

EcoStruxure Machine Expert - HVAC Operating Guide

(Original Document)

10/2018



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.	13
	About the Book.	15
Part I	Getting Started with EcoStruxure Machine Expert - HVAC	19
Chapter 1	Getting Started with EcoStruxure Machine Expert - HVAC	21
1.1	Introduction to EcoStruxure Machine Expert - HVAC	22
	General Description.	22
1.2	System Requirements and Supported Devices.	24
	System Requirements	25
	Registering the EcoStruxure Machine Expert - HVAC Software	26
	Supported Devices	27
	Connection and Communication Accessories	29
Chapter 2	Software Interface.	31
2.1	EcoStruxure Machine Expert - HVAC User Interface Basics	32
	Creating Projects with EcoStruxure Machine Expert - HVAC	33
	Developing Programs with EcoStruxure Machine Expert - HVAC.	34
	Navigating within EcoStruxure Machine Expert - HVAC	35
	Operating Modes	39
2.2	Workspace Customization.	41
	Layout	42
	Toolbars.	43
	Tool Windows Management	44
	Window Management	45
	Full Screen Mode	46
	Software Options	47
Chapter 3	Managing Projects	51
3.1	Managing Projects.	52
	Create a New Project	53
	Print a Project	56
	Save a Project	57
	Manage Existing Projects	59
	Distribute Projects	60
	Export CSV Files	61
	Select The Target Device	62

	Build All	64
	Download a Project to The Target	65
	Installer Software	67
	Close EcoStruxure Machine Expert - HVAC	68
Part II	Configuration	69
Chapter 4	The CONFIGURATION Tab	71
4.1	Overview	72
	Overview of the CONFIGURATION Window	73
	Menu Bar	75
	Toolbar	79
	Status Bar	80
Chapter 5	Managing Resources Elements	81
5.1	Overview	82
	Resources Window	83
	Supported Protocols	85
5.2	Target Device	87
	Target Device	87
5.3	Modbus Objects	88
	Modbus Objects	88
5.4	Target Menus	92
	Target Menu M171O	93
	Target Menu M171P/M172	95
5.5	I/O Mapping	96
	I/O Mapping	96
5.6	Alarms	97
	Alarms	97
5.7	Web Site	98
	Web Site	98
5.8	CAN Expansion Bus	101
	CAN Expansion Bus	102
	Using an Expansion Module as CAN Expansion Bus Field Slave	104
	CAN Expansion Bus for M171P Flush Mounting	109
	CAN Custom Device	110
	Using a CAN Custom Device	112
	CAN Expansion Bus Field - Virtual Master Channels	116

5.9	RS485	117
	RS485	118
	Using a TM171E27I/O as RS485 Slave	119
	Generic Modbus Object Overview	120
	Generic Modbus Object Messages	122
	Modbus Custom Devices	124
	Using a Modbus Custom Device	126
5.10	Ethernet	128
	Ethernet	128
5.11	Plugins	130
	Communication Modules Range Overview	130
Chapter 6	Technical Reference	133
6.1	Modbus Protocol	134
	Overview	135
	Data Types	136
	Function Codes	137
Part III	Programming	139
Chapter 7	The PROGRAMMING Tab	141
7.1	Overview	142
	Overview of the Programming Window	143
	Menu Bar	147
	Toolbars	155
	Status Bar	162
Chapter 8	Project Options	163
8.1	Project Options	164
	General Tab	165
	Code Generation Tab	166
	Build Output Tab	168
	Debug Tab	170
	Build Events Tab	171
	Cross Reference Tab	172
8.2	Working with Libraries	174
	Library Manager	175
	Exporting to a Library	177
	Importing from a Library or Another Source	178
	Updating Existing Libraries	180

Chapter 9	Managing Project Elements	181
9.1	Overview	182
	Project Window	182
9.2	Program Organization Units	183
	Overview	183
	Creating a Program	184
	Creating a Function Block/Function	185
	Editing POUs	186
	Source Code Encryption/Decryption	186
9.3	Variables	188
	Global Variables	188
	Local Variables	195
	Creating Multiple Variables	198
9.4	Tasks	199
	Associating a Program to a Task	199
	Task Configuration	201
9.5	Derived Data Types	202
	Overview	202
	Typedefs	203
	Structures	205
	Enumerations	207
	Subranges	209
9.6	Browse the Project	211
	Overview	211
	Object Browser	212
	Search with the Find in Project Command	217
9.7	Project Custom Workspace	219
	Overview	220
	Enable Custom Workspace Into an Existing Project	221
	Workspaces Migration	221
	Custom Workspace Basic Units	221
	Custom Workspace Operations	222
	Workspace Elements with Limitations	223
Chapter 10	Editing the Source Code	225
10.1	Overview	226
	Overview	226

10.2	Instruction List (IL) Editor	227
	Overview	227
	Editing Functions	228
	Reference to PLC Objects	228
	Automatic Error Location	228
	Bookmarks	229
10.3	Function Block Diagram (FBD) Editor	230
	Overview	230
	Creating a New FBD Document	231
	Adding/Removing Networks	231
	Labeling Networks	232
	Inserting and Connecting Blocks	233
	Editing Networks	235
	Modifying Properties of Blocks	235
	Getting Information on a Block	235
	Automatic Error Retrieval	235
10.4	Ladder Diagram (LD) Editor	236
	Overview	237
	Creating a New LD Document	237
	Adding/Removing Networks	238
	Labeling Networks	239
	Inserting Contacts	240
	Inserting Coils	242
	Inserting Blocks	243
	Editing Coils and Contacts Properties	244
	Editing Networks	244
	Modifying Properties of Blocks	245
	Getting Information on a Block	246
	Automatic Error Retrieval	246
	Inserting Variables	247
	Inserting Constants	247
	Inserting Expression	248
	Comments	249
	Branches	250

10.5	Structured Text (ST) Editor	251
	Overview	251
	Creating and Editing ST Objects	252
	Editing Functions	252
	Reference to PLC Objects	252
	Automatic Error Location	252
	Bookmarks	253
10.6	Sequential Function Chart (SFC) Editor	254
	Overview	254
	Creating a New SFC Document	254
	Inserting a New SFC Element	255
	Connecting SFC Elements	255
	Assigning an Action to a Step	256
	Specifying a Conditional Transition	258
	Assigning Conditional Code to a Transition	259
	Specifying the Destination of a Jump	261
	Editing SFC Networks	261
10.7	Variables Editor	262
	Overview	262
	Opening a Variables Editor	263
	Creating a New Variable	264
	Editing Variables	264
	Deleting Variables	267
	Sorting Variables	267
	Copying Variables	267
Chapter 11	Compiling	269
11.1	Overview	270
	Overview	270
11.2	Compiling the Project	271
	Overview	271
	Image File Loading	272
11.3	Compiler Output	273
	Overview	273
	Compiler Errors	274
11.4	Command-Line Compiler	276
	Overview	276

Chapter 12	Launching the Application.	277
12.1	Overview	278
	Overview	278
12.2	Setting Up the Communication	279
	Overview	280
	Saving the Last Used Communication Port.	286
12.3	On-Line Status.	287
	Connection Status	287
	Project Status	288
12.4	Downloading the Application	289
	Overview	289
12.5	Simulation	290
	Overview	290
12.6	Control the PLC Execution	291
	Control the PLC Execution	291
Chapter 13	Debugging	293
13.1	Overview	294
	Overview	294
13.2	Watch Window.	295
	Overview	295
	Opening and Closing the Watch Window	296
	Adding Items to the Watch Window.	296
	Removing a Variable	299
	Refreshment of Values	299
	Changing the Format of Data	300
	Working with Watch Lists	301
	Autosave Watch List	302
13.3	Oscilloscope	303
	Overview	304
	Opening and Closing the Oscilloscope	305
	Adding Items to the Oscilloscope	305
	Removing a Variable	309
	Variables Sampling	309
	Controlling Data Acquisition and Display.	310
	Changing the Polling Rate.	318
	Saving and Printing the Graph	318
13.4	Edit and Debug Mode	321
	Overview	321

13.5	Live Debug	322
	Overview	322
	SFC Simulation	323
	LD Simulation	324
	FBD Simulation	324
	IL and ST Simulation	325
13.6	Triggers	326
	Trigger Window	326
	Debugging with Trigger Windows	332
13.7	Graphic Triggers	347
	Graphic Trigger Window	347
	Debugging with the Graphic Trigger Window.	354
Chapter 14	Language Reference	371
14.1	Common Elements.	372
	Overview	372
	Basic Elements	372
	Elementary Data Types	373
	Derived Data Types	373
	Literals	376
	Variables	378
	Program Organization Units	382
	Operator and Standard Blocks.	385
14.2	Instruction List (IL)	404
	Overview	404
	Syntax and Semantics	404
	Standard Operators	406
	Calling Functions and Function Blocks	407
14.3	Function Block Diagram (FBD)	409
	Overview	409
	Representation of Lines and Blocks.	409
	Direction of Flow in Networks.	410
	Evaluation of Networks	410
	Execution Control Elements.	412

