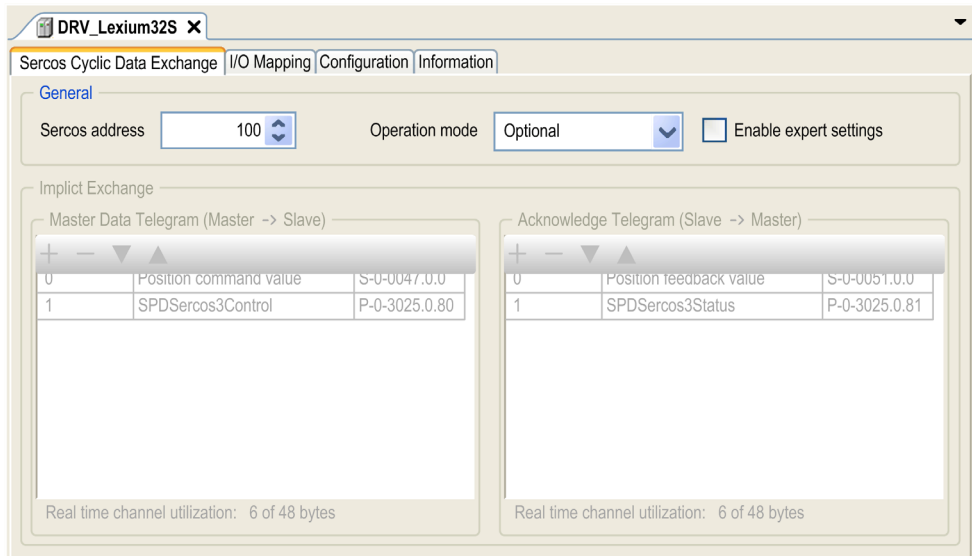
Step	Action
1	Enter the desired number of the new devices.
2	Select the desired device from the list: <ul style="list-style-type: none"> ● LXM32S: Lexium 32S drive ● TM5NS31: TM5 Sercos interface module
3	Click Add . Result: The devices are added to the controller configuration.

NOTE: If an error is detected when adding a device, the devices that could not be added are listed with a respective explanation in the message window.

Lexium LXM32S Drive Configuration

Description

To access to the device editor screen, double-click the drive node in the **Devices tree**:



The device editor screen of the LXM32S drive contains the followings tabs:

- **Sercos Cyclic Data Exchange:**
 - Configuration of the Sercos address of the drive.
 - Configuration of the Operating mode (*see page 212*) of the drive.
 - Configuration of the Sercos implicit exchanges (IDN configuration of MDT and AT telegrams).
- **I/O Mapping:** This tab allows you to create and assign IEC variables to the IDN selected for cyclical exchanges.
- **Configuration:** Configuration parameters of the drive (use the **Sercos Cyclic Data Exchange** tab to configure drive parameters).
- **Information:** This tab displays general information about the device (name, description, provider, version, image).

NOTE:

The default configuration of the cyclical exchange contains four IDN, not editable:

- MDT telegram (controller to drive):
 - Position command value: S-0-0047.0.0
 - SPDSercos3Control: P-0-3025.0.80
- AT telegram (drive to controller):
 - Position feedback value: S-0-0051.0.0
 - SPDSercos3Status: P-0-3025.0.81

For more information about the IDN implemented in the LXM32S drives, refer to the Lexium LXM32S Product Manual.

Expert Setting Configuration

The expert setting option allows you to modify the list of Sercos IDN exchanged cyclically between the controller and the drive.

NOTE: The total IDN length of the MDT and AT telegrams is limited to 48 bytes.

To activate the expert setting, tick the check box **Enable expert settings**:

The screenshot shows the configuration interface for a Lexium32S drive. The 'Sercos Cyclic Data Exchange' tab is selected, and the 'General' sub-tab is active. The 'Sercos address' is set to 100, and the 'Operation mode' is set to 'Optional'. The 'Enable expert settings' checkbox is checked. The 'Implicit Exchange' section is expanded, showing two tables of IDNs. The 'Master Data Telegram (Master -> Slave)' table lists four IDNs: Position command value (S-0-0047.0.0), SPDSercos3Control (P-0-3025.0.80), Position at reference point (P-0-3040.0.11), and Monitoring of velocity threshold (P-0-3006.0.27). The 'Acknowledge Telegram (Slave -> Master)' table lists five IDNs: Position feedback value (S-0-0051.0.0), SPDSercos3Status (P-0-3025.0.81), Physical status of the digital inputs and outputs (P-0-3008.0.1), Active operating mode (P-0-3027.0.4), and Actual velocity (P-0-3030.0.32). Both tables show a real-time channel utilization of 14 of 48 bytes.

IDN	Parameter Name	Value
0	Position command value	S-0-0047.0.0
1	SPDSercos3Control	P-0-3025.0.80
2	Position at reference point	P-0-3040.0.11
3	Monitoring of velocity threshold	P-0-3006.0.27

Real time channel utilization: 14 of 48 bytes

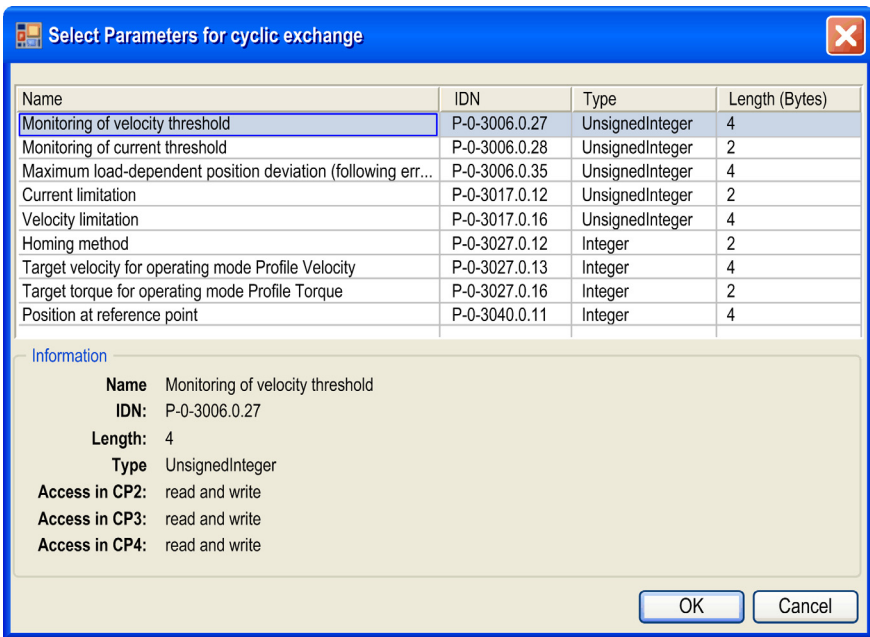
IDN	Parameter Name	Value
0	Position feedback value	S-0-0051.0.0
1	SPDSercos3Status	P-0-3025.0.81
2	Physical status of the digital inputs and outputs	P-0-3008.0.1
3	Active operating mode	P-0-3027.0.4
4	Actual velocity	P-0-3030.0.32

Real time channel utilization: 14 of 48 bytes

The following buttons are now available:

Button	Description
+	Click this button to add an IDN to the list (see description here-after).
-	Select an IDN in the list and click this button to remove an IDN from the list.
Up arrow	Select an IDN in the list and click this button to move up the IDN in the list.
Down arrow	Select an IDN in the list and click this button to move down the IDN in the list.

To add an IDN to the MDT or AT telegram, proceed as fo follows:

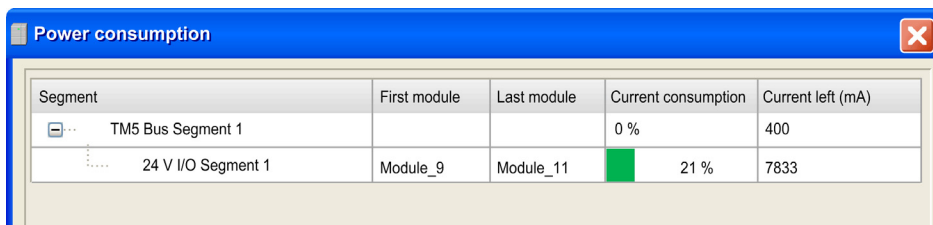
Step	Action
1	<p>Click +.</p> <p>Result: The following dialog box appears:</p> 
2	Select the IDN to add in the list.
3	<p>Click Ok.</p> <p>Result: The IDN is added to the MDT or AT telegram and the number of bytes used is updated (Real time channel utilization).</p>

TM5NS31 Sercos Interface Module

Power Consumption

To display the estimated power consumption of the expansion modules:

Step	Action
1	Right-click the TM5NS31 Interface node of the Device tree .
2	Select Power consumption .



Segment	First module	Last module	Current consumption	Current left (mA)
TM5 Bus Segment 1			0 %	400
24 V I/O Segment 1	Module_9	Module_11	21 %	7833

NOTE: The current consumption figures presented by the **Power consumption** function are based on assumed values, and not on actual current measurements. The assumed values for the outputs are based on classic loads but can be adjusted using the 24 Vdc I/O segment external current setting in the I/O Configuration tab of each module. The assumptions for input signals are based on known internal loads and are therefore not modifiable. While the use of the **Power consumption** function to test the power budget is required, it is no substitute for actual and complete system testing and commissioning. Refer to the TM5 / TM7 System Planning and Installation Guide.

Sercos Error Codes

Sercos Slave Error Messages

4-digit codes identify error messages reported to the master by Sercos slaves. The following are standard Sercos error codes:

Error code	Description	Comment
0x0nnn	General error	–
0x0000	No error in the service channel	–
0x0001	Service channel not open	–
0x0009	Invalid access to closing the service channel	–
0x1nnn	Element 1	Identification number
0x1001	IDN not supported	–
0x1009	Invalid access to element 1	–
0x2nnn	Element 2	Name
0x2001	Name not supported	–
0x2002	Name transmission too short	Master set "last transmission" too early
0x2003	Name transmission too long	Master does not set "last transmission"
0x2004	Name cannot be changed	Name is read only
0x2005	Name is write-protected	–
0x3nnn	Element 3	Attribute
0x3002	Attribute transmission too short	Master set "last transmission" too early
0x3003	Attribute transmission too long	Master does not set "last transmission"
0x3004	Attribute cannot be changed	Attribute is read only
0x3005	Attribute is write-protected	–
0x4nnn	Element 4	Unit
0x4001	Unit not supported	–
0x4002	Unit transmission too short	Master set "last transmission" too early
0x4003	Unit transmission too long	Master does not set "last transmission"
0x4004	Unit cannot be changed	Unit is read only
0x4005	Unit is write-protected	–
0x5nnn	Element 5	Minimum input value
0x5001	Minimum input value not supported	–
0x5002	Minimum input value transmission too short	Master set "last transmission" too early
0x5003	Minimum input value transmission too long	Master does not set "last transmission"
0x5004	Minimum input value cannot be changed	Minimum input value is read only

Error code	Description	Comment
0x5005	Minimum input value is write-protected	–
0x6nnn	Element 6	Maximum input value
0x6001	Maximum input value not supported	–
0x6002	Maximum input value transmission too short	Master set “last transmission” too early
0x6003	Maximum input value transmission too long	Master does not set “last transmission”
0x6004	Maximum input value cannot be changed	Maximum input value is read only
0x6005	Maximum input value is write-protected	–
0x7nnn	Element 7	Operation data
0x7002	Operation data transmission too short	Master set “last transmission” too early
0x7003	Operation data transmission too long	Master does not set “last transmission”
0x7004	Operation data cannot be changed	Operation data is read only
0x7005	Operation data is write-protected at this communication phase	–
0x7006	Operation data is smaller than the minimum input value	–
0x7007	Operation data is greater than the maximum input value	–
0x7008	Invalid operation data	The invalid operation data may be an unsupported: <ul style="list-style-type: none"> ● bit number or bit combination, ● value, code or ● configured IDN
0x7009	Operation data write-protected by a password	–
0x700A	Operation data is write-protected, it is configured cyclically	IDN is configured in the MDT or AT. Therefore writing via the service channel is not allowed
0x700B	Invalid indirect addressing	e.g., data container, list handling, etc
0x700C	Operation data is write-protected, due to other settings	e.g., operation mode, sub-device is enabled, setting of communication version, etc
0x700D	Invalid floating point number	–
0x700E	Operation data is write-protected at parametrization level	–
0x700F	Operation data is write-protected at operating level	–
0x7010	Procedure command already active	–
0x7011	Procedure command not interruptible	–
0x7012	Procedure command not executable	e.g., in this phase the procedure command can not be activated
0x7013	Procedure command not executable	The corresponding parameters are invalid