14.4	Ladder Diagram (LD)	414
	Overview	414
	Power Rails	415
	Link Elements and States	415
	Contacts	416
	Coils	417
	Operators, Functions, and Function Blocks	418
14.5	Structured Text (ST)	419
	Overview	419
	Expressions	419
	Statements in ST	421
	Assignments	422
	Function and Function Block Statements	423
	Selection Statements	425
	Iteration Statements	427
14.6	Sequential Function Chart (SFC)	430
	Overview	430
	Steps	431
	Transitions	433
	Rules of Evolution	434
	SFC Control Flags	439
	Check an SFC POU from Other Programs	440
14.7	EcoStruxure Machine Expert - HVAC Language Extensions	442
	Overview	442
	Macros	442
	Pointers	443
	Waiting Statement	444
Chapter 15	Errors Reference	445
15.1	Compile Time Error Messages	446
	Overview	446
Glossary		467
Index		475

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

This guide describes how to use the EcoStruxure™ Machine Expert - HVAC software to configure, program, and commission applications for supported logic controllers.

Validity Note

The information in this manual is applicable **only** for EcoStruxure Machine Expert - HVAC products.

This document has been updated for the release of EcoStruxure Machine Expert - HVAC V1.0.

For product compliance and environmental information (RoHS, REACH, PEP, EOL, etc.), go to www.schneider-electric.com/green-premium.

The technical characteristics of the devices described in the present document also appear online. To access the information online:

Step	Action
1	Go to the Schneider Electric home page www.schneider-electric.com .
2	In the Search box type the reference of a product or the name of a product range. - Do not include blank spaces in the reference or product range. - To get information on grouping similar modules, use asterisks (*).
3	If you entered a reference, go to the Product Datasheets search results and click on the reference that interests you. If you entered the name of a product range, go to the Product Ranges search results and click on the product range that interests you.
4	If more than one reference appears in the Products search results, click on the reference that interests you.
5	Depending on the size of your screen, you may need to scroll down to see the data sheet.
6	To save or print a data sheet as a .pdf file, click Download XXX product datasheet .

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

Related Documents

Title of documentation	Reference number
Modicon M171 Optimized Logic Controller Hardware Guide	<i>EIO0000002032 (ENG)</i>
EcoStruxure Machine Expert HVAC&R Function Library User Guide	<i>EIO0000002057 (ENG)</i>
TM172 Optimized & Performance 7/18 IO Instruction Sheet	<i>QGH90428</i>
TM172 Performance 28/42 IO Instruction Sheet	<i>NHA87740</i>
TM172 Optimized & Performance Isolated 28/42 IO Instruction Sheet	<i>PHA83703</i>
TM172 Optimized & Performance Expansion 12/28 IO Instruction Sheet	<i>QGH26895</i>
TM172DCLW*** Display Color Touchscreen Instruction Sheet	<i>QGH26896</i>
TM172DCLF• Display Color Touchscreen Flush Mounting Instruction Sheet	<i>PHA38669</i>

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

Product Related Information

WARNING

LOSS OF CONTROL

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

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

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

WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

Terminology Derived from Standards

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

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

Among others, these standards include:

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

Standard	Description
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.

Part I

Getting Started with EcoStruxure Machine Expert - HVAC

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
1	Getting Started with EcoStruxure Machine Expert - HVAC	21
2	Software Interface	31
3	Managing Projects	51

Chapter 1

Getting Started with EcoStruxure Machine Expert - HVAC

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
1.1	Introduction to EcoStruxure Machine Expert - HVAC	22
1.2	System Requirements and Supported Devices	24

Section 1.1

Introduction to EcoStruxure Machine Expert - HVAC

General Description

Overview

The EcoStruxure Machine Expert - HVAC (TM171SW) development tool makes it possible to create and customize IEC 61131-3 programs for various types of applications. You can download EcoStruxure Machine Expert - HVAC from [*Schneider-electric web site download center*](#). It is intended for applications in HVAC&R.

Features

EcoStruxure Machine Expert - HVAC is an IEC 61131-3 Integrated Development Environment supporting the whole range of languages defined in the standard.

In order to support the user in the activities involved in the development of an application, EcoStruxure Machine Expert - HVAC includes:

- Textual source code editors programming languages:
 - Instruction List (IL) (*see page 404*)
 - Structured Text (ST) (*see page 419*)
- Graphical source code editors programming languages:
 - Ladder Diagram (LD) (*see page 414*)
 - Function Block Diagram (FBD) (*see page 409*)
 - Sequential Function Chart (SFC) (*see page 430*)
- A compiler, which translates applications written according to the IEC standard directly into object code, avoiding the need for a run-time interpreter, thus making the program execution as fast as possible.
- A communication system which allows the download of the application to the target environment.
- A set of debugging tools, ranging from an easy-to-use watch window to more powerful tools, which allows the sampling of fast changing data directly on the target environment.

EcoStruxure Machine Expert - HVAC (TM171SW) Software Component

EcoStruxure Machine Expert - HVAC allows you to:

- Create and manage libraries, applications, and diagnostics
- Manage previously developed projects, upload/download projects, and modify device parameters from a communication port

EcoStruxure Machine Expert - HVAC is divided into four tabs:

- **CONFIGURATION**
- **PROGRAMMING**
- **DISPLAY**
- **COMMISSIONING**

For more information about the tabs of EcoStruxure Machine Expert - HVAC, refer to EcoStruxure Machine Expert - HVAC Software Tabs ([see page 37](#)).

Section 1.2

System Requirements and Supported Devices

What Is in This Section?

This section contains the following topics:

Topic	Page
System Requirements	25
Registering the EcoStruxure Machine Expert - HVAC Software	26
Supported Devices	27
Connection and Communication Accessories	29

System Requirements

Overview

The minimum system requirements for the PC on which EcoStruxure Machine Expert - HVAC software is installed are:

- Intel Pentium 1.6 GHz processor or greater
- 1 GB RAM, 2 GB preferred
- 500 MB hard disk space for a default installation; allow a further 1 GB hard disk space for installing libraries and dedicated applications
- Display resolution 800 x 600 pixels, 1280 x 768 pixels preferred
- USB interface for connecting devices
- The 32-bit or 64-bit version of one of the following operating systems:
 - Microsoft Windows 7
 - Microsoft Windows 8
 - Microsoft Windows 8.1
 - Microsoft Windows 10

Registering the EcoStruxure Machine Expert - HVAC Software

Overview

You can use the EcoStruxure Machine Expert - HVAC software for 30 days before you are required to register the software. When you register, you receive an authorization code to use the software.

Registering your EcoStruxure Machine Expert - HVAC software entitles you to receive technical support and software updates.

Registering

To register your EcoStruxure Machine Expert - HVAC software during installation:

Step	Action
1	Click the Register now button at the top of the Welcome page window.
2	Follow the instructions on the Registration Wizard. Click the Help button for more details.

To register your EcoStruxure Machine Expert - HVAC software after installation:

Step	Action
1	Select the Help → Registration command.
2	Click the Activate button.
3	Follow the instructions on the Registration Wizard. Click the Help button for more details.

To view details on the license key installed on your PC, click **About** on the **Welcome** page window.