Error code	Description	Comment
0x7014	The received current length of list parameter does not match to expectation	–
0x7015	Operation data is not yet created completely	If it takes more time to create the operation data, try again later
0x71nn	Segment wise transmission of list parameters via SVC	–
0x7101	IDN in S-0-0394 not valid	–
0x7102	Empty list in S-0-0397 not allowed for write access	–
0x7103	Maximum length of the list in S-0-0394 is exceeded by take-over of the list segment	–
0x7104	Read access only: the length of the list segment as of the list index exceeds the current length of the list in S-0-0394	–
0x7105	IDN in S-0-0394 is write-protected	–
0x7106	Operation data in list segment is smaller than the minimum input value	–
0x7107	Operation data in list segment is greater than the maximum input value	–
0x7108	Invalid list index in IDN S-0-0395	–
0x7109	Parameter in IDN S-0-0394 does not have variable length	–
0x710A	IDN S-0-0397 not permitted as operation data in S-0-0394	–
0x8nnn	Reserved for master internal error codes	Error codes may be defined by the manufacturer of control units (e.g., NC, PLC)
0xAxxx	Reserved	–
0xBxxx	Reserved	–
0xCxxx	Reserved for slave specific error codes	Used for error analysis and trace functionality (troubleshooting)
0xDxxx	Error codes are not generated and transmitted via SVC	Error codes are defined by a TWG of Sercos
0xD000	No error	–
0xD001	Service channel (temporarily) not available	–
0xD002	Service channel engaged by an application	–
0xD003	Service channel busy, slave is processing previous request	–
0xD004	Sercos slave not reachable	–
0xD005	Service channel transaction aborted	–

Error code	Description	Comment
0xD006	Writing this element is not supported by the service channel	–
0xEnnn	Reserved for master internal error codes	–
0xFnnn	Reserved for master internal error codes	–

NOTE: All other error codes are reserved.

Sercos Master Error Messages

If errors are recognized by the Sercos master, they are specified either as a 5-digit hexadecimal value, or a negative decimal value.

Error Code	Description
20001	SVC: New request with higher priority during active internal request
20002	SVC: New internal request during active internal request
20003	SVC: Transmission canceled by another function call with higher priority
20004	SVC: New transmission requested but MBusy is not set
20005	SVC: Invalid state: AHS != MHS during set BusyAT
20006	SVC: Timeout because slave has not set the BusyAT flag
20007	SVC: Timeout because slave has set BusyAT flag for too long
20008	SVC: Write with unsupported element (allowed 1 or 7)
20009	SVC: Write with data length = zero
-1	Other error
-431	Service request error (e.g. timeout)
-445	Service timeout
-467	Internal state machine error

Chapter 14

Serial Line Configuration

Introduction

This chapter describes how to configure the serial line communication of the Modicon LMC078 Motion Controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Serial Line Configuration	222
ASCII Manager	224
SoMachine Network Manager	226
Modbus Serial IOScanner	227
Adding a Device on the Modbus Serial IOScanner	229
Modbus Manager	236
Adding a Modem to a Manager	240

Serial Line Configuration

Introduction

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

Serial Line Configuration

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

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

The screenshot shows a configuration window with the following settings:

- Serial line**
 - Baud rate: 19200
 - Parity: Even
 - Data bits: 8
 - Stop bits: 1
- Physical Medium**
 - RS 485
 - RS 232
 - Polarisation Resistor: No

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

Element	Description
Baud rate	Transmission speed in bits/s
Parity	Used for error detection
Data bits	Number of bits for transmitting data
Stop bits	Number of stop bits
Physical Medium	Specify the medium to use: <ul style="list-style-type: none"> ● RS485 (using polarisation resistor or not) ● RS232
Polarization Resistor	Polarization resistors are integrated in the controller. They are switched on or off by this parameter.

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

NOTICE

INTERRUPTION OF SERIAL LINE COMMUNICATIONS

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

Failure to follow these instructions can result in equipment damage.

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

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

ASCII Manager

Introduction

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

Adding the Manager

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

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

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

ASCII Manager Configuration

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

The ASCII Manager configuration window is displayed as below:

The screenshot shows the ASCII Manager configuration window with three tabs: Configuration (selected), Status, and Information. The window is divided into two main sections: ASCII and Serial Line Settings.

ASCII Section:

Start Character:	<input type="text" value="0"/>	Frame Length Received:	<input type="text" value="0"/>
First End Character:	<input type="text" value="10"/>	Frame received Timeout (ms):	<input type="text" value="0"/>
Second End Character:	<input type="text" value="0"/>		

Serial Line Settings Section:

Baud Rate:	115200
Parity:	None
Data Bits:	8
Stop Bits:	1
Physical Medium:	RS485

Set the parameters as described in this table:

Parameter	Description
Start Character	If 0, no start character is used in the frame. Otherwise, in Receiving Mode , the corresponding character in ASCII is used to detect the beginning of a frame. In Sending Mode , this character is added at the beginning of the frame.
First End Character	If 0, no first end character is used in the frame. Otherwise, in Receiving Mode , the corresponding character in ASCII is used to detect the end of a frame. In Sending Mode , this character is added at the end of the frame.
Second End Character	If 0, no second end character is used in the frame. Otherwise, in Receiving Mode , the corresponding character in ASCII is used to detect the end of a frame. In Sending Mode , this character is added at the end of the frame.
Frame Length Received	If 0, this parameter is not used. This parameter allows the system to conclude an end of frame at reception when the controller received the specified number of characters. Note: This parameter cannot be used simultaneously with Frame Received Timeout (ms) .
Frame Received Timeout (ms)	If 0, this parameter is not used. This parameter allows the system to conclude the end of frame at reception after a silence of the specified number of ms.
Serial Line Settings	Parameters specified in the Serial Line configuration window (<i>see page 222</i>).

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

Adding a Modem

To add a Modem to the ASCII manager, refer to Adding a Modem to a Manager (*see page 240*).

SoMachine Network Manager

Introduction

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

Adding the Manager

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

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

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

Configuring the Manager

There is no configuration for SoMachine Network Manager.

Adding a Modem

To add a modem to the SoMachine Network Manager, refer to Adding a Modem to a Manager (*see page 240*).

Modbus Serial IOScanner

Introduction

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

Add a Modbus IOScanner

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

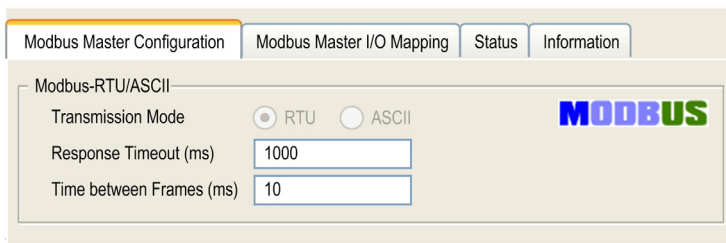
For more information on adding a device to your project, refer to:

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

Modbus IOScanner Configuration

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

The configuration window is displayed as below:



Set the parameters as described in this table:

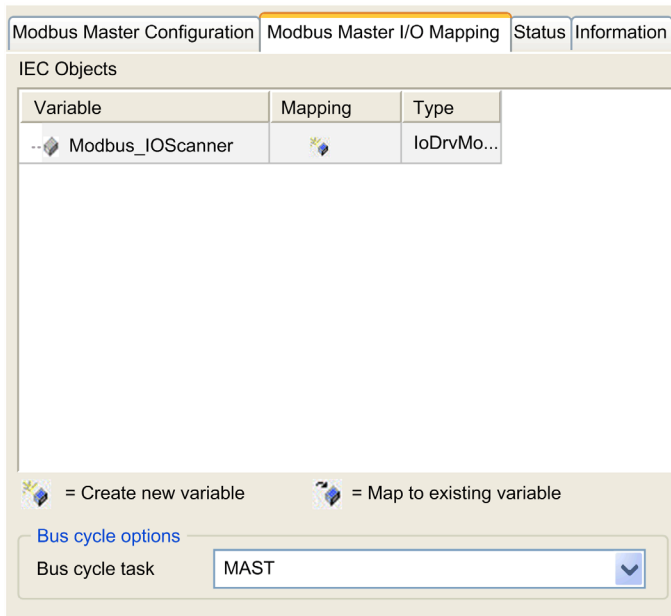
Element	Description
Transmission Mode	Specifies the transmission mode to use: <ul style="list-style-type: none"> • RTU: uses binary coding and CRC error-checking (8 data bits) • ASCII: messages are in ASCII format, LRC error-checking (7 data bits) Set this parameter identical for each Modbus device on the network.
Response Timeout (ms)	Timeout used in the exchanges.
Time between Frames (ms)	Delay to reduce data collision on the bus. Set this parameter identical for each Modbus device on the network.

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.

Bus Cycle Task Selection

The Modbus IOScanner and the devices exchange data at each cycle of the chosen application task.

To select this task, select the **Modbus Master IO Mapping** tab. The configuration window is displayed as below:



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

- **Use parent bus cycle setting:** associate the scanner with the application task that manages the controller.
- **MAST:** associate the scanner with the MAST task.
- **Another existing task:** you can select an existing task and associate it to the scanner. For more information about the application tasks, refer to the SoMachine Programming Guide.

The scan time of the task associated with the scanner must be less than 500 ms.

Adding a Device on the Modbus Serial IOScanner

Introduction

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

Add a Device on the Modbus IOScanner

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

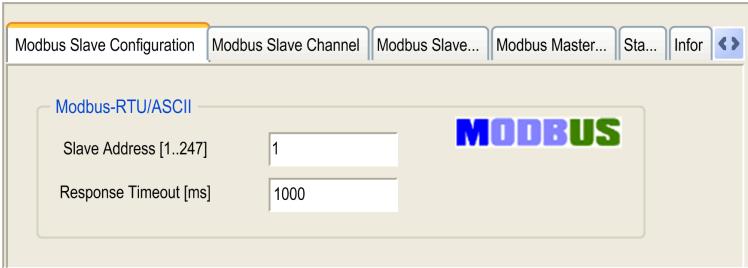
For more information on adding a device to your project, refer to:

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

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

Configure a Device Added on the Modbus IOScanner

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

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

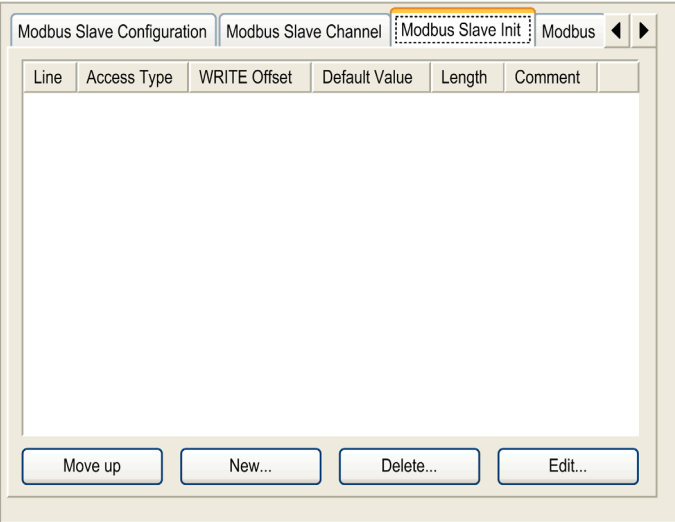
To configure the **Modbus Channels**, proceed as follow:

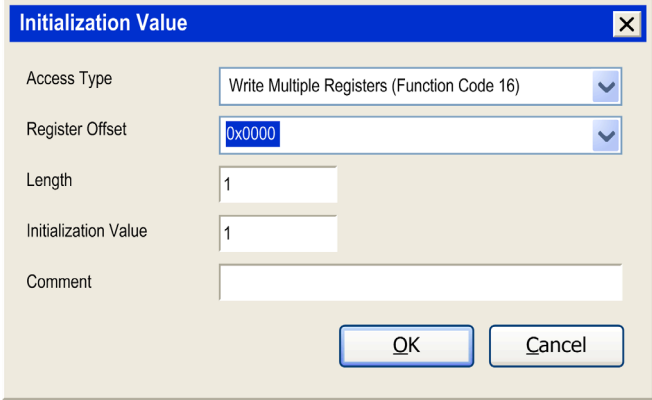
Step	Action
1	<p>Click the Modbus Slave Channel tab:</p> <p>The screenshot shows a software window titled "Modbus Slave Configuration". The "Modbus Slave Channel" tab is selected and highlighted with an orange border. The window contains a table with the following columns: Name, Access Type, Trigger, READ Offset, Length, Error Handling, and WRIT... Below the table, there are three buttons: "Add Channel...", "Delete...", and "Edit...".</p>