Supported Devices

Logic Controllers

For more information about the logic controllers, refer to the following hardware guides:

Reference	Description	Hardware Guide
TM171O.....	Modicon M171 Optimized Logic Controller	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM171P.....	Modicon M171 Performance Logic Controller	<i>Modicon M171 Performance Logic Controller Hardware Guide</i>
TM172O.....	Modicon M172 Optimized Logic Controller	<i>Modicon M172 Logic Controller Hardware Guide</i>
TM172P.....	Modicon M172 Performance Logic Controller	

Expansion Modules

For more information about the expansion modules and their compatibility, refer to the following hardware guides:

Reference	Description	Hardware Guide
TM171EO14R	M171 Optimized Expansion 14 I/Os	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM171EO15R	M171 Optimized Expansion 15 I/Os	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM171EO22R	M171 Optimized Expansion 22 I/Os	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM172E12R	Modicon M172 Optimized & Performance Expansion 12 I/Os	<i>Modicon M172 Logic Controller Hardware Guide</i>
TM172E28R	Modicon M172 Optimized & Performance Expansion 28 I/Os	<i>Modicon M172 Logic Controller Hardware Guide</i>
TM171EP14R	Modicon M171 Performance Expansion 14 I/Os	<i>Modicon M171 Performance Logic Controller Hardware Guide</i>
TM171EP27R	Modicon M171 Performance Expansion 27 I/Os	

Displays

For more information about the displays and their compatibility, refer to the following hardware guides:

Reference	Description	Hardware Guide
TM171DLED	M171 Optimized Display LED	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM171DLCD2U	M171 Optimized Display LCD	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM171DWAL2U	M171 Optimized Wall thermostat without backlight	<i>Modicon M171 Optimized Logic Controller Hardware Guide</i>
TM171DWAL2L	M171 Optimized Wall thermostat with backlight	
TM171DGRP	M171 Performance Display Graphic	<i>Modicon M171 Performance Logic Controller Hardware Guide</i>
TM172DCLFG	M172 Color Touchscreen remote display flush mounting white	<i>Modicon M172 Logic Controller Hardware Guide</i>
TM172DCLFW	M172 Color Touchscreen remote display flush mounting gray	
TM172DCLWT	M172 Color Touchscreen remote display vertical mounting with built-in temperature sensor	<i>Modicon M172 Logic Controller Hardware Guide</i>
TM172DCLWTH	M172 Color Touchscreen remote display vertical mounting with built-in temperature and humidity sensors	
TM172DCLWTHP	M172 Color Touchscreen remote display vertical mounting with built-in temperature, humidity, and presence (PIR) sensors	

Connection and Communication Accessories

Overview

Several connection and communication accessories may be required to connect the target to a PC. Connection to the PC allows you to debug, commission, download and upload your application.

WARNING

AUTOMATIC RESTART OF CONTROLLER

- Do not download your application without first accessing the state of your machine or process.
- Do not download your application without first ascertaining that there is no risk of injury to anyone in or around your machine or process.

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

M1710 PC Connection

M1710 can be connected to a PC through the USB/TTL-I2C hardware interface:

- Use the software itself.
- Connect to the target device in order to control it.
- Connect to the Programming Stick component.

The Programming Stick is a memory support, which allows to:

- Update the firmware of the target device.
- Update the controller application of the target device.
- Update the parameter values of the target device.
- Upload the parameter values from the target device.

The following connection cables are available:

- “Yellow” cable with JST – molex terminals.
- “Blue” cable with JST – JST terminals.
- USB-A/A extension lead, 2 m.

M172 PC Connection

M172 can be connected to a PC through the USB port and a USB cable:

- Type Mini-B USB (DEVICE). Used to connect TM172P••••• / TM172O••••• to a PC via Mini-B/A USB cable for debugging, commissioning, downloading, and uploading with EcoStruxure Machine Expert - HVAC.
- Type micro-B USB (DEVICE). Used to connect TM172DCL•••• to a PC via micro-B/A USB cable for debugging, commissioning, downloading, and uploading with EcoStruxure Machine Expert - HVAC.

Alternatively, the type A USB (HOST) port of the controller can be used to connect a USB memory key drive to download the application.

The TM172P••••• / TM172O••••• can also be supplied through the USB cable with limited functionalities related to debugging, commissioning, downloading and uploading with EcoStruxure Machine Expert - HVAC.

For more details, refer to *Modicon M172 Logic Controller Hardware Guide*.

M171P PC Connection

M171P can be connected to a PC through the USB port and a USB cable:

- Type A USB (HOST). Used to connect a USB memory key drive when downloading the application, and to export if the application running on the PLC supports this feature.
- Type Mini-B USB (DEVICE). Used to connect M171P to a PC via Mini-B/A USB cable for debugging, commissioning, downloading, and uploading with EcoStruxure Machine Expert - HVAC.

M171P/M172 Programming Converters

To communicate with the controller, you can also use an USB/485 adapter TSXCUSB485 with cable VW3A83O6D3O.

Alternatively, if there is an RS232 serial port, M171P/M172 can be connected to the PC using an RS485/RS232 adapter.

M171P/M172 Communication Modules

A wide range of communication modules allows integration with industrial systems, BMS, and Ethernet networks.

NOTE: Not available with M171P Flush Mounting.

For more details, refer to communication modules description (*see page 130*).

Chapter 2

Software Interface

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
2.1	EcoStruxure Machine Expert - HVAC User Interface Basics	32
2.2	Workspace Customization	41

Section 2.1

EcoStruxure Machine Expert - HVAC User Interface Basics

What Is in This Section?

This section contains the following topics:

Topic	Page
Creating Projects with EcoStruxure Machine Expert - HVAC	33
Developing Programs with EcoStruxure Machine Expert - HVAC	34
Navigating within EcoStruxure Machine Expert - HVAC	35
Operating Modes	39

Creating Projects with EcoStruxure Machine Expert - HVAC

Overview

EcoStruxure Machine Expert - HVAC is a graphical programming tool designed to configure, develop, and commission programs for logic controllers.

EcoStruxure Machine Expert - HVAC Terminology

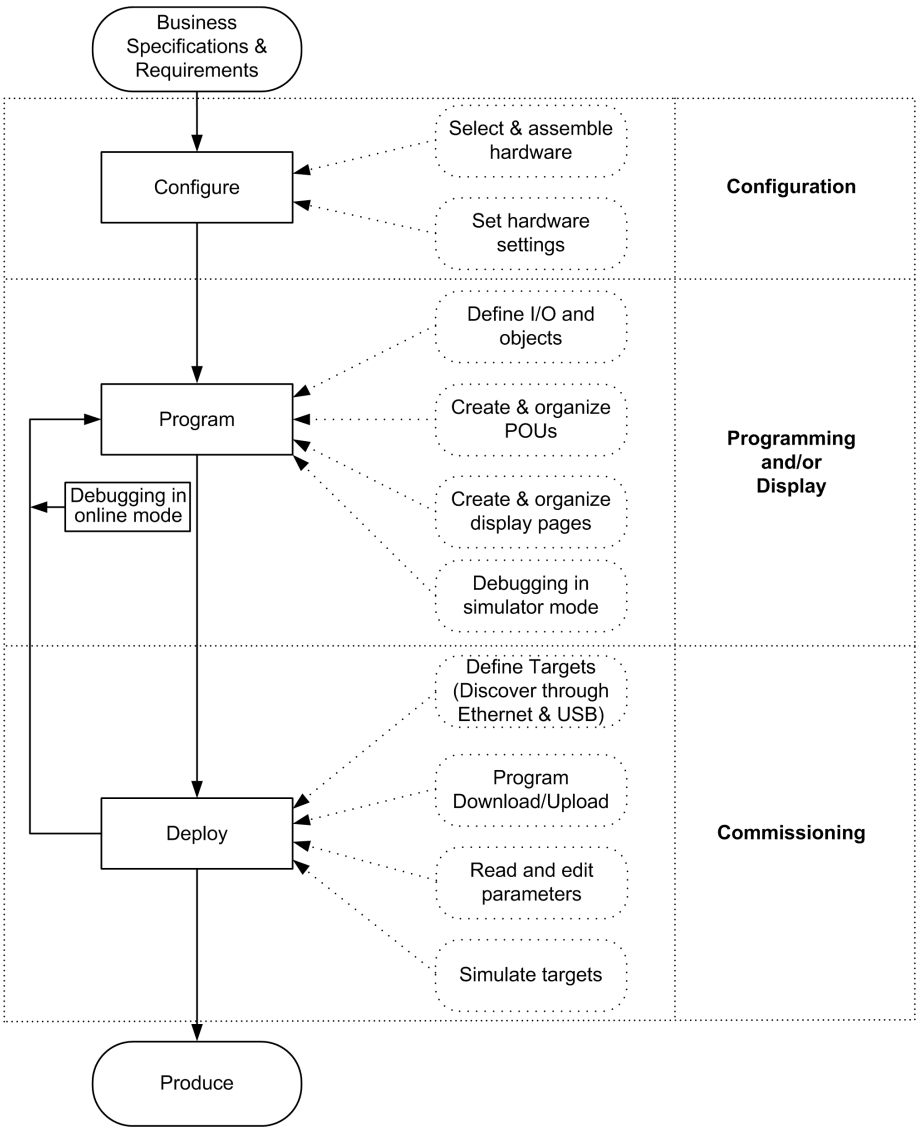
EcoStruxure Machine Expert - HVAC uses the following terms:

- **Project:** A EcoStruxure Machine Expert - HVAC project contains:
 - The developer and purpose of the project.
 - The configuration of the logic controller and associated expansion modules targeted by the project.
 - The source code of a program, symbols, comments, documentation, and the other related information.
- **Application:** Contains the parts of the project that are downloaded to the logic controller, including the compiled program and hardware configuration.
- **Program:** The compiled source code that runs on the logic controller.
- **POU (Program Organization Unit):** The reusable object that contains a variable declaration and a set of instructions used in a program.
- **Target:** Device connected to the PC.