Step	Action
2	<p>Click the Add Channel button:</p> <div data-bbox="378 233 1108 971"><p>ModbusChannel</p><p>Channel</p><p>Name: Channel 1</p><p>Access Type: Read/Write Multiple Registers (Function Code 23)</p><p>Trigger: CYCLIC Cycle Time (ms): 100</p><p>Comment:</p><p>READ Register</p><p>Offset: 0x0000</p><p>Length: 1</p><p>Error Handling: Keep last Value</p><p>WRITE Register</p><p>Offset: 0x0000</p><p>Length: 1</p><p>OK Cancel</p></div>

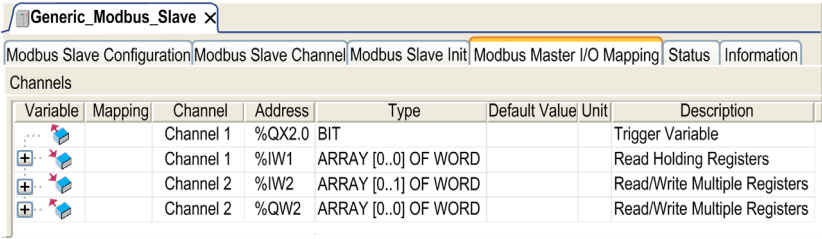
Step	Action
3	<p>Configure an exchange:</p> <p>In the field Channel, you can add the following values:</p> <ul style="list-style-type: none"> ● Channel: Enter a name for your channel. ● Access Type: Choose the exchange type: Read or Write or Read/Write multiple registers (i.e. %MW) (<i>see page 235</i>). ● 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. <p>In the field READ Register (if your channel is Read or Read/Write one), you can configure the %MW to be read on the Modbus slave. Those will be mapped on %IW (see Modbus Master I/O Mapping tab):</p> <ul style="list-style-type: none"> ● 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. <p>In the field WRITE Register (if your channel is Write or Read/Write one), you can configure the %MW to be written to the Modbus slave. Those will be mapped on %QW (see Modbus Master I/O Mapping tab):</p> <ul style="list-style-type: none"> ● Offset: Offset of the %MW to write. 0 means that the first object that will be written will be %MW0. ● Length: Number of %MW to be written. For example, if 'Offset' = 2 and 'Length' = 3, the channel will write %MW2, %MW3 and %MW4.
5	<p>Click OK to validate the configuration of this channel.</p> <p>NOTE: You can also:</p> <ul style="list-style-type: none"> ● Click the Delete button to remove a channel. ● Click the Edit button to change the parameters of a channel.

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

Step	Action
1	<p>Click the Modbus Slave Init tab:</p>  <p>The screenshot shows a software window titled "Modbus Slave Configuration". It has four tabs: "Modbus Slave Configuration", "Modbus Slave Channel", "Modbus Slave Init", and "Modbus". The "Modbus Slave Init" tab is selected and highlighted with a red box. Below the tabs is a table with the following columns: "Line", "Access Type", "WRITE Offset", "Default Value", "Length", and "Comment". The table is currently empty. At the bottom of the window, there are four buttons: "Move up", "New...", "Delete...", and "Edit...".</p>

Step	Action
2	<p>Click New to create a new initialization value:</p>  <p>The Initialization Value window contains the following parameters:</p> <ul style="list-style-type: none"> ● Access Type: Choose the exchange type: Read or Write or Read/Write multiple registers (that is, %MW) (<i>see page 235</i>). ● Register Offset: Register number of register to be initialized. ● Length: Number of %MW to be read. For example, if 'Offset' = 2 and 'Length' = 3, the channel will read %MW2, %MW3 and %MW4. ● Initialization Value: Value the registers are initialized with. ● Comment: Add a comment about this channel.
4	<p>Click OK to create a new Initialization Value.</p> <p>NOTE: You can also:</p> <ul style="list-style-type: none"> ● Click Move up to change the position of a value in the list. ● Click Delete to remove a value in the list. ● Click Edit to change the parameters of a value.

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

Step	Action
1	<p>Click the Modbus Master I/O Mapping tab:</p> 
2	<p>Double-click in a cell of the Variable column to open a text field. Enter the name of a variable or click the browse button [...] and chose a variable with the Input Assistant.</p>
3	<p>For more information on I/O mapping, refer to SoMachine Programming Guide.</p>

Access Types

This table describes the different access types available:

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

Modbus Manager

Introduction

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

Adding the Manager

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

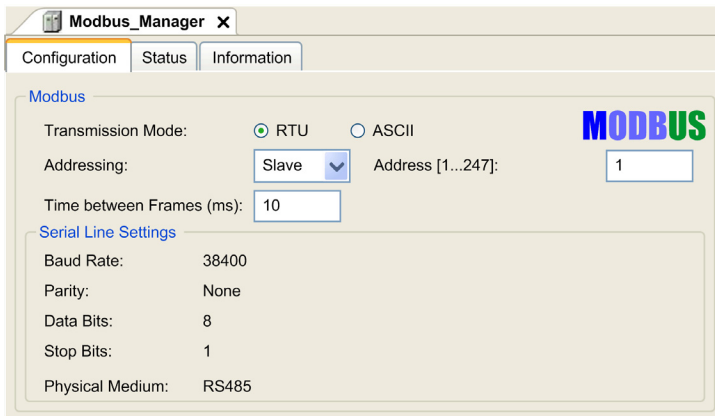
For more information on adding a device to your project, refer to:

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

Modbus Manager Configuration

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

The Modbus Manager configuration window is displayed as below:



Set the parameters as described in this table:

Element	Description
Transmission Mode	Specify the transmission mode to use: <ul style="list-style-type: none"> • RTU: uses binary coding and CRC error-checking (8 data bits) • ASCII: messages are in ASCII format, LRC error-checking (7 data bits) Set this parameter identical for each Modbus device on the link.
Addressing	Specify the device type: <ul style="list-style-type: none"> • Master • Slave

Element	Description
Address	Modbus address of the device, when slave is selected.
Time between Frames (ms)	Time to avoid bus-collision. Set this parameter identical for each Modbus device on the link.
Serial Line Settings	Parameters specified in the Serial Line configuration window.

Modbus Master

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

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

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

Modbus Slave

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

Function Code Dec (Hex)	Sub-Function Dec (Hex)	Function
1 (1 hex)	–	Read digital outputs (%Q)
2 (2 hex)	–	Read digital inputs (%I)
3 (3 hex)	–	Read multiple register (%MW)
5 (5 hex)	–	Write single coil (%M)
6 (6 hex)	–	Write single register (%MW)
8 (8 hex)	–	Diagnostic
15 (F hex)	–	Write multiple digital outputs (%Q)
16 (10 hex)	–	Write multiple registers (%MW)
23 (17 hex)	–	Read/write multiple registers (%MW)
43 (2B hex)	14 (E hex)	Read device identification

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

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

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

Object ID	Object Name	Type	Value
00 hex	Vendor code	ASCII String	Schneider Electric
01 hex	Product code	ASCII String	Controller reference LMC078CECS20T
02 hex	Major / Minor revision	ASCII String	aa.bb.cc.dd (same as device descriptor)

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

WARNING

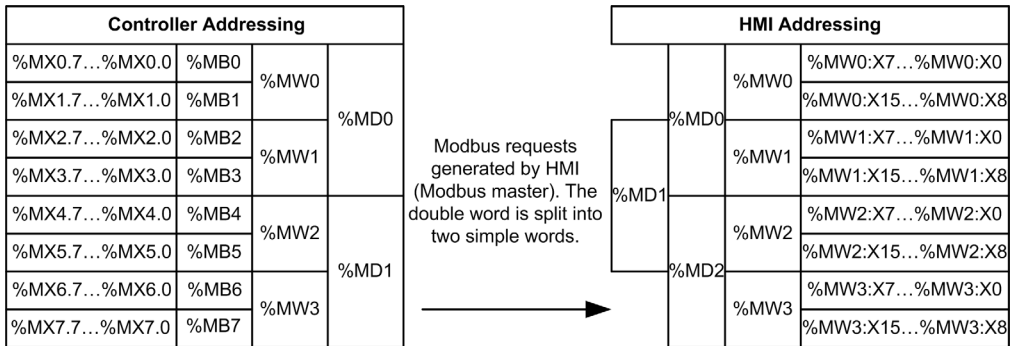
UNINTENDED EQUIPMENT OPERATION

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

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

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

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



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

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

The following gives examples of memory match for the bits:

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

Adding a Modem

To add a Modem to the Modbus Manager, refer to Adding a Modem to a Manager ([see page 240](#)).

Adding a Modem to a Manager

Introduction

A modem can be added to the following managers:

- ASCII Manager
- Modbus Manager
- SoMachine Network Manager

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

Adding a Modem to a Manager

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

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

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

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

Chapter 15

Connecting a Modicon LMC078 Motion Controller to a PC

Connecting the Controller to a PC

Overview

To transfer, run, and monitor the applications, connect the controller to a computer that has SoMachine installed, using either a USB cable or an Ethernet connection.

<i>NOTICE</i>

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 to the PC at any one time.

NOTE: The LMC078 Motion Controller must be selected in the Gateway Management Console, accessible by double-clicking the **Gateway Management Console** icon  in the Windows notification area. This option is not selected by default.

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

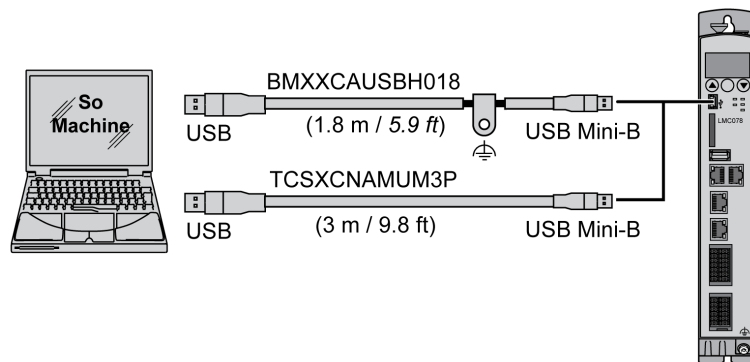
UNINTENDED EQUIPMENT OPERATION OR INOPERABLE EQUIPMENT

- You must use a shielded USB cable such as a BMXXCAUSBH0** 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.
- Do not use the USB port(s), if so equipped, unless the location is known to be non-hazardous.

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 presents the USB connection to a PC:



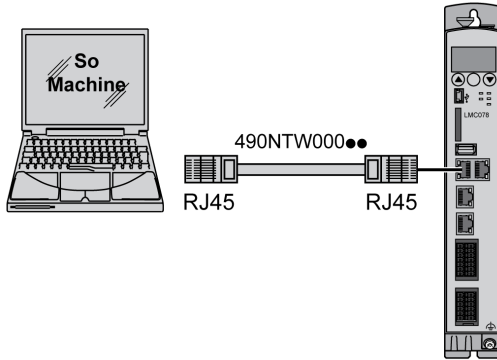
To connect the USB cable to your controller, follow the steps below:

Step	Action
1	<p>1a If making a long-term connection using the cable BMXXCAUSBH045, or other cable with a ground shield connection, 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.</p> <p>1b If making a short-term connection using the cable TCSXCNAMUM3P or other non-grounded USB cable, proceed to step 2.</p>
2	Connect the USB cable connector to the PC.
3	Connect the Mini-B connector of your USB cable to the controller USB connector.

Ethernet Port Connection

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

The following illustration presents the Ethernet connection to a PC:



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

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

NOTE: The default IP address (*see page 172*) is 190.201.100.100.

Chapter 16

Firmware Update

Updating Modicon LMC078 Motion Controller Firmware

Introduction

The firmware updates for Modicon LMC078 Motion Controller are available on the <http://www.schneider-electric.com> website.

The firmware update is possible by using the **Controller Assistant** software.

The **Controller Assistant** provides two different ways to update the firmware:

- The first firmware update procedure automatically removes the application in the controller.
- The second firmware update procedure does not remove the application from the controller.

Firmware Update Automatically Removing the Application

Performing a firmware change deletes the current application program in the device, including the Boot Application in the SD card.

<i>NOTICE</i>
LOSS OF APPLICATION DATA <ul style="list-style-type: none">• Perform a backup of the application program to the hard disk of the PC before attempting a firmware update.• Restore the application program to the device after a successful firmware update. Failure to follow these instructions can result in equipment damage.