Developing Programs with EcoStruxure Machine Expert - HVAC

Introduction

The following diagram presents the typical stages of developing a project in EcoStruxure Machine Expert - HVAC:



Navigating within EcoStruxure Machine Expert - HVAC

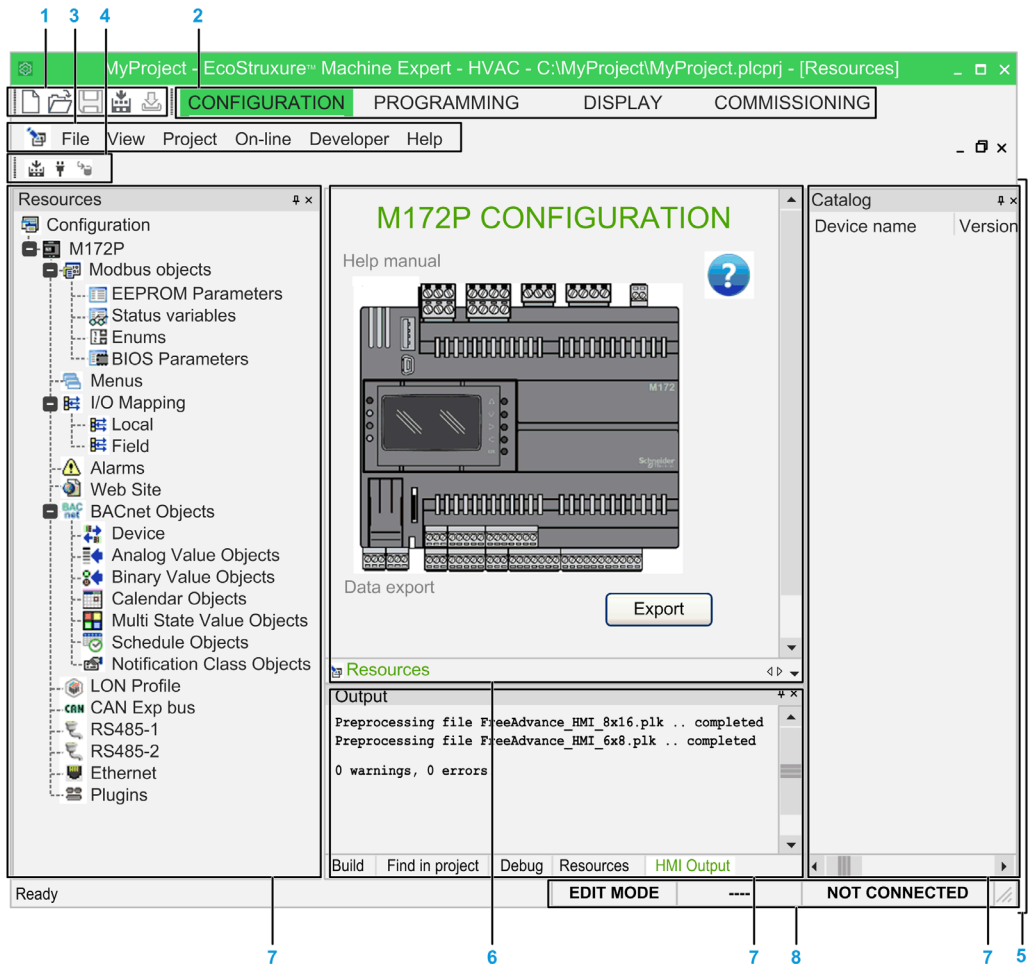
Welcome Window

The **Welcome** window is always displayed when you launch EcoStruxure Machine Expert - HVAC. Use this window to select and open an existing project or create a project (*see page 53*).

Main Window

The EcoStruxure Machine Expert - HVAC main window provides access to menus and commands, windows, and toolbars.

The illustration presents the EcoStruxure Machine Expert - HVAC main window:



Area	Description
1	Project toolbar (see page 36)
2	Tabs (see page 37)
3	Menu bar The content of the menu bars depends on the selected Tabs (see page 37).
4	Toolbars (see page 37)
5	Workspace (see page 41)
6	Editor window
7	Tool windows (see page 38)
8	Status bar (see page 162)

Project Toolbar

The project toolbar is located at the top left of the main window which provides access to the main commands using icons:

Icon	Description
	New project: creates a new project, any opened project must be closed. If the project is not saved, a dialog box asks to save the project. For more information, refer to Create a New Project (see page 53).
	Open project: opens a project saved on your computer. For more information, refer to Open an Existing Project (see page 59).
	Save project: saves modifications to the current project. For more information, refer to Save a Project (see page 57).
	Build all: compiles your project. For more information, refer to Build All (see page 64).
	Download all: downloads onto the target. For more information, refer to Download and Upload Applications (see page 65).

EcoStruxure Machine Expert - HVAC Software Tabs

EcoStruxure Machine Expert - HVAC consists of four development environments for programming controllers:

- **CONFIGURATION** (*see page 69*)
For creating networks. It is the entry point of the software.
- **PROGRAMMING** (*see page 139*)
For creating and managing libraries, controller applications and diagnostics.
It includes a **Simulation mode**, dedicated to software developers, for running and testing controller applications without a logic controller and expansion modules physically connected.
- **DISPLAY**
For creating the graphical interface on built-in displays and remote displays.
- **COMMISSIONING**
For downloading the project to the target (logic controller) device and modifying device parameters from a serial port.

Toolbars

EcoStruxure Machine Expert - HVAC has several toolbars dedicated to software tabs:

Toolbars	Dedicated tabs
Main	All
Project	PROGRAMMING (<i>see page 155</i>)
Debug	
FBD	
SFC	
LD	
Network	
Configuration	CONFIGURATION (<i>see page 79</i>)
HMI Page	DISPLAY
HMI Project	
HMI Profiles	
Commissioning	COMMISSIONING

To manage toolbars, refer to Toolbars Management (*see page 43*).

Tool Windows

EcoStruxure Machine Expert - HVAC has several tool windows dedicated to software tabs:

Tool windows	Dedicated tabs
Local variables	CONFIGURATION (<i>see page 69</i>)
Project	PROGRAMMING (<i>see page 139</i>)
Watch	
Properties Window	
Oscilloscope	
PLC run-time status	
Operators and blocks	
Library Tree	
Output	CONFIGURATION (<i>see page 69</i>)
Cross Reference	PROGRAMMING (<i>see page 139</i>)
Resources	CONFIGURATION (<i>see page 69</i>)
Catalog	
HMI Project	DISPLAY
HMI Properties	
HMI Actions	
HMI Vars and Parameters	
HMI Templates	
Commissioning	COMMISSIONING
Commissioning Watch	
Commissioning Oscilloscope	

To manage tool windows, refer to Tool Windows Management (*see page 44*).

Operating Modes

Introduction

The operating modes provide control to develop, debug, monitor, and modify the application when the controller is connected or not connected to EcoStruxure Machine Expert - HVAC.

EcoStruxure Machine Expert - HVAC can operate in the following modes:

- Offline mode
- Online mode
- Simulator mode

Offline Mode

EcoStruxure Machine Expert - HVAC operates in offline mode when no physical connection to a logic controller has been established.

In offline mode, you configure EcoStruxure Machine Expert - HVAC to match the hardware components you are targeting, then develop your application.

Online Mode

EcoStruxure Machine Expert - HVAC operates in online mode when a logic controller is physically connected to the PC.

In online mode, you can download your application to the logic controller. Downloading and uploading application is not possible in the simulator mode. EcoStruxure Machine Expert - HVAC then synchronizes the application in the PC memory with the version stored in the logic controller, allowing you to debug, monitor, and modify the application.

WARNING

AUTOMATIC RESTART OF CONTROLLER

- Do not download your application without first accessing the state of your machine or process.
- Do not download your application without first ascertaining that there is no risk of injury to anyone in or around your machine or process.

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

You can modify certain elements of a program in online mode. For example, you can add or delete rungs, or modify the values of certain function block parameters.

NOTE: Online program modifications are subjected to the predefined configuration. For more information, refer to Live Debug ([see page 322](#)) and Triggers ([see page 326](#)).

Simulator Mode

EcoStruxure Machine Expert - HVAC operates in simulator mode when a connection has been established with a simulated logic controller. In simulator mode, no physical connection to a logic controller is established; instead EcoStruxure Machine Expert - HVAC simulates a connection to a logic controller and the expansion modules to run and test the program.

For more information, refer to Simulation (*see page 290*).

Section 2.2

Workspace Customization

Overview

This section describes how to manage the user interface elements of the software. It allows you to set up the integrated software environment in the way which best suits to your specific development process.

What Is in This Section?

This section contains the following topics:

Topic	Page
Layout	42
Toolbars	43
Tool Windows Management	44
Window Management	45
Full Screen Mode	46
Software Options	47

Layout

Overview

The layout of the software workspace can be freely customized in order to suit your needs.

The layout configuration is saved on application exit. Your modifications remain between different working sessions.

Reset Layout

To reset layout parameters to default values of standard layout:

Step	Action
1	Click File → Options... . The Program options dialog box appears.
2	In Tool windows area, click Reset bars positions button. A new dialog box appears.
3	Click OK button.
4	In Program options dialog box, click OK button.
5	In the project toolbar, click Save project icon.
6	Click File → Exit .
7	Restart EcoStruxure Machine Expert - HVAC.

NOTE: When you reset the layout, all the tab layouts are reset.

Toolbars

Overview

The toolbars are displayed as follows:



For more details, refer to Toolbars ([see page 155](#)).

Show/Hide

To show or hide a toolbar, proceed as follow:

Step	Action
1	Click View → Toolbars and select Or right-click in the toolbar area.
2	The toolbar list is displayed in a pop-up window: ✓ Main ✓ Project ✓ Debug ✓ FBD Bar ✓ SFC Bar ✓ LD bar Network Configuration HMI Page HMI Project HMI Profils Commissioning
3	Click the Toolbar name you want to show/hide.

Move

To move a toolbar, click its left border and drag it to the new destination.

Tool Windows Management

Show/Hide Tool Windows

To show or hide a tool window:

Step	Action
1	Click View → Tool windows or right click in the Tool window area. A pop-up window is displayed.
2	Select the toolbar to show or hide

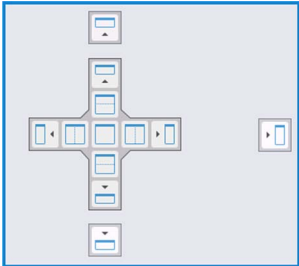
Undock Tool Windows

To undock a tool window from its default location, click its title bar and drag it to a new location.

To take back a tool window to its last docked location, double-click the title bar of the window.

Dock Tool Windows

To dock a tool window to another location:

Step	Action
1	Click the title bar of the tool window and move the pointer. A guide diamond appears, move the pointer over the desired portion of the guide diamond. The designated area is shaded. 
2	Release the mouse button.

Auto-Hide Tool Windows

To auto-hide a tool window, click the pin button on the top right corner of the tool window.

The tool window is reduced in a tab on the upper left corner of the main window.

To show the tool window again, click the tab.

To switch the tool window from auto-hide mode to regular docking mode, relick the pin button when the tool window is displayed.

Window Management

Overview

EcoStruxure Machine Expert - HVAC allows you to navigate between the opened source code editor windows.

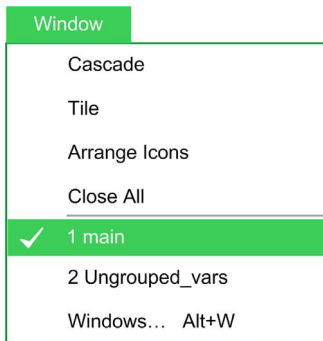
Document Tab

To switch between the currently open editors, click the title program on the tab located below the programming window.

Window Menu

The **Window** menu allows you:

- To present the currently opened programs in a cascade.
- To present the currently opened programs in tiles.
- To arrange the icons of the minimized documents in the bottom left-hand corner of the editors window.
- To close the currently opened programs.
- To switch between the currently opened programs by clicking the title program.



Full Screen Mode

Overview

To switch on or off the full screen mode, click **View → Full screen** (or press Ctrl+U).

In full screen mode, the source code editor extends to the whole working area, making the coding with graphical programming languages easier.

Software Options

Overview

EcoStruxure Machine Expert - HVAC allows you to customize some options of the software.

To display the dialog box options, click **File → Options....**

General

General tab allows you to configure:

- **Visual Theme:**
You can choose in the **Color theme** list between **Standard** and **Dark** color themes.
- **Save options:**
 - **Autosave:** if **Autosave** box is checked, the software periodically saves the whole project. You can specify the period of execution of this task by entering the number of minutes between two automatic savings in **Interval (min)** box.
 - **Max previous version to keep:** it indicates the maximum number of copies of the project that must be zipped and stored in the **PreviousVersions** folder.
- **Output window:**
You can specify the family and the size of the font used for output window.
- **Communication:**
If **Use last port** check box is selected, the last used port is set as the default one.
- **Tooltip:**
If **Enable tooltip on editors** check box is selected, small information boxes appear when the cursor is placed over a symbol in the editors.
- **Tool windows:**
You can specify the family and the size of the font used for tool windows.
Reset bars positions: the layout of the dock bars in the IDE is reinitiate to default positions and dimensions. In order to take effect, the software must be restarted.
- **Source editors options:**
If **ST - LD: Auto declaration of variables** check box is selected, variables are automatically declared for ST and Ladder programs.

Graphic Editor

This panel lets you edit the properties of the LD, FBD, and SFC source code editors.

You can specify the family and the size of the font used for graphical editors.

You can also modify the colors of the graphical object.

Text Editors

You can specify the family and the size of the font both for code and variable editors.

Language

You can modify the language of the environment:

Step	Action
1	Select a language from the list displayed in this panel.
2	Click the Select button.
3	Click the OK button to confirm.
4	To make effective this modification, restart the software.

Custom Tools

You can add up to 16 commands to the **Custom tools** menu. These commands can be associated with any program that runs on your operating system. You can also specify arguments for any command that you add to the **Custom tools** menu.

To add a tool to the **Custom tools** menu:

Step	Action
1	In the Command box, type the full path of the program file you want to use as a tool. Otherwise, you can select the program file by clicking the browse button.
2	In the Arguments box, type the arguments - if any - to be passed to the executable command mentioned at the first step. They must be separated by a space.
3	In Menu string box, type the name you want to give to the tool you are adding. This is the string that is displayed in the Tools menu.
4	Click Add button to insert the new command into the suitable menu.
5	Click OK button to confirm, or Cancel button to quit.

For example, if you want to add **Windows Calculator** to the **Custom tools** menu:

Step	Action
1	Fill the fields of the dialog box as displayed.
2	Click Add button. The name you gave to this tool (in this example, Calc) is now displayed in the Custom tools menu.

Merge

If **Enable Merge** check box is selected, you can configure the following parameters:

- **Identical name:**
 - **Objects with different types**
 - **Object with same type (not variables)**
 - **Variables**
- **Check address:**
 - **Overlapped**
 - **Copy\Paste mapped variable**

For more information about Merge, refer to Merge Function ([see page 179](#)).

Chapter 3

Managing Projects

Section 3.1

Managing Projects

What Is in This Section?

This section contains the following topics:

Topic	Page
Create a New Project	53
Print a Project	56
Save a Project	57
Manage Existing Projects	59
Distribute Projects	60
Export CSV Files	61
Select The Target Device	62
Build All	64
Download a Project to The Target	65
Installer Software	67
Close EcoStruxure Machine Expert - HVAC	68

Create a New Project

Overview

There are two ways to create a new project:

- In the Welcome page (*see page 53*).
- In the New project window (*see page 55*).

Welcome Page

When EcoStruxure Machine Expert - HVAC starts, the **Welcome** page appears:

New project

Name:

Directory:

☐ Case sensitive

	M172 07-18 I/Os	668
	M172 28-42 I/Os	596
	M172DCL Portrait	659
	M172DCL Landscape	659

Open project

Recent projects

MyProject

Import old project

Schneider Electric

The **Welcome** page is divided into three group boxes:

- **New project**
- **Open project**
- **Import old project**

To create a new project:

Step	Action
1	In Name box, type the name of the new project. The string you enter is also the name of the folder which contains the files making up the project. The name can be modified afterward, refer to Project Options (<i>see page 164</i>).
2	In the Directory box, the default location of this folder is indicated. Click the browse button to choose another folder.
3	In the device list, click the target device which runs the project. NOTE: Available targets are listed in Supported Devices (<i>see page 27</i>).
4	If Case-sensitive check box is selected, the source code of the project is case-sensitive. This option can be modified afterward, refer to Project Options (<i>see page 164</i>). NOTE: This option is not compliant with IEC 61131-3 standard.
5	Click Create button.