If you remove power to the device, or there is a power outage or communication interruption during the transfer of the application, your device may become inoperative. If a communication interruption or a power outage occurs, reattempt the transfer. If there is a power outage or communication interruption during a firmware update, or if an invalid firmware is used, your device will become inoperative. In this case, use a valid firmware and reattempt the firmware update.

NOTICE

INOPERABLE EQUIPMENT

- Do not interrupt the transfer of the application program or a firmware change once the transfer has begun.
- Re-initiate the transfer if the transfer is interrupted for any reason.
- Do not attempt to place the device (logic controller, motion controller, HMI controller or drive) into service until the file transfer has completed successfully.

Failure to follow these instructions can result in equipment damage.

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

To execute a complete firmware update of a controller, proceed as follows:

Step	Action
1	On the Home dialog, click the Update firmware... button. Result: The Update firmware dialog opens.
2	Proceed as described in the chapter <i>Updating the Firmware</i> of the <i>SoMachine Controller Assistant User Guide</i> .

Firmware Update Without Removing the Application

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

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

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

Step	Action
6	On the Select next action dialog, click the Write on controller.... button. Result: The Controller selection dialog opens.
7	Select the required connection type and controller and click the Write button. Result: The image is transmitted from your computer to the controller. After the transmission, you are redirected to the Home dialog.

For more information about the firmware update and creating a new SD card with firmware, refer to the *SoMachine Controller Assistant User Guide*.

Appendices



Overview

This appendix lists the documents necessary for technical understanding of the Modicon LMC078 Motion Controller Programming Guide.

What Is in This Appendix?

The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	How to Change the IP Address of the Controller	251
B	Diagnostic Messages	255
C	LMC078 Sercos3 Library	271
D	Functions to Get/Set Serial Line Configuration in User Program	299
E	Controller Performance	305

Appendix A

How to Change the IP Address of the Controller

changeIPAddress: Change the IP address of the controller

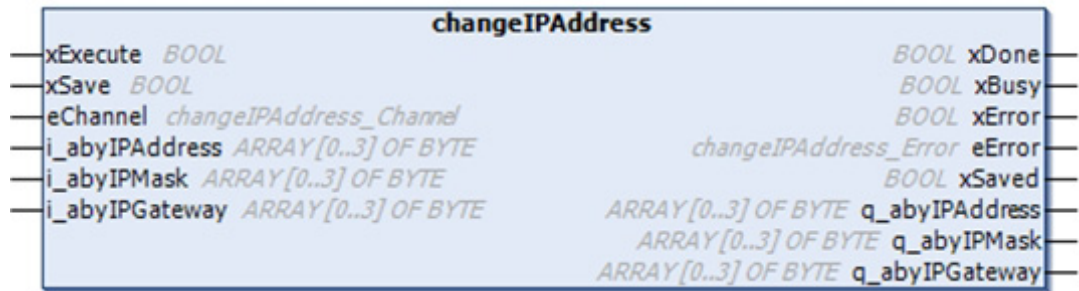
Function Block Description

The `changeIPAddress` function block provides the capability to change dynamically a controller IP address, its subnet mask and its gateway address. The function block can also save the IP address so that it is used in subsequent reboots of the controller.

NOTE: Changing the IP addresses is only possible if the IP mode is configured to **fixed IP address**. For more details, refer to IP Address Configuration (*see page 169*).

NOTE: For more information on the function block, use the **Documentation** tab of SoMachine Library Manager Editor. For the use of this editor, refer SoMachine Programming Guide.

Graphical Representation



Parameter Description

Input	Type	Comment
xExecute	BOOL	<ul style="list-style-type: none"> 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 that an error is detected. In this case, the corresponding output values (<code>xDone</code>, <code>xError</code>, <code>iError</code>) are present at the outputs for exactly one cycle.
xSave	BOOL	TRUE: save configuration for subsequent reboots of the controller.

Input	Type	Comment
eChannel	changeIPAddress_Channel	The input eChannel is the Ethernet port to be configured. Depending on the number of the ports available on the controller, it is one of 2 values (<i>see page 252</i>) in changeIPAddress_Channel (0 or 1).
i_abyIPAddress	ARRAY[0..3] OF BYTE	The new IP Address to be configured. Format: 0.0.0.0. NOTE: If this input is set to 0.0.0.0 then the controller default IP addresses (<i>see page 172</i>) is configured.
i_abyIPMask	ARRAY[0..3] OF BYTE	The new subnet mask. Format: 0.0.0.0
i_abyIPGateway	ARRAY[0..3] OF BYTE	The new gateway IP address. Format: 0.0.0.0

Output	Type	Comment
xDone	BOOL	TRUE: if IP Addresses have been successfully configured or if default IP Addresses have been successfully configured because input i_abyIPAddress is set to 0.0.0.0.
xBusy	BOOL	Function block active.
xError	BOOL	<ul style="list-style-type: none"> ● TRUE: error detected, function block aborts action. ● FALSE: no error has been detected.
eError	changeIPAd-dress_Error	Error code of the detected error (<i>see page 253</i>).
xCreated	BOOL	Configuration saved for the subsequent reboots of the controller.
q_abyIPAddress	ARRAY[0..3] OF BYTE	Current controller IP address. Format: 0.0.0.0.
q_abyIPMask	ARRAY[0..3] OF BYTE	Current subnet mask. Format: 0.0.0.0.
q_abyIPGateway	ARRAY[0..3] OF BYTE	Current gateway IP address. Format: 0.0.0.0.

changeIPAddress_Channel: Ethernet port to be configured

The changeIPAddress_Channel enumeration data type contains the following values:

Enumerator	Value	Description
CHANNEL_ETHERNET_NETWORK	0	M241, M251MESC, M258, LMC058, LMC078: Ethernet port M251MESE: Ethernet_2 port
CHANNEL_DEVICE_NETWORK	1	M241: TM4ES4 Ethernet port M251MESE: Ethernet_1 port

changeIPAddress_Error: Error Codes

The changeIPAddress_Error enumeration data type contains the following values:

Enumerator	Value	Description
ERR_NO_ERROR	00 hex	No error detected.
ERR_UNKNOWN	01 hex	Internal error detected.
ERR_INVALID_MODE	02 hex	IP address is not configured as a fixed IP address.
ERR_INVALID_IP	03 hex	Invalid IP address.
ERR_DUPLICATE_IP	04 hex	The new IP address is already used in the network.
ERR_WRONG_CHANNEL	05 hex	Incorrect Ethernet communication port.
ERR_IP_BEING_SET	06 hex	IP address is already being changed.
ERR_SAVING	07 hex	IP addresses not saved due to a detected error or no non-volatile memory present.

Appendix B

Diagnostic Messages

Introduction

This chapter describes the diagnostic messages of the Modicon LMC078 Motion Controller.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Message Logger	256
Diagnostic Messages	262

Message Logger

Overview

The message logger records the important events on the controller. This information is categorized, evaluated, and clearly displayed. In case an error is detected, this information helps to solve a problem or localize the error.

To open the **Message logger** window, select the **Tool tree** and double-click the **Message logger** node:

Messagelogger	No.	Timestamp	Type	Object	Instance	Diag. code
+ .. LMCxx8 ...	1	Mi 14.Jul 2010 12:06:28				
+ .. LMCxx8 ...	2	Mi 14.Jul 2010 12:06:50				
+ .. LMCxx8 ...	3	Mi 14.Jul 2010 12:07:01				
+ .. LMCxx8 ...	4	Mi 14.Jul 2010 12:07:10				
+ .. LMCxx8 ...	5	Mi 14.Jul 2010 12:08:02				

This data view displays the information of the message logger. For example, you can consecutively call and process data of various message loggers from the controller. You can also add message logger files which have previously been saved.

It is also possible to

- Save the displayed message loggers as a file.
- Remove the displayed message loggers from the list.

Further, it is possible to

- Provide every line with its own comment.
- Mark all lines with certain properties in a different color.

These options are available directly through the context menu (right-click).

The displayed time corresponds to the time the item was added to this dialog. The numbering is continuous.

You can open the selected message logger by clicking the plus (+) icon on the left of the logger. To close it, click the minus (-) icon:

Message logger	No.	Timestamp	Type	Object	Instance	Diag. code	Ext. diagnosis	Message
LMCxx8 r...	1	Wed 02.Apr 2014 08:45:24						
	500	We Apr/02/2014 08:32:57...	1	LMCxx8	MyCon...	8003		Controller boot finished
	499	We Apr/02/2014 08:32:56...	1	LMCxx8	MyCon...	8051		LMC 078/AX=8/RAM=500/n
	498	We Apr/02/2014 08:32:56...	1	SERC32	Sercos...	8042	CP4	Sercos phase switched
	497	We Apr/02/2014 08:32:56...	1	SERC32	Sercos...	8042	CP3/1 ms	Sercos phase switched
	496	We Apr/02/2014 08:32:55...	1	LMCxx8	MyCon...	8970	end	Fast Device Replacement
	495	We Apr/02/2014 08:32:55...	1	LMCxx8	MyCon...	8970	start	Fast Device Replacement
	494	We Apr/02/2014 08:32:55...	1	SERC32	Sercos...	8042	CP2/use=0	Sercos phase switched
	493	We Apr/02/2014 08:32:54...	1	SERC32	Sercos...	8042	CP1/scan=2	Sercos phase switched
	492	We Apr/02/2014 08:32:53...	1	SERC32	Sercos...	8042	CP0	Sercos phase switched
	491	We Apr/02/2014 08:32:52...	1	LMCxx8	MyCon...	8002	V01.51.03.03	Controller boot started
	490	We Apr/02/2014 08:32:14...	1	OBJVER		8016		Controller reset
	489	We Apr/02/2014 08:31:52...	1	LMCxx8	MyCon...	8003		Controller boot finished
	488	We Apr/02/2014 08:31:51...	1	LMCxx8	MyCon...	8051		LMC 078/AX=8/RAM=500/n
	487	We Apr/02/2014 08:31:51...	1	SERC32	Sercos...	8042	CP4	Sercos phase switched
	486	We Apr/02/2014 08:31:51...	1	SERC32	Sercos...	8042	CP3/1 ms	Sercos phase switched
	485	We Apr/02/2014 08:31:50...	1	LMCxx8	MyCon...	8970	end	Fast Device Replacement
	484	We Apr/02/2014 08:31:50...	1	LMCxx8	MyCon...	8970	start	Fast Device Replacement
	483	We Apr/02/2014 08:31:50...	1	SERC32	Sercos...	8042	CP2/use=0	Sercos phase switched
	482	We Apr/02/2014 08:31:49...	1	SERC32	Sercos...	8042	CP1/scan=2	Sercos phase switched
	481	We Apr/02/2014 08:31:48...	1	SERC32	Sercos...	8042	CP0	Sercos phase switched
	480	We Apr/02/2014 08:31:47...	1	LMCxx8	MyCon...	8002	V01.51.03.03	Controller boot started

Some entries are highlighted in color. This helps you to find lines with comparable properties within the message logger. You can adjust it using the context menu. After saving or sending and then opening the file, the last selected color settings are retained.

Edit Comment

Select **Edit comment** from the context menu. You can add any comment for each entry. In this way, you can add additional information, which provides an overview of the detected errors in case service is required, even after a long time.

Copy

Select **Copy** from the context menu or press **Ctrl+C**. You can copy the contents of the selected line to the Windows Clipboard and then paste them into any text processing application.

Add Message Logger from Controller

Select **Add message logger from controller** from the context menu. The action creates a new message logger in the data view. A new continuous numbering with the current time stamp is created.

Reset Message Logger

Select **Reset message logger** from the context menu. This action locally creates an empty message logger (that does not contain any messages) in the data view and also deletes the message logger on the controller. All message logger entries of the controller are removed.

Save Message Logger to File

Select **Save message logger to file** from the context menu. A standard Windows dialog box opens. Enter a file name and save the selected message logger to any directory.

Load Message Logger from File

Select **Load message logger from file** from the context menu. A standard Windows dialog box opens. Use this dialog box to select the desired message logger which is saved in any directory. The action creates a new message logger in the data view. A continuous numbering without the current time stamp is created.

Import Message Logger from Controller

Select **Import message logger from controller** from the context menu. A standard Windows dialog box opens. Use this dialog box to select the desired message logger which is saved on the controller. The action creates a new message logger in the data view. A continuous numbering without the current time stamp is created.

Remove Message Logger from List

Select **Remove message logger from list** from the context menu or press the **Delete** key. This deletes the selected message logger from the list.