To open an existing project, use one of the two procedures:

- Click **Choose from disk...** button.
Result: A dialog box to appear, which lets you load the directory containing the project and select the relative project file.
- In the **Recent projects** list, double-click the project name.

To import an old project:

- Click **Choose from disk...** button.
Result: A dialog box appears, which lets you load the directory containing the project and select the relative project file.
The "old project" is one created with SoMachine HVAC. EcoStruxure Machine Expert - HVAC will proceed to convert the old program. However, incompatibilities may exist between SoMachine HVAC and EcoStruxure Machine Expert - HVAC.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Always verify that your application program operates as it did prior to the conversion, having all the correct configurations, parameters, parameter values, functions, and function blocks as required.
- Modify the application as necessary such that it conforms to its previous operation.
- Thoroughly test and validate the newly compiled version prior to putting your application into service.

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

New Project Window

Click **File** → **New project** or **File** icon  of the project toolbar to display the **New project** window:

New project

Project

Name

Directory

C:\User\Documents

...

Target selection

Select the target for a new project

M171O 542.8

Options

☐ Case sensitive

OK

Cancel

NOTE: If you already have an open project, a dialog box appears to ask you if you want to save the current project.

Create a new project:

Step	Action
1	In Name box, type the name of the new project. The string you enter is also the name of the folder which contains the files making up the project. The name can be modified afterward, refer to Project Options (see page 164).
2	In the Directory box, the default location of this folder is indicated. Click the  browse button to choose another folder.
3	In the Select the target for a new project list, click the target device which runs the project. NOTE: Available targets are listed in Supported Devices (see page 27).
4	If Case-sensitive check box is selected, the source code of the project is case-sensitive. This option can be modified afterward, refer to Project Options (see page 164). NOTE: This option is not compliant with IEC 61131-3 standard.
5	Click OK button.

Print a Project

Print the Project

EcoStruxure Machine Expert - HVAC allows you to print the data which make up the project such as programs and variables.

To print a project:

Step	Action
1	In PROGRAMMING tab, click File → Print Project .
2	In Name box, select the printer to print your project.
3	Click OK button.

Print the Current Working Window

EcoStruxure Machine Expert - HVAC allows you to print only the current working window.

To print the current working window:

Step	Action
1	In PROGRAMMING tab, click File → Print...
2	In Name box, select the printer to print your project.
3	Click OK button.

Print Preview

To preview your printing before, click **File → Print preview**. The preview is displayed in the current working window.

Save a Project

Overview

EcoStruxure Machine Expert - HVAC projects can be saved as files to the local PC or into a server directory. This file has the extension *.plcprj or *.ppjs and contains:

- The source code of the program.
- The current hardware configuration.
- Settings and preferences of the EcoStruxure Machine Expert - HVAC project.

Save the Project

To save the project:

- Click  **Save project** icon on the project toolbar.
- Click **File** → **Save project**.

Save the Project As

To save the project with a different name, a different format or in a different folder:

Step	Action
1	Click File → Save project As
2	Type the new name of the project file.
3	Browse and select the new folder in which to store the project file.
4	Choose the new format (<i>see page 60</i>) of the project file.
5	Click OK button.

AutoSave

EcoStruxure Machine Expert - HVAC includes an **AutoSave** feature that periodically saves your project as you work on it.

AutoSave saves data in a separate folder, called **Backup**, stored at the same location of the project folder.

If you regularly save or close the currently opened file, the associated auto save file is deleted, unless the save file command is canceled or terminated in error. In this case, the file is kept. If you reopen a project for which an appropriate auto save file is found, the **AutoSave** Backup dialog box is displayed. You can reopen the auto save project or the saved last version.

You can specify the interval time (in minutes) between saving. By default, **AutoSave** is running with 1 minute of interval. For more information, refer to Save options (*see page 47*).

Backup Copies

EcoStruxure Machine Expert - HVAC includes a backup feature of the previous version of the project on which you are working.

When you explicitly save the project, EcoStruxure Machine Expert - HVAC saves the current version (before save) of the project in the **PreviousVersions** folder stored at the same location of the project folder.

You can set the upper limit of the backup files to be kept on your PC. By default this is 10; set to 0 if you want to disable this feature. For more information, refer to Save options ([see page 47](#)).

Manage Existing Projects

Open an Existing Project

To open an existing project, click **File** → **Open project**.

You can also open an existing project in the **Welcome** page (when no project is opened).

This causes a dialog box to appear, which lets you load the directory containing the project and select the relative project file.

Edit the Project

To modify an element of a project:

Step	Action
1	Locate the element in the tree structure of the tool window.
2	Double-click its name to open it. Result: An editor consistent with the object type is opened. For example, when you double-click the name of a project POU, the appropriate source code editor is displayed; if you double-click the name of a global variable, the variable editor is displayed.

EcoStruxure Machine Expert - HVAC cannot modify elements of a project when at least one of the following conditions holds:

- EcoStruxure Machine Expert - HVAC is in debug mode.
- It is an object of an included library (whereas you can modify an object that you imported from a library).
- The project is opened in read-only mode (view project).

Close the Project

To close the project, click **File** → **Close project** or close the software.

In both cases, when there are modifications which have not been saved, EcoStruxure Machine Expert - HVAC asks you to choose between saving and discarding them.

Then the **Welcome** page ([see page 53](#)) is displayed so that you can start a new working session.

Distribute Projects

Overview

To share a project with another developer, send either a copy of one or more project files or a redistributable source module (RSM) generated by EcoStruxure Machine Expert - HVAC.

In the former case, the number of files to share depends on the format of the project file:

- PLC single project file (`.ppjs` file extension): the project file itself contains the whole information needed to run the application (assuming the receiving developer has an appropriate available target device). It includes the source code modules so that only the `.ppjs` file is needed to share the project.
- PLC multiple project file (`.ppjx` or `.ppj` file extension): the project file contains only the links to the source code modules composing the project, which are stored as single files in the project directory. The whole directory is needed to share the project.
- Full XML PLC project file (`.plcprj`): the project file is generated entirely in XML language. The information contained in the project file and its behavior are the same as `.ppjs` file extension.

To generate a redistributable source module (RSM), click **Project → Generate redistributable source module**.

EcoStruxure Machine Expert - HVAC displays the name of the RSM file and lets you choose whether to protect the file with a password. To protect the file, a password must be entered.

The advantages of the RSM file format are:

- It is encoded in binary format, thus it can only be read by third parties which use EcoStruxure Machine Expert - HVAC.
- It can require a password when opening the file in EcoStruxure Machine Expert - HVAC.
- Its size is reduced because it is a binary file.

Export CSV Files

Overview

EcoStruxure Machine Expert - HVAC allows you to export parameters and variables defined in **.csv** format which can be used for sharing information and developing documentation to be supplied with the product.

Data Export

To export data:

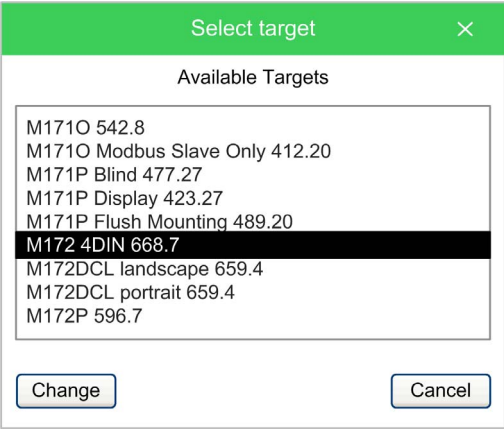
Step	Action
1	Click CONFIGURATION tab.
2	In the Resources window, click the target device.
3	In the main box, click Export button.
4	In Data Export window, select data you want to export and click OK button.
5	In the Save As dialog box, choose the name of your file and click Save button.
6	In Export succeeded dialog box, click OK button.

Select The Target Device

Overview

You may need to adapt a PLC application on a new target device which differs from the one for which you originally wrote the code.

To adapt your application project to a new target device:

Step	Action
1	In PROGRAMMING tab, click Project → Select target... The following dialog box appears: 
2	Select one of the target devices listed in the combo box.
3	Click Change to confirm your choice, Cancel to discard.
4	If you confirm, click Yes button to complete the conversion or No button to quit. If you click Yes button, EcoStruxure Machine Expert - HVAC updates the project to work with the new target. It also makes a backup copy of the project files in a subdirectory inside the project directory. Therefore you can roll back the operation by manually (that is, using Windows Explorer) replacing the project files with the backup copy.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Always verify that your application program operates as it did prior to the conversion, having all the correct configurations, parameters, parameter values, functions, and function blocks as required.
- Modify the application as necessary such that it conforms to its previous operation.
- Thoroughly test and validate the newly compiled version prior to putting your application into service.