You can select several message loggers by pressing the **Ctrl** key.

Mark Lines

In the message logger, you can mark (highlight) lines which contain certain properties in common in different colors. For example, you can highlight all lines of **Type 2** in pink. Or you can highlight all lines with **Diag. code = 8002** in lime green.

The following context menu presents the configuration that is described in the preceding paragraph. The selection criteria are presented after the color highlighting.

Show help of diagnostics code	F1	MCx00C	LMC_P...	8002	V01
Edit comment		SERC3	SERCO...	8042	CP1
Copy	Ctrl+C	SERC3	SERCO...	8042	CP2
Add messagellogger from controller		SERC3	SERCO...	8042	CP3
Reset messagellogger		SERC3	SERCO...	8042	CP4
Save messagellogger to file		SERC3	SERCO...	8042	CP5
Load messagellogger from file		SERC3	SERCO...	8042	CP6
Import messagellogger from controller		SERC3	SERCO...	8042	CP7
Remove messagellogger from list	Del	SERC3	SERCO...	8042	CP8
Mark lines (No.=448)					
Use selected line for comparson	Ins	<ul style="list-style-type: none"> Type=2 Type=3 Type=4 Diag. Code=8002 			
Do not compare		<ul style="list-style-type: none"> unmark unmark all 			
Expand/Collapse list					
431	Mi 14.Jul 2010 12:58:30.876	1			
430	Mi 14.Jul 2010 12:58:29.256	1			
429	Mi 14.Jul 2010 12:58:27.682	1			
428	Mi 14.Jul 2010 12:58:26.000	1			
427	Mi 14.Jul 2010 12:51:42.276	1			
426	Do 08.Jul2010 12:51:37.500	1			

Select **Mark lines (xxx=yyy)** from the context menu. A submenu which already presents some color entries with their corresponding selection text opens. Here, as described in the example, with **Type=2** and **Diag.code=8002**.

You can define your own selection criteria. The selected text is determined by the last column selected before the context menu was called. Proceed as follows:

Add/Change a color entry:

Step	Action
1	Right-click the column with the desired selection criterion (for example, column header Diag. Code and cell value 8014). Result: The context menu opens.
2	Click Mark lines (Diag. code=8014) . Result: The submenu with the color selections opens.

Step	Action
3	Choose a highlighting color, for example, blue. Result: All lines with the selection criterion (Diag. code=8014) are highlighted in blue.

Delete a color entry:

Step	Action
1	Right-click the line with the color you want to remove in the message logger. Result: The context menu opens.
2	Click Mark lines . Result: The submenu with the color selections opens.
3	Click Unmark . Result: All line highlighting with this color is unmarked.

Delete all color entries:

Step	Action
1	Right-click the line with the color you want to remove in the message logger. Result: The context menu opens.
2	Click Mark lines . Result: The submenu with the color selections opens.
3	Click Unmark all . Result: All color markings are removed.

NOTE: It may happen that one line matches several valid color selection criteria. For example, the colors green and red could correspond to the **Diag.Code=8014**. In this case, the color used is the color that is listed lower in the context menu.

NOTE: If a color selection criterion has been applied to several colors, the marking with the highest priority is removed first when unmarking.

Use Selected Line for Comparison

Select **Use selected line for comparison** from the context menu or press the **Insert** key. Thus, you can highlight the selected line in **bold** in order to facilitate comparison with other lines. If you click a different message, the time difference to this line marked in bold is presented in the status bar (for example, **Time difference: 0.00:03:36.992 of 500(1) with regard to 481(1)**).

Do Not Compare

Select **Do not compare** from the context menu to reset a preliminary selected line for comparison again and to remove the bold marking.

Expand/Collapse List

By selecting **Expand/collapse list** from the context menu, all message loggers can alternatively be fully collapsed or expanded.

Diagnostic Messages

Diagnostic Message Classes

This table describes the diagnostic message classes:

Diagnostic class	Designation	Priority
4	Error detected resulting in complete stop	High
3	Error detected resulting in single stop (if the error is triggered by an axis)	-
2	Advisory	-
1	Message	Low
0	Deactivated	None

Diagnostic Messages

This table lists the diagnostic messages and their classes:

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8001	Diagnostic acknowledgement	1
8002	Controller boot started	1
8003	Controller boot finished	1
8004	Program started	1
8005	Program automatic start active	1
8006	Program stopped	1
8007	Controller login	1
8008	Controller logout	1
8009	Program reset	1
8010	write file	1
8013	Controller connect to TCP/IP server	1
8014	Controller disconnect from TCP/IP server	1
8015	filesystem <ide0:> repaired	1
8016	Controller reset	1
8017	CANopen emergency message reset	1
8018	CANopen node guarding error resolved	1
8019	CANopen node error info	1
8020	Program cycle check has changed	1
8021	Program cycle check values are changed	1

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8022	FC_SetTaskPriority() called	1
8023	Controller shutdown	1
8027	File write open	1
8028	File write close	1
8029	UPS OK	1
8030	UPS active -no power	1
8031	UPS power supply OK	1
8032	UPS begin saving retain area	1
8033	UPS retain area saved	1
8034	UPS program tasks terminated	1
8035	UPS active -system shutdown started	1
8036	UPS controller rebooting started	1
8037	Battery low	2
8038	NvRam/RTC power outage detected	2
8042	SERCOS phase switched	1
8043	SERCOS detect configuration	1
8044	SERCOS firmware download	1
8045	File write error detected	1
8046	FPGA firmware download	1
8047	PIC firmware download	1
8048	BT-4 firmware download	1
8051	Controller type	1
8052	SERCOS extended diagnostic (MASTER)	1
8053	UPS active overtemperature	1
8054	Controller temperature out of range	2
8055	Controller message HW monitor	1
8056	Controller power supply low	1
8057	Program online change	1
8059	UPS active -IEC-control task running	1
8060	UPS changing state	1
8100	Motor overload	3
8101	Power stage overtemperature	3
8102	Motor overtemperature	3
8104	Control voltage out of range	3

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8105	Encoder signal out of range	3
8106	DC bus controller communication not possible	3
8107	Overcurrent	3
8108	DC bus overvoltage	3
8109	DC bus undervoltage	3
8110	Phase missing	3
8111	Shutdown due to tracking deviation	3
8112	SERCOS telegram invalid	3
8113	Braking resistor error detected	3
8114	Device type plate not readable	3
8116	Commutation error detected	3
8117	Motor type plate not readable	3
8119	Power stage short-circuit /ground error detection	3
8120	Power stage overload	3
8121	Braking resistor - overtemperature	3
8122	Shutdown due to velocity limit	3
8123	Safe Torque Off incorrect	3
8125	Motor load high	2
8126	Power stage temperature high	2
8127	Motor temperature high	2
8129	Power stage load high	2
8130	Temperature of braking resistor high	2
8132	Tracking deviation limit exceeded	2
8133	Speed-dependent current reduction	2
8134	External 24 Vdc low	2
8135	DC bus voltage low	2
8136	Safe Torque Off active	2
8137	Motorless	3
8138	Motor/Drive combination not supported	3
8139	DC bus precharge not possible	3
8140	Motor stop time limit exceeded	3
8142	Control board overtemperature	3
8143	Encoder temperature high	2
8144	DC bus short-circuit or ground error	3

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8146	DC bus overload	3
8153	DC bus discharge not possible	3
8154	Phase L1 missing	2
8155	Phase L2 missing	2
8156	Phase L3 missing	2
8157	DC bus load high	2
8159	DC bus discharge delayed	2
8161	Control board temperature high	2
8163	SERCOS Slave C1D error detected	3
8164	SERCOS C1D man.-specific error detected	3
8165	SERCOS Slave C2D advisory detected	2
8166	SERCOS C2Dman.specific advisory detected	2
8169	SERCOS Slave communication disturbance detected	2
8170	Encoder position not accessible	3
8171	Encoder communication disturbance detected	2
8172	Encoder extended diagnostic error detected	1
8173	Encoder error (track monitoring) detected	1
8177	Power board overtemperature	3
8178	Device internal error detected	3
8179	Braking resistor load high	2
8180	Power board temperature high	2
8181	Fan error detected	2
8182	External 24 Vdc power supply high	1
8183	Device fallback firmware active	3
8184	HW/SW combination not supported	3
8185	Device error detected	3
8186	DC bus voltage high	2
8204	Program cannot be loaded	3
8205	Impermissible parameter value	3
8209	Last boot unsuccessful	3
8300	Program divide by zero	3
8301	coprocessor segment overflow	3
8302	stack error detected	3
8303	general protection error detected	3

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8304	coprocessor error detected	3
8305	memory limit exceeded	3
8306	arithmetic overflow	3
8307	double execution error detected	3
8308	invalid task state segment	3
8309	no memory segment	3
8310	invalid memory segment adjustment	3
8311	coprocessor division error detected	3
8312	Parameter relocation unsuccessful	3
8313	excessive cycle time overrun	3
8316	NvRam data not valid	2
8317	Program cycle time overrun	2
8318	Program calculated profile deleted	3
8320	incorrect array access	3
8321	division by zero	3
8322	exception in IEC task	3
8323	string too long	3
8324	UPS error detected	3
8325	File corrupt	3
8326	Program function not supported	3
8327	CamTrack invalid Position Source	2
8328	CamTrack invalid Destination	2
8329	CamTrack invalid Bit number	2
8330	Program master job not executable	3
8331	Licensing	3
8332	Licensing	3
8333	EncoderNet receiving data not possible	3
8334	EncoderNet receiving data dist.detected	2
8335	EncoderNet synchronization not possible	3
8336	EncoderNet synchronization disturbance detected	2
8337	Parameter DynIECData value too high	3
8338	UPS battery not charged	3
8339	UPS active -system temperature high	3
8340	Data/parameter out of range	3

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8341	CamTrack invalid Position type	2
8400	Program diagnostic message class 0	0
8401	Program diagnostic message class 1	1
8402	Program diagnostic message class 2	2
8403	Program diagnostic message class 3	3
8404	Program diagnostic message class 4	4
8406	IEC diagnostic message class 1	1
8501	SERCOS slave not found	3
8502	SERCOS loop not closed	1
8503	SERCOS service channel error detected	3
8504	SERCOS read cycle overflow	3
8505	SERCOS Master communication disturbance detected	2
8506	SERCOS Master communication not possible	3
8507	SERCOS write cycle overflow	3
8508	SERCOS run-up not possible	3
8509	SERCOS slave SW not supported	3
8510	SERCOS Interrupt lost	3
8511	CPU time overflow	3
8512	SERCOS incorrect device type	3
8517	SERCOS addressing not unique	3
8518	SERCOS too many real slaves	3
8600	Master Encoder communication not possible	3
8601	Master Encoder signal out of range	3
8610	Async FB error	3
8611	Copied Async FB. Use a reference	3
8612	Async FB declared as retain/persistent	3
8613	Async job timeout error	3
8700	CAN layer 2 driver error detected	3
8701	CAN layer2 initialization error detected	3
8702	CAN layer2 single error detected	3
8703	CAN layer2 errors reach advisory limit	3
8704	CAN layer2 switched passive	3
8705	CAN layer 2 system error detected	3
8706	CAN layer2 errors below advisory limit	1

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8707	CAN layer2 switched active	1
8710	communication error detected	3
8720	no module found	3
8722	no cyclic telegram	3
8723	no PROFIBUS config data	3
8725	firmware of the module was replaced	1
8726	firmware of the module is incorrect	3
8730	incorrect master parameter data	3
8131	automatic bus deactivation	3
8732	slave not responding	3
8733	unrecoverable bus error detected	3
8734	Bus short circuit detected	3
8735	reject bus telegrams	3
8736	no I/O data exchange with slave	3
8737	double IEC address assigned	3
8738	Configuration I/O data > permissible I/O area	3
8739	double PROFIBUS address assigned	3
8750	CANopen node does not exist	3
8751	CANopen node not configured	1
8752	no CANopen EDS file exists	3
8753	initialisation CANopen module unsuccessful	3
8754	CANopen Emergency Message	3
8755	CANopen node guarding error detected	3
8756	CANopen DPM access timeout	3
8757	CANopen configuration error detected	3
8758	Application object Size not supported	3
8759	Application object maximum count limit reached	3
8780	Encoder output frequency > 1MHz	3
8781	Master Encoder no connection	3
8782	Master Encoder signal out of range	3
8785	Hardware component error detected	3
8786	Asynchronous to SERCOS bus	3
8787	configuration error detected	3
8788	Wiring error detected	2

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8789	PacNet communication disturbance detected	2
8790	Module error detected	3
8791	TM5/TM7 module error detected on TM5NS31	3
8800	Insufficient working memory	3
8826	PIC update not possible	4
8827	Controller power-off/hardware monitor	3
8828	Library error detected	1
8903	Software error detected (class 3)	1
8904	Software error detected (class 4)	4
8905	FC_UserRefGeneratorStart not possible	4
8906	ControlMode invalid	3
8907	Encoder interface invalid	3
8908	Unintended motor reaction detected	3
8909	Motor type plate parameter invalid	3
8910	Reference value invalid	3
8910	Name too long	3
8957	SERCOS bus topology changed	2
8958	Encoder communication not possible	3
8959	Mains contactor error detected	3
8960	Invalid mains voltage mode setting	2
8961	Phase missing	2
8963	NRT IPAddr not in IPAddressRangeStatic	2
8964	NRT IPAddressRangeDynamic is insufficient	2
8965	NRT IP parameter read not possible	2
8966	NRT IP parameter write not possible	2
8967	NRT IP parameter device different	2
8968	NRT network overlapping detected	2
8969	Motor cable not connected	3
8970	Fast Device Replacement	1
8971	Fast Device Replacement not successful	2
8972	NRT gateway not in network	2
8973	Program download	1
8974	Brake voltage too low	3
8975	Motor commutation invalid	2

Diagnostic code (DiagCode)	Diagnostic message (DiagMsg)	Diagnostic class (DiagClass)
8976	Mains phases wiring not correct	3
8977	Motor temp. monitoring disabled	2
8978	InverterEnableConfig invalid	3
8979	STO_A and STO_B different levels	3
8980	Braking resistor not connected	3
8981	Bootloader update	1
8982	Device state	1
8983	DC bus precharge active	1
8984	DC bus precharge complete	1
8990	Firmware update not possible	1
8991	Data transfer invalid	1
8992	Braking resistor short circuit	3
8993	Last device on SERCOS port	1
8994	Invalid ProducerCycleTime	3
8995	Update motor type plate	1
8996	Update motor type plate not successful	3
8997	Motor identification invalid	3
8998	TM5/TM7 supply voltage low	3
8999	TM5/TM7 supply voltage advisory	1

Appendix C

LMC078 Sercos3 Library

Introduction

This chapter describes the LMC078 Sercos3 library.

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
C.1	Data Types	272
C.2	Sercos Functions	278
C.3	Asynchronous Sercos Function Blocks	291

Section C.1

Data Types

Overview

This section describes the data types included in the LMC078 Sercos3 library.

What Is in This Section?

This section contains the following topics:

Topic	Page
ST_SercosConfiguration Data Type	273
ST_SercosConfigurationDevice Data Type	274
ET_Sercos3CmdType Data Type	276
ET_Sercos3IDNType Data Type	277

ST_SercosConfiguration Data Type

Introduction

The ST_SercosConfiguration structure stores the general values of the Sercos bus.

This data type is used with the following functions:

- FC_SercosGetConfiguration
- FC_SercosScanConfiguration

Data Type Structure

This table describes the content of the ST_SercosConfiguration structure:

Enumeration	Type	Description
uiNumberOfEntries	UINT	Number of Sercos devices scanned on the Sercos bus.
uiNumberOfPhysicalDevices	UINT	Number of Sercos devices found on the Sercos bus and are configured in the SoMachine project.
uiPhaseRunUpCount	UINT	Counts the Sercos phase up runs.
iCurrentPhase	INT	Current phase which the Sercos bus is running.
astDevices	ARRAY [0...254] of ST_SercosConfigurationDevice	Array with specific description of Sercos devices found on the Sercos bus.

ST_SercosConfigurationDevice Data Type

Introduction

The ST_SercosConfigurationDevice structure stores the specific description of a Sercos device found on the Sercos bus.

This data type is used by the ST_SercosConfiguration data type.

This data type is used with the following functions:

- FC_SercosGetConfiguration
- FC_SercosScanConfiguration

Data Type Structure

This table describes the content of the ST_SercosConfigurationDevice structure:

Enumeration	Type	Description
stLogicalAddress	ST_LogicalAddress	Logical address of the Sercos slave.
uiVendorCode	UINT	Manufacturer code of the device.
sVendorDeviceId	STRING(40)	Sercos device identifier.
sName	STRING(80)	Name given to the device in the controller configuration.
sPowerSupply	STRING(40)	Currently assigned power supply.
udiWorkingMode	UDINT	Work mode of the device.
udiWorkingState	UDINT	Currently active mode.
udiIdentificationMode	UDINT	Identification mode of device.
uiTopologyAddress	UINT	Physical position of the device in the Sercos ring.
uiConfiguredTopologyAddress	UINT	Topological address found at the Sercos bus, which the device is to be assigned to.
uiSercosAddress	UINT	Sercos address to which the device has been assigned. The Sercos address is automatically assigned by the master during start-up of the Sercos bus if the device is not in the IdentificationMode SercosAddress
uiConfiguredSercosAddress	UINT	Configured Sercos address.
sSerialNumberController	STRING(80)	Serial number of a motor controller at the Sercos bus.
sConfiguredSerialNumberController	STRING(80)	Serial number of a motor controller found at the Sercos bus, which the drive is to be assigned to.
sSerialNumberMotor	STRING(40)	Serial number of the servo motor
sConfiguredSerialNumberMotor	STRING(40)	Configured servo motor serial number.

Enumeration	Type	Description
sApplicationType	STRING(40)	Application type.
sConfiguredApplicationType	STRING(40)	Configured application type.
sControllerType	STRING(40)	Device type.
sMotorType	STRING(40)	Name of the motor.
sFW_Version	STRING(40)	Firmware version of the device. If the device in the controller configuration is virtual (non-real), then the parameter value is <LMC version>.virtual.
sHW_Version	STRING(40)	Hardware version of the device If the device in the controller configuration is virtual (non-real), then the parameter value is <LMC version>.virtual.
sBootloaderVersion	STRING(40)	Version name of the device bootloader.
sFPGA_Version	STRING(40)	FPGA version of the device.
sIPAddress	STRING(15)	Defined IP address of the device.
sSubnetmask	STRING(15)	Defined subnet mask of the device.
sGateway	STRING(15)	Defined gateway of the device.
sMACAddress	STRING(17)	MAC address of the device.

ET_Sercos3CmdType Data Type

Introduction

The ET_Sercos3CmdType data type is used with the following function blocks to set Sercos commands:

- FB_SercosReadServiceDataAsync
- FB_SercosWriteServiceDataAsync

Read and write commands use the entries 0...6 while procedure commands use entries 7...10.

Data Type Structure

This table describes the content of the ET_Sercos3CmdType enumeration:

Enumeration	Type	Initial value	Description
ReadWriteName	WORD	0	Read or write a name.
ReadWriteAttribute	WORD	1	Read or write an attribute.
ReadWriteUnit	WORD	2	Read or write a unit.
ReadWriteMinValue	WORD	3	Read or write a minimal value.
ReadWriteMaxValue	WORD	4	Read or write a maximal value.
ReadWriteUserData	WORD	5	Read or write user data.
ReadWriteVarLength	WORD	6	Read or write a variable length.
ExecuteCommand	WORD	7	Execute a command.
ExecuteCommandStart	WORD	8	Start the execution of a command.
ExecuteCommandCheck	WORD	9	Verify the state of executed command.
ExecuteCommandStop	WORD	10	Stop the execution of a command.

ET_Sercos3IDNType Data Type

Introduction

The ET_Sercos3IDNType data type is used with the following function blocks to set the Sercos IDN type (standard or proprietary):

- FB_SercosProcedureCommandAsync
- FB_SercosReadServiceDataAsync
- FB_SercosWriteServiceDataAsync

Data Type Structure

This table describes the content of the ET_Sercos3IDNType enumeration:

Enumeration	Type	Initial value	Description
ParameterType_P	WORD	1	Proprietary IDN (P-x-xxxx.x.x).
ParameterType_S	WORD	2	Standard IDN (S-x-xxxx.x.x).

Section C.2

Sercos Functions

Overview

This section describes the Sercos functions.

What Is in This Section?

This section contains the following topics:

Topic	Page
FC_SercosGetConfiguration Function	279
FC_SercosReadServiceData Function	280
FC_SercosReadServiceDataByTopAddr Function	283
FC_SercosScanConfiguration Function	285
FC_SercosWriteServiceData Function	287
FC_SercosWriteServiceDataByTopAddr Function	289

FC_SercosGetConfiguration Function

Function Description

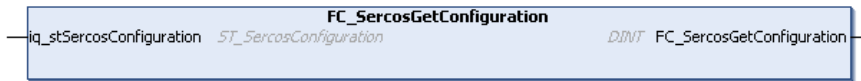
The `FC_SercosGetConfiguration` function is used to obtain a list of all devices, even not configured, which are connected to the Sercos bus.

NOTE: The Sercos bus must have at least one time reached phase 2.

The function returns the Sercos devices which are determined at the end of the change to phase 2.

The scanned devices are returned in a structure. If the Sercos bus has never reached phase 2 then the function returns an error.

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variable:

Input	Type	Comment
<code>iq_stSercosConfiguration</code>	<code>ST_SercosConfiguration</code>	Structure which gets configuration on Sercos bus.

This table describes the output variable:

Output	Type	Comment
<code>FC_SercosGetConfiguration</code>	<code>DINT</code>	See the return value description table below.

This table describes the return value:

Value	Description
0	The function is successfully executed.
-1	Error detected.
-2	Sercos bus has never reached phase 2.

FC_SercosReadServiceData Function

Function Description

The `FC_SercosReadServiceData` function reads service data via Sercos that are used for debugging.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Use this function only after consulting the Schneider Electric application department.

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

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
i_stLogAddr	ST_LogicalAddress	Logical address of the Sercos device.
i_dwIDN	DWORD	Data service IDN.
i_wType	WORD	Data service type: <ul style="list-style-type: none"> ● 0: Read name ● 1: Read attribute ● 2: Read unit ● 3: Read minimum value ● 4: Read maximum value ● 5: Read user data ● 6: Read user data with variable length ● 7: Execute command
i_dwPointerToData	DWORD	Pointer on the data memory, filled with the actual length of the data bytes.
i_wMaxDataLen	WORD	Actual length of the data bytes to read.

This table describes the input/output variables:

Input	Type	Comment
iq_uiReadDataLen	UINT	Variable for reading out the data length of the Sercos parameter (uwType=6).
iq_uiMaxReadDataLen	UINT	Variable for reading out the maximum data length of the Sercos parameter (uwType=6).

This table describes the output variable:

Output	Type	Comment
FC_SercosReadServiceData	DINT	See the return value description table below.

This table describes the return value:

Value	Description
0	The function is successfully executed.
-1	The logical address is invalid.
-431	Error detected during service request (for example, timeout).
-445	Service timeout.
-461	In the current phase, reading parameters via the service channel is not supported.
-462	The addressed device does not support the <code>ServiceDataRead</code> function.
-464	Invalid service transfer.

FC_SercosReadServiceDataByTopAddr Function

Function Description

The FC_SercosReadServiceDataByTopAddr function reads service data via the Sercos service channel from the addressed device.

WARNING

UNINTENDED EQUIPMENT OPERATION

Use this function only after consulting the Schneider Electric application department.

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

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
i_wTopoAddr	WORD	Topological address of the Sercos device.
i_dwIDN	DWORD	Data service IDN.

Input	Type	Comment
i_wType	WORD	Data service type: <ul style="list-style-type: none"> ● 0: Read name ● 1: Read attribute ● 2: Read unit ● 3: Read minimum value ● 4: Read maximum value ● 5: Read user data ● 6: Read user data with variable length ● 7: Execute command
i_dwPointerToData	DWORD	Pointer to the data memory, filled with the actual length of the data bytes.
i_wMaxDataLen	WORD	Actual length of the data bytes to read.

This table describes the input/output variables:

Input	Type	Comment
iq_uiReadDataLen	UINT	Variable for reading out the data length of the Sercos parameter (<code>uwType=6</code>).
iq_uiMaxReadDataLen	UINT	Variable for reading out the maximum data length of the Sercos parameter (<code>uwType=6</code>).

This table describes the output variable:

Output	Type	Comment
FC_SercosReadServiceDataByTopAddr	DINT	See the return value description table below.

This table describes the return value:

Value	Description
0	The function is successfully executed.
-1	The topological address is invalid.
-431	Error detected during service request (for example, timeout).
-445	Service timeout.
-461	In the current phase, reading parameters via the service channel is not supported.
-462	The addressed device does not support the <code>ServiceDataRead</code> function.
-464	Invalid service transfer.