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

Build All

Overview

To compile your project, click  **Build All** icon in the project toolbar.

The result of the compilation is displayed in the **Output** window ([see page 146](#)).

Download a Project to The Target

Overview

A project can be downloaded in different ways depending on the controller.

The following is a table of correspondence indicating possible connection types with the controllers:

Controller	TTL port		USB A port	USB Mini-B port	RS485 port Modbus	Ethernet
	TM171ADMI programming cable	TM171AMFK programming stick	USB A Memory key	USBA/USB Mini-B Cable	USB/RS485 adapter	Ethernet Cable
M171O	✓	✓	-	-	-	-
M171P	-	-	✓	✓	✓	✓ ⁽¹⁾
M172O	-	-	-	✓	✓	✓ ⁽¹⁾
M172P	-	-	✓	✓	✓	✓
(1) An Ethernet communication module must be connected to the controller, except for M171P Flush Mounting which has integrated Ethernet communications.						

Download Controller Project onto Target

Steps to download the project onto the target:

Step	Action
1	Configure the communication with the Device Link Manager Config, accessible in: ● On-line → Set up communication... menu of CONFIGURATION tab or PROGRAMMING tab. ● Or Target → Communication settings menu of COMMISSIONING tab. The properties are visible and can be edited by clicking Properties. The protocol must be activated beforehand. For more information, refer to Setting Up the Communication (see page 279)
2	Connect the target physically to the computer. For more information on the hardware connection, refer to the related hardware guides.
3	Connect the target using On-line → Connect menu of CONFIGURATION tab or PROGRAMMING tab. For more information, refer to On-Line Status (see page 287)
4	Download the project from CONFIGURATION tab or PROGRAMMING tab: ● Select On-line → Download code ● Or press F5 It is also possible to use the Download all button in the project toolbar.
5	Follow the dialog boxes instructions.

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.

 WARNING

AUTOMATIC RESTART OF CONTROLLER

- Do not download your application without first accessing the state of your machine or process.
- Do not download your application without first ascertaining that there is no risk of injury to anyone in or around your machine or process.

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

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.

Installer Software

Description

A stand-alone software, **Installer** software, is delivered in addition with EcoStruxure Machine Expert - HVAC.

The **Installer** software allows you to:

- Manage the maintenance of systems by downloading projects.
- Manage the configuration of bound controllers.
- Configure the devices.
- Modify the BIOS parameters.

The **Installer** software is dedicated for maintenance use. The project code cannot be modified. For more details, refer to the **Installer** software on-line help.

NOTE: Binding is defined herein as the connection, and exchange of variables, between logic controllers.

Close EcoStruxure Machine Expert - HVAC

Overview

To exit EcoStruxure Machine Expert - HVAC, click the **Close** button in the top right-hand corner of the EcoStruxure Machine Expert - HVAC window.

You can also click the **Exit** button on the **Welcome** page window.

Part II

Configuration

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
4	The CONFIGURATION Tab	71
5	Managing Resources Elements	81
6	Technical Reference	133

Chapter 4

The CONFIGURATION Tab

Section 4.1

Overview

What Is in This Section?

This section contains the following topics:

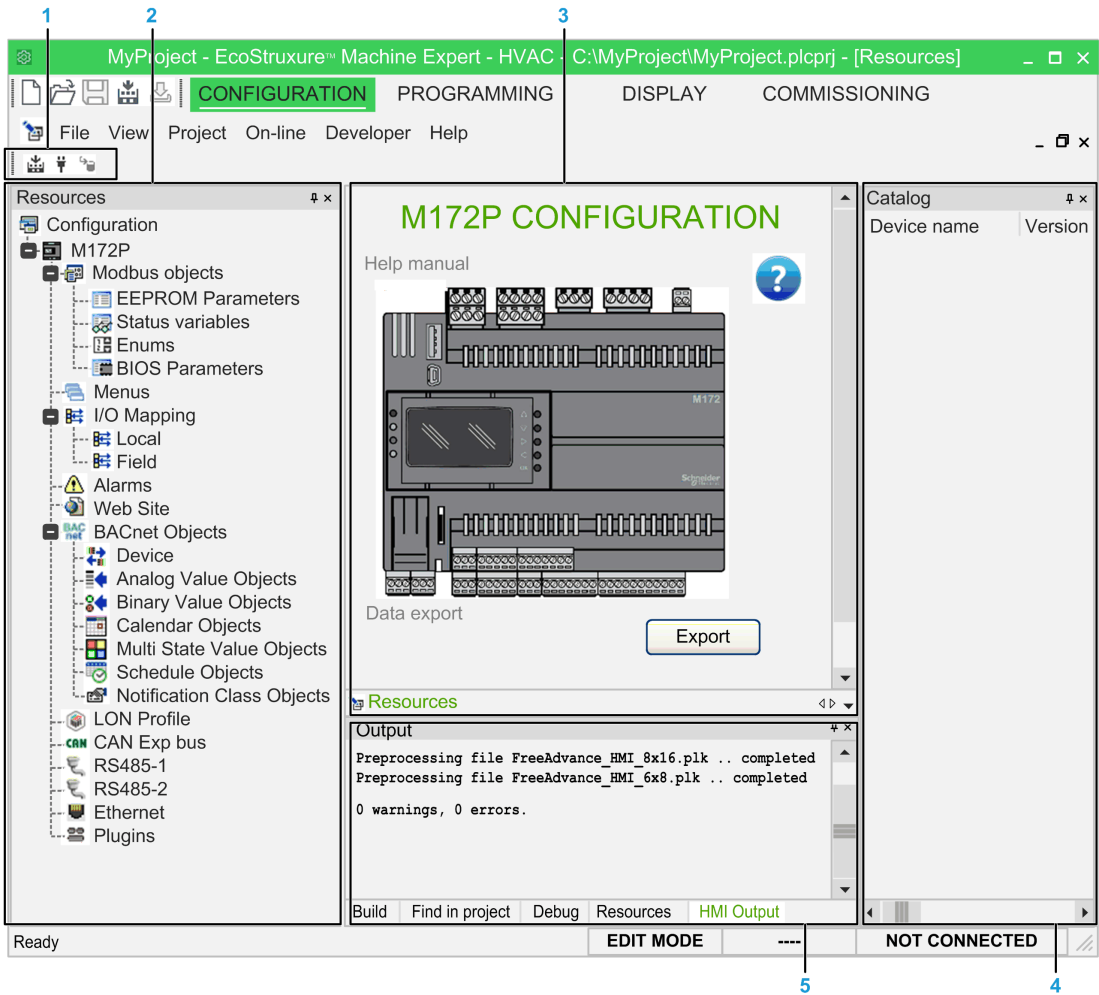
Topic	Page
Overview of the CONFIGURATION Window	73
Menu Bar	75
Toolbar	79
Status Bar	80

Overview of the CONFIGURATION Window

General Description

CONFIGURATION is the entry point for starting to develop projects.

The following illustration presents the default **CONFIGURATION** window:



Marker	Description	
1	Toolbar	This toolbar shows the tools in form of icons. For more information, refer to Toolbars (see page 79).
2	Resources window	This window shows the configurable parameters of the device. For more information, refer to Content of the Resources Window (see page 83).
3	Editor window	This window allows you to edit the content of the current selection in Resources window.
4	Catalog window	This window shows the devices available from the catalog. NOTE: Dynamic visibility of devices based on selections (for example communication modules).
5	Output window	This window shows the messages relating to the development of the project (file opening, reading/writing errors, status of connection to device, and so on). NOTE: The connection to the device is also visible in the status bar (see page 80). For more information, refer to Download and Upload Applications (see page 65).

Menu Bar

Overview

The Menu bar is composed by several menus:

- File ([see page 75](#))
- View ([see page 76](#))
- Project ([see page 77](#))
- On-line ([see page 77](#))
- Developer ([see page 77](#))
- Help ([see page 78](#))

File Menu

This menu gives access to features allowing you to manage your project:

Command	Icon	Key	Description
New project		-	Creates a new project (see page 53).
Open project		Ctrl+O	Opens an existing project (see page 59).
Save project		Ctrl+S	Saves the current open project. (see page 57)
Save project as...	-	-	Saves the current open project specifying new name, location and extension (see page 57).
Close project	-	-	Closes the open project (see page 59).
Options...	-	-	Opens the Program options dialog box (see page 47).
Print...		Ctrl+P	Prints the document of the currently active window (see page 56).
Print preview		-	Creates a preview of the document of the currently active window, ready to be printed.
Printer setup...	-	-	Opens the Printer setup dialog box.
..recent..	-	-	Lists a set of recently opened project file.
Exit	-	-	Closes EcoStruxure Machine Expert - HVAC.