FC_SercosScanConfiguration Function

Function Description

The `FC_SercosScanConfiguration` function is used to obtain a list of all devices, even if not configured, which are connected to the Sercos bus.

The Sercos bus is scanned by this function automatically.

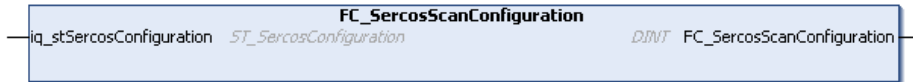
If the Sercos bus is in a phase below 2 then the function switches to phase 2.

In phase 2 the Sercos bus is scanned. When the Sercos bus is scanned it is put back to the initial phase.

If the Sercos bus is in phase 2 or greater, the scanned devices are returned.

The scanned devices are returned in a structure.

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input/output variable:

Input	Type	Comment
<code>iq_stSercosConfiguration</code>	<code>ST_SercosConfiguration</code>	Structure which gets configuration on Sercos bus.

This table describes the output variable:

Output	Type	Comment
<code>FC_SercosScanConfiguration</code>	DINT	See the return value description table below.

This table describes the return value:

Value	Description
0	The function is successfully executed.
-1	Error detected.
-2	The Sercos bus cannot reach phase 2.

FC_SercosWriteServiceData Function

Function Description

The FC_SercosWriteServiceData function writes service data via Sercos that are used for debugging.

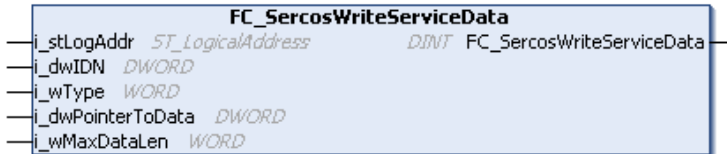
⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Use this function only after consulting the Schneider Electric application department.

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

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
i_stLogAddr	ST_LogicalAddress	Logical address of the Sercos device.
i_dwIDN	DWORD	Data service IDN.
i_wType	WORD	Data service type: <ul style="list-style-type: none"> ● 0: Write name ● 1: Write attribute ● 2: Write unit ● 3: Write minimum value ● 4: Write maximum value ● 5: Write user data ● 6: Write user data with variable length ● 7: Execute command

Input	Type	Comment
i_dwPointerToData	DWORD	Pointer on the data memory, filled with the actual length of the data bytes.
i_wMaxDataLen	WORD	Actual length of the data bytes to write.

This table describes the output variable:

Output	Type	Comment
FC_SercosWriteServiceData	DINT	See the return value description table below.

This table describes the return value:

Value	Description
0	The function is successfully executed.
-1	The logical address is invalid.
-431	Error detected during service request (for example, timeout).
-445	Service timeout.
-461	In the current phase, writing parameters via the service channel is not supported.
-462	The addressed device does not support the <code>ServiceDataWrite</code> function.
-464	Invalid service transfer.

FC_SercosWriteServiceDataByTopAddr Function

Function Description

The FC_SercosWriteServiceDataByTopAddr function writes service data via the Sercos service channel into the addressed device.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Use this function only after consulting the Schneider Electric application department.

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

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
i_wTopoAddr	WORD	Topological address of the Sercos device.
i_dwIDN	DWORD	Data service IDN.
i_wType	WORD	Data service type: <ul style="list-style-type: none"> ● 0: Write name ● 1: Write attribute ● 2: Write unit ● 3: Write minimum value ● 4: Write maximum value ● 5: Write user data ● 6: Write user data with variable length ● 7: Execute command

Input	Type	Comment
i_dwPointerToData	DWORD	Pointer to the data memory, filled with the actual length of the data bytes.
i_wMaxDataLen	WORD	Actual length of the data bytes to write.

This table describes the output variable:

Output	Type	Comment
FC_SercosWriteServiceDataByTopAddr	DINT	See the return value description table below.

This table describes the return value:

Value	Description
0	The function is successfully executed.
-1	The topological address is invalid.
-431	Error detected during service request (for example, timeout).
-445	Service timeout.
-461	In the current phase, writing parameters via the service channel is not supported.
-462	The addressed device does not support the <code>ServiceDataWrite</code> function.
-464	Invalid service transfer.

Section C.3

Asynchronous Sercos Function Blocks

Overview

This section describes the asynchronous Sercos function blocks.

These function blocks are used for acyclic communication on Sercos service channel.

They are used for asynchronously reading/writing device parameters and sending commands via the Sercos interface.

All IDN of the LXM32S drive are accessible through these function blocks.

What Is in This Section?

This section contains the following topics:

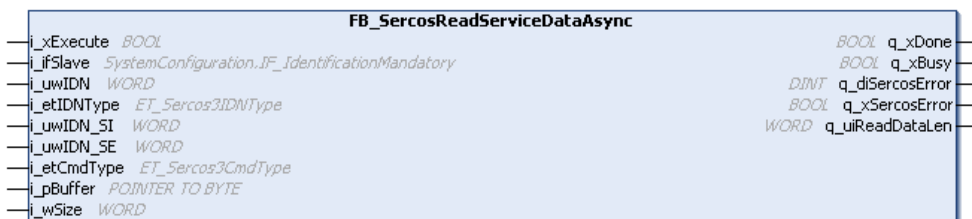
Topic	Page
FB_SercosReadServiceDataAsync : Read Data Asynchronously via theSercos Interface	292
FB_SercosWriteServiceDataAsync: Write Data Asynchronously via theSercos Interface	294
FB_SercosProcedureCommandAsync: Send Commands Asynchronously via the Sercos interface	296

FB_SercosReadServiceDataAsync : Read Data Asynchronously via theSercos Interface

Function Block Description

The FB_SercosReadServiceDataAsync function block reads data asynchronously via the Sercos interface.

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
i_xExecute	BOOL	If TRUE, starts the function block execution.
i_ifSlave	SystemConfiguration.IF_IdentificationMandatory	Interface that describes the device. It contains the name of the device and its logical number.
i_uwIDN	WORD	IDN number of the device. Example for IDN S-0-1027.0.1: i_uwIDN = 1027
i_etIDNType	ET_Sercos3IDNType (see page 277)	IDN type (standard or proprietary). Example for IDN S-0-1027.0.1: i_etIDNType = ET_Sercos3IDNType.ParameterType_S;
i_uwIDN_SI	WORD	IDN structure instance. Example for IDN S-0-1027.0.1: i_uwIDN_SI = 0
i_uwIDN_SE	WORD	IDN structure element. Example for IDN S-0-1027.0.1: i_uwIDN_SE = 1

Input	Type	Comment
i_etCmdType	ET_Sercos3CmdType (<i>see page 276</i>)	Type of command to execute. The command supports command types 0..6.
i_pBuffer	POINTER TO BYTE	Pointer to a buffer of length i_szSize.
i_wSize	WORD	Length of the buffer.

This table describes the output variables:

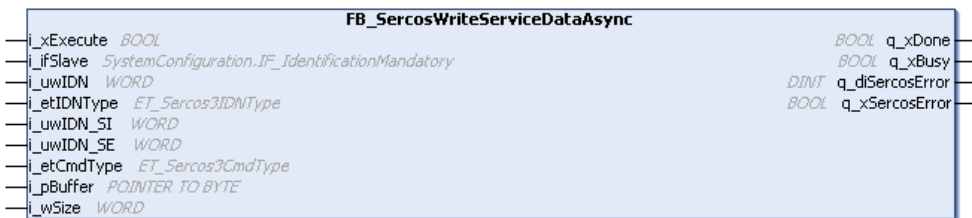
Output	Type	Comment
q_xDone	BOOL	If TRUE, indicates that the function block execution is finished with no error detected.
q_xBusy	BOOL	If TRUE, indicates that the function block execution is in progress.
q_diSercosError (<i>see page 217</i>)	DINT	Sercos error (<i>see page 217</i>) value that is returned by the synchronous read command.
q_xSercosError	BOOL	If TRUE, indicates that a Sercos error has been detected (negative value).
q_uiReadDataLen	WORD	Number of data items read.

FB_SercosWriteServiceDataAsync: Write Data Asynchronously via the Sercos Interface

Function Block Description

The `FB_SercosWriteServiceDataAsync` function block writes data asynchronously via the Sercos interface.

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
<code>i_xExecute</code>	BOOL	If TRUE, starts the function block execution.
<code>i_ifSlave</code>	SystemConfiguration.IF_IdentificationMandatory	Interface that describes the device. It contains the name of the device and its logical number.
<code>i_uwIDN</code>	WORD	IDN number of the device. Example for IDN S-0-1027.0.1: <code>i_uwIDN = 1027</code>
<code>i_etIDNType</code>	ET_Sercos3IDNType (see page 277)	IDN type (standard or proprietary). Example for IDN S-0-1027.0.1: <code>i_etIDNType =</code> <code>ET_Sercos3IDNType.ParameterType_S;</code>
<code>i_uwIDN_SI</code>	WORD	IDN structure instance. Example for IDN S-0-1027.0.1: <code>i_uwIDN_SI = 0</code>
<code>i_uwIDN_SE</code>	WORD	IDN structure element. Example for IDN S-0-1027.0.1: <code>i_uwIDN_SE = 1</code>

Input	Type	Comment
i_etCmdType	ET_Sercos3CmdType (<i>see page 276</i>)	Type of command to execute. The command supports command types 0..6.
i_pBuffer	POINTER TO BYTE	Pointer to a buffer of length i_szSize.
i_wSize	WORD	Length of the buffer.

This table describes the output variables:

Output	Type	Comment
q_xDone	BOOL	If TRUE, indicates that the function block execution is finished with no error.
q_xBusy	BOOL	If TRUE, indicates that the function block execution is in progress.
q_diSercosError (<i>see page 217</i>)	DINT	Sercos error (<i>see page 217</i>) value that is returned by the synchronous read command.
q_xSercosError	BOOL	If TRUE, indicates that a Sercos error has been detected (negative value).

FB_SercosProcedureCommandAsync: Send Commands Asynchronously via the Sercos interface

Function Block Description

The FB_SercosProcedureCommandAsync function block sends commands asynchronously via the Sercos interface.

Graphical Representation



IL and ST Representation

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

I/O Variable Description

This table describes the input variables:

Input	Type	Comment
i_xExecute	BOOL	If TRUE, starts the function block execution.
i_xAbort	BOOL	If TRUE, execution of the command is aborted. If FALSE, execution of the command as planned.
i_ifSlave	SystemConfiguration.IF_IdentificationMandatory	Interface that describes the device. It contains the name of the device and its logical number.
i_uwIDN	WORD	IDN number of the device. Example for IDN S-0-1027.0.1: i_uwIDN = 1027
i_etIDNType	ET_Sercos3IDNType (see page 277)	IDN type (standard or proprietary). Example for IDN S-0-1027.0.1: i_etIDNType = ET_Sercos3IDNType.ParameterType_S;

Input	Type	Comment
i_uwIDN_SI	WORD	IDN structure instance. Example for IDN S-0-1027.0.1: i_uwIDN_SI = 0
i_uwIDN_SE	WORD	IDN structure element. Example for IDN S-0-1027.0.1: i_uwIDN_SE = 1

This table describes the output variables:

Output	Type	Comment
q_xDone	BOOL	If TRUE, indicates that the function block execution is finished with no error.
q_xBusy	BOOL	If TRUE, indicates that the function block execution is in progress.
q_diSercosError (<i>see page 217</i>)	DINT	Sercos error (<i>see page 217</i>) value that is returned by the synchronous read command.
q_xSercosError	BOOL	If TRUE, indicates that a Sercos error has been detected (negative value).
q_uiProcCmdError	UINT	Procedure command error value that is returned by the synchronous read command.
q_xProcCmdError	BOOL	If TRUE, indicates that a procedure command error has been detected.

Appendix D

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:

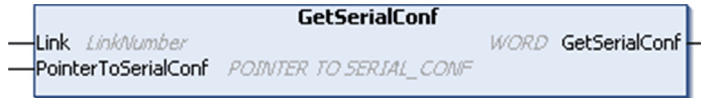
Topic	Page
GetSerialConf: Get the Serial Line Configuration	300
SetSerialConf: Change the Serial Line Configuration	301
SERIAL_CONF: Structure of the Serial Line Configuration Data Type	303

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 <i>(see SoMachine, Modbus and ASCII Read/Write Functions, PLCCommunication Library Guide)</i>	Link is the communication port number.
PointerToSerialConf	POINTER TO SERIAL_CONF <i>(see page 303)</i>	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: <ul style="list-style-type: none"> ● 0: The configuration parameters are returned ● 255: The configuration parameters are not returned because: <ul style="list-style-type: none"> ○ the function was not successful ○ the function is in progress

Example

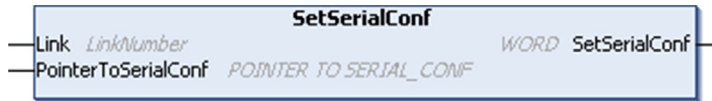
Refer to the SetSerialConf *(see page 302)* example.

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 303)	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: <ul style="list-style-type: none"> ● 0: The new configuration is set ● 255: The new configuration is refused because: <ul style="list-style-type: none"> ○ 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 (polarization 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 E

Controller Performance

Processing Performance

Introduction

This chapter provides information about the LMC078 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	1 μ s
Addition/subtraction/multiplication of DINT	1 μ s
Addition/subtraction/multiplication of REAL	3 μ s
Division of REAL	48 μ s
Operation on BOOLEAN, for example, Status:= Status and value	2 μ s
LD INT + ST INT	1 μ s
LD DINT + ST DINT	1 μ s
LD REAL + ST REAL	9 μ s

Communication and System Processing Time

The communication processing time varies, depending on the number of sent/received requests.

Response Time on Event

The response time presented in the following table represents the time between a signal rising edge on an input triggering an external task and the edge of an output set by this task. The event task also process 100 IL instructions before setting the output:

Minimum	Typical	Maximum
120 μ s	126 μ s	140 μ s



!

%IW

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

%QW

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

A

application

A program including configuration data, symbols, and documentation.

ARP

(*address resolution protocol*) An IP network layer protocol for Ethernet that maps an IP address to a MAC (hardware) address.

AT

(*acknowledge telegram*) On Sercos bus, data are sent by the slaves to the master through AT telegrams (feedback values).

B

BOOL

(*boolean*) A basic data type in computing. A `BOOL` variable can have one of these values: 0 (`FALSE`), 1 (`TRUE`). A bit that is extracted from a word is of type `BOOL`; for example, `%MW10.4` is a fifth bit of memory word number 10.

Boot application

(*boot application*) The binary file that contains the application. Usually, it is stored in the controller and allows the controller to boot on the application that the user has generated.

BOOTP

(*bootstrap protocol*) A UDP network protocol that can be used by a network client to automatically obtain an IP address (and possibly other data) from a server. The client identifies itself to the server using the client MAC address. The server, which maintains a pre-configured table of client device MAC addresses and associated IP addresses, sends the client its pre-configured IP address. BOOTP was originally used as a method that enabled diskless hosts to be remotely booted over a network. The BOOTP process assigns an infinite lease of an IP address. The BOOTP service utilizes UDP ports 67 and 68.

C

CANopen

An open industry-standard communication protocol and device profile specification (EN 50325-4).

CFC

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

CIP

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

configuration

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

continuous function chart language

A graphical programming language (an extension of the IEC61131-3 standard) based on the function block diagram language that works like a flowchart. However, no networks are used and free positioning of graphic elements is possible, which allows feedback loops. For each block, the inputs are on the left and the outputs on the right. You can link the block outputs to inputs of other blocks to create complex expressions.

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.

D**DHCP**

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

DINT

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

DWORD

(double word) Encoded in 32-bit format.

E**EDS**

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

Ethernet

A physical and data link layer technology for LANs, also known as IEEE 802.3.

EtherNet/IP

(Ethernet industrial protocol) An open communications protocol for manufacturing automation solutions in industrial systems. EtherNet/IP is in a family of networks that implement the common industrial protocol at its upper layers. The supporting organization (ODVA) specifies EtherNet/IP to accomplish global adaptability and media independence.

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.

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.

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.

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 block

A programming unit that has 1 or more inputs and returns 1 or more outputs. FBs are called through an instance (function block copy with dedicated name and variables) and each instance has a persistent state (outputs and internal variables) from 1 call to the other.

Examples: timers, counters

H

health bit

Variable that indicates the communication state of the channels.

I

I/O

(input/output)

ICMP

(Internet control message protocol) Reports errors detected and provides information related to datagram processing.

IEC

(international electrotechnical commission) A non-profit and non-governmental international standards organization that prepares and publishes international standards for electrical, electronic, and related technologies.

IEC 61131-3

Part 3 of a 3-part IEC standard for industrial automation equipment. IEC 61131-3 is concerned with controller programming languages and defines 2 graphical and 2 textual programming language standards. The graphical programming languages are ladder diagram and function block diagram. The textual programming languages include structured text and instruction list.

IL

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

Input Assembly

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

INT

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

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**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`).

LRC

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

LWORD

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

M**MAC address**

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

MAST

A processor task that is run through its programming software. The MAST task has 2 sections:

- **IN:** Inputs are copied to the IN section before execution of the MAST task.
- **OUT:** Outputs are copied to the OUT section after execution of the MAST task.

MDT

(*master data telegram*) On Sercos bus, an MDT telegram is sent by the master once during each transmission cycle to transmit data (command values) to the servo drives (slaves).

ms

(*millisecond*)

MST

(*master synchronization telegram*) On Sercos bus, an MST telegram is broadcast by the master at the beginning of each transmission cycle to synchronize the timing of the cycle.

N

network

A system of interconnected devices that share a common data path and protocol for communications.

node

An addressable device on a communication network.

O

originator

In EtherNet/IP explicit messaging, the device, usually the logic controller, that initiates data exchanges with target network devices.

See also *target*

OS

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

Output Assembly

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

P

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).

persistent data

Value of persistent data is used at next application change or cold start. Only get re-initialized at a reboot of the controller or reset origin. Especially, they maintain their values after a download.

POU

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

Profibus DP

(Profibus decentralized peripheral) An open bus system uses an electrical network based on a shielded 2-wire line or an optical network based on a fiber-optic cable. DP transmission allows for high-speed, cyclic exchange of data between the controller CPU and the distributed I/O devices.

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.

R

REAL

A data type that is defined as a floating-point number encoded in a 32-bit format.

retained data

A value used in the next power-on or warm start. The value is retained because of a power outage shutdown of the controller or a normal requested shutdown of the controller.

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.

RPI

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

RTC

(real-time clock) A battery-backed time-of-day and calendar clock that operates continuously, even when the controller is not powered for the life of the battery.

RTP

(real-time process) The real-time process is the most important system task. It is responsible for executing all real-time tasks at the correct time. Real-time processing is triggered by the Sercos real-time bus cycle.

S

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).

Sercos

(*serial real-time communications system*) A digital control bus that interconnects, motion controls, drives, I/Os, sensors, and actuators for numerically controlled machines and systems. It is a standardized and open controller-to-intelligent digital device interface, designed for high-speed serial communication of standardized closed-loop real-time data.

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.

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.

string

A variable that is a series of ASCII characters.

system variable

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

T

target

In EtherNet/IP explicit messaging, the device, that responds to data exchange requests sent by originator devices.

See also *originator*

task

A group of sections and subroutines, executed cyclically or periodically for the MAST task or periodically for the FAST task.

A task possesses a level of priority and is linked to inputs and outputs of the controller. These I/O are refreshed in relation to the task.

A controller can have several tasks.

TCP

(transmission control protocol) A connection-based transport layer protocol that provides a simultaneous bi-directional transmission of data. TCP is part of the TCP/IP protocol suite.

touchprobe input

Touchprobe inputs are advanced digital inputs. These inputs are used for measuring functions, which accurately detect positions relative to a measure input. Once a touchprobe function has been activated, it runs independently in the system, independent of the IEC program. The IEC program can use parameters to detect the state of the measuring function. This function is supported by hardware and software.

TPDO

(transmit process data object) An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

U**UDINT**

(unsigned double integer) Encoded in 32 bits.

UDP

(user datagram protocol) A connectionless mode protocol (defined by IETF RFC 768) in which messages are delivered in a datagram (data telegram) to a destination computer on an IP network. The UDP protocol is typically bundled with the Internet protocol. UDP/IP messages do not expect a response, and are therefore ideal for applications in which dropped packets do not require retransmission (such as streaming video and networks that demand real-time performance).

UINT

(unsigned integer) Encoded in 16 bits.

V**variable**

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

W**WORD**

A type encoded in a 16-bit format.

Z

zero track

The track of an incremental encoder which serves as the reference point and enables reinitialization at each revolution, also known as top 0 or top Z.



A

ASCII Manager, *224*

C

CANopen interface, *194*
changeIPAddress, *251*
 changing the controller IP address, *251*
configuration parameters, *76*
Controller Configuration
 Controller Selection, *86*
 PLC Settings, *88*

D

data type
 ET_Sercos3CmdType, *276*
 ET_Sercos3IDNTType, *277*
 ST_SercosConfiguration, *273*
 ST_SercosConfigurationDevice, *274*
Download application, *67*

E

embedded I/O configuration, *92*
encoder configuration, *100*
ET_Sercos3CmdType
 data type, *276*
ET_Sercos3IDNTType
 data type, *277*
Ethernet
 changeIPAddress function block, *251*
 EtherNet/IP adapter, *120*
 FTP Server, *176*
 Modbus TCP Client/Server, *174*
 Modbus TCP slave device, *179*
 Services, *167*
EtherNet/IP
 acyclic data exchange, *125*
External Event, *40*

F

FB_SercosProcedureCommandAsync
 sending Sercos commands asynchronously, *296*
FB_SercosReadServiceDataAsync
 reading Sercos data asynchronously, *292*
FB_SercosWriteServiceDataAsync
 writing Sercos data asynchronously, *294*
FC_SercosGetConfiguration
 function, *279*
FC_SercosReadServiceData
 function, *280*
FC_SercosReadServiceDataByTopAddr
 function, *283*
FC_SercosScanConfiguration
 function, *285*
FC_SercosWriteServiceData
 function, *287*
FC_SercosWriteServiceDataByTopAddr
 function, *289*
firewall
 configuration, *187*
 default script file, *187*
 script commands, *188*
firmware update, *245*
FTP client, *178*
FTP Server
 Ethernet, *176*
FTPRemoteFileHandling library, *178*
function
 FC_SercosGetConfiguration, *279*
 FC_SercosReadServiceData, *280*
 FC_SercosReadServiceDataByTopAddr,
 283
 FC_SercosScanConfiguration, *285*
 FC_SercosWriteServiceData, *287*
 FC_SercosWriteServiceDataByTopAddr,
 289

G

GetSerialConf
 getting the serial line configuration, *300*

H

Hardware Initialization Values, *62*

I

IP address
 changeIPAddress, *251*

L

libraries, *23*
Libraries
 FTPRemoteFileHandling, *178*
LMC078 Sercos3
 FB_SercosProcedureCommandAsync,
 296
 FB_SercosReadServiceDataAsync, *292*
 FB_SercosWriteServiceDataAsync, *294*
LXM32S configuration, *213*

M

M2•• communication
 GetSerialConf, *300*
 SetSerialConf, *301*
Memory Mapping, *29*
Modbus
 Protocols, *174*
Modbus Ioscanner, *227*
Modbus Manager, *236*
Modbus TCP Client/Server
 Ethernet, *174*
motion
 performance, *44*
 programming requirements, *43*
Motion task, *42*

O

Output Behavior, *62, 62, 62*
Output Forcing, *62*
overview of the Sercos standard , *200*

P

parameter types, *27*
PROFIBUS DP
 acyclic data exchange, *116*
Protocols, *167*
 IP, *169*
 Modbus, *174*

R

Reboot, *66*
Remanent variables, *70*
Reset cold, *65*
Reset origin, *66*
Reset warm, *65*
Run command, *64*

S

script commands
 firewall, *188*
script file
 syntax rules, *192*
Sercos
 adding devices, *208*
 adding third-party devices, *208*
 FB_SercosProcedureCommandAsync,
 296
 FB_SercosReadServiceDataAsync, *292*
 FB_SercosWriteServiceDataAsync, *294*
 interface configuration, *203*
Sercos scan, *209*
serial line
 ASCII Manager, *224*
 GetSerialConf, *300*
 Modbus Manager, *236*
 SetSerialConf, *301*
SERIAL_CONF, *303*

SetSerialConf, *301*
 setting the serial line configuration, *301*
Software Initialization Values, *62*
ST_SercosConfiguration
 data type, *273*
ST_SercosConfigurationDevice
 data type, *274*
state diagram, *52*
Stop command, *64*

T

Task
 Cyclic task, *39*
 Event task, *40*
 External Event Task, *40*
 Status task, *41*
 Types, *39*
 Watchdogs, *45*
third-party Sercos devices, adding, *208*