View Menu

This menu gives access to features allowing you to choose what is displayed in the workspace:

Command	Icon	Key	Description
Toolbars	-	-	Refer to Toolbars (<i>see page 43</i>).
Main	-	-	Shows or hides the Main toolbar.
Project	-	-	Shows or hides the Project bar.
Debug	-	-	Shows or hides the Debug toolbar.
FBD Bar	-	-	Shows or hides the FBD toolbar.
SFC Bar	-	-	Shows or hides the SFC toolbar.
LD Bar	-	-	Shows or hides the LD toolbar.
Network	-	-	Shows or hides the Network toolbar.
Configuration	-	-	Shows or hides the Configuration toolbar.
HMI Page	-	-	Shows or hides the HMI Page toolbar.
HMI Project	-	-	Shows or hides the HMI Project toolbar.
HMI Profiles	-	-	Shows or hides the HMI Profiles toolbar.
Commissioning	-	-	Shows or hides the Commissioning toolbar.
Tool windows	-	-	Refer to Tool Windows Management (<i>see page 44</i>).
Local variables	-	-	Shows or hides the Local variables window.
Project	-	-	Shows or hides the Project window.
Watch	-	Ctrl+T	Shows or hides the Watch window.
Properties Window	-	Ctrl+W	Shows or hides the Properties Window window.
Oscilloscope	-	-	Shows or hides the Oscilloscope window.
PLC run-time status	-	-	Shows or hides the PLC run-time status bar.
Operators and blocks	-	-	Shows or hides the Operators and blocks window.
Library Tree	-	-	Shows or hides the Library Tree window.
Output	-	-	Shows or hides the Output window.
Cross Reference	-	-	Shows or hides the Cross Reference window.
Resources	-	-	Shows or hides the Resources window.
Catalog	-	-	Shows or hides the Catalog window.
HMI Project	-	-	Shows or hides the HMI Project window.
HMI Properties	-	-	Shows or hides the HMI Properties window.
HMI Actions	-	-	Shows or hides the HMI Actions window.
HMI Vars and Parameters	-	-	Shows or hides the HMI Vars and Parameters window.
HMI Templates	-	-	Shows or hides the HMI Templates window.
Commissioning	-	-	Shows or hides the Commissioning window.

Command	Icon	Key	Description
Commissioning Watch	-	-	Shows or hides the Commissioning Watch window.
Commissioning Oscilloscope	-	-	Shows or hides the Commissioning Oscilloscope window.
Full screen		Ctrl+U	Expands the currently active document window to fill entire screen (<i>see page 46</i>). (Esc to exit from this mode).

Project Menu

This menu gives access to features allowing you to compile your project and to manage your target device:

Command	Icon	Key	Description
Compile		F7	Launches the compiler (<i>see page 64</i>).
Select target...	-	-	Lets you select a new target for the project (<i>see page 62</i>).
Refresh current target	-	-	Lets you update the target file for the same version of the target.

On-line Menu

This menu gives access to features allowing you to communicate with the target device:

Command	Icon	Key	Description
Set up communication...	-	-	Lets you set the properties of the connection to the target (<i>see page 280</i>).
Connect		-	Starts the communication with the device (<i>see page 280</i>).
Download code		F5	Download the configuration and PLC application of the project to the device (<i>see page 289</i>).

Developer Menu

This menu gives access to features allowing you to share your project with other developers:

Command	Icon	Key	Description
Import EDS	-	-	Import EDS (Electronic Data Sheet) file (<i>see page 110</i>).
Run Modbus custom Editor	-	-	Run Modbus custom editor (<i>see page 124</i>).
Data export	-	-	Export data to a CSV file (<i>see page 61</i>).

Help Menu

This menu gives access to features allowing you to read documentation about hardware or some specific EcoStruxure Machine Expert - HVAC features:

Command	Icon	Key	Description
Index	-	-	Lists the Help keywords and opens the related topic.
Context	-	F1	Context-sensitive help. Opens the topic related to the currently active window.
Registration	-	-	Register the software (<i>see page 26</i>).
About...	-	-	Credits and version information.

Toolbar

Introduction

The toolbar appears at the top of the EcoStruxure Machine Expert - HVAC window to provide access to frequently used functions.

For generalities of toolbars, refer to Toolbars description (*see page 43*).

Configuration Toolbar

The **Configuration** toolbar has the following buttons:

Icon	Key	Description
	-	Compile Launches the compiler.
	-	Connects to the target Starts the communication with the device.
	-	Code download Download the configuration and PLC application of the project to the device.

Status Bar

Overview

The status bar is located at the bottom right of the EcoStruxure Machine Expert - HVAC window. It indicates the state of communication and the status of the application currently executing on the target device.



For more information, refer to:

- Edit and Debug Mode ([see page 321](#))
- Connection Status ([see page 287](#))
- Application Status ([see page 288](#))

Chapter 5

Managing Resources Elements

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
5.1	Overview	82
5.2	Target Device	87
5.3	Modbus Objects	88
5.4	Target Menus	92
5.5	I/O Mapping	96
5.6	Alarms	97
5.7	Web Site	98
5.8	CAN Expansion Bus	101
5.9	RS485	117
5.10	Ethernet	128
5.11	Plugins	130

Section 5.1

Overview

What Is in This Section?

This section contains the following topics:

Topic	Page
Resources Window	83
Supported Protocols	85

Resources Window

Overview

The **Resources** window allows you to configure the device:

- Define parameters and variables.
- Create and configure the embedded website (for M171P and M172 only).
- Configure the hardware structure of the project.
- Configure communication protocols.

Content of the Resources Window

The **Resources** window consists of the following items:

Item	Icon	Description
Target device (see page 87)		Shows the picture of the target device and allows you to configure some settings.
Modbus objects (see page 88)		Define the EEPROM parameters (non-volatile memory parameters) and Status variables which may then be used in the application code. Defines EEPROM parameters (non-volatile memory parameters) and Status variables which can be displayed on the target device and read using the Modbus protocol (RTU or TCP) or the CAN protocol.
Menus (see page 92)		Manages menus where you can group the Modbus objects that are shown in COMMISSIONING .
I/O Mapping (see page 96)		Defines the links between variables and physical I/O of the target device.
Alarms (see page 97)		Defines alarm variables which status must be managed by the developer.
Web Site (see page 98)		Defines website pages to monitor the device from a web browser.
BACnet Objects		Configures the BACnet objects.
LON Profile		Configures the LonWorks protocol.
CAN Exp bus (see page 101)		Configures the CAN Expansion bus.
RS485-1 (see page 117)		Configures the first RS485 port.
RS485-2 (see page 117)		Configures the second RS485 port.

Item	Icon	Description
Master Modbus RTU (see page 117)		Configure the RS485 port. It only applies to M171O Modbus master/slave.
Ethernet (see page 130)		Configures the Ethernet port.
Plugins (see page 130)		Configures protocols using communication modules.
Help		Shows LED reference for the developer. For M171O only.

NOTE: The **Resources** window content depends on the selected device.

Match Software and Hardware Configuration

The I/O that may be embedded in your controller is independent of the I/O that you may have added in the form of I/O expansion. It is important that the logical I/O configuration within your program matches the physical I/O configuration of your installation. If you add or remove any physical I/O to or from the I/O expansion bus, then you must update your application configuration. This is also true for any field bus devices you may have in your installation. Otherwise, there is the potential that the expansion bus or field bus no longer function while the embedded I/O that may be present in your controller continues to operate.

 **WARNING**

UNINTENDED EQUIPMENT OPERATION
Update the configuration of your program each time you add or delete any type of I/O expansions on your I/O bus, or you add or delete any devices on your field bus.
Failure to follow these instructions can result in death, serious injury, or equipment damage.

Expansion Bus

You must monitor within your application the state of the bus and the error state of the module(s) on the bus, and to take the appropriate action necessary given your particular application.

 **WARNING**

UNINTENDED EQUIPMENT OPERATION

- Include in your risk assessment the possibility of unsuccessful communication between the logic controller and any I/O expansion modules.
- Monitor the state of the I/O expansion bus using the dedicated system words and take appropriate actions as determined by your risk assessment.

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

Supported Protocols

Overview

Each device has the following resources, which are shown as nodes of the target. Select the **Mode** and add the device from the catalog:

Target	Communication Bus	Description
M171P	CAN expansion bus	On-board For I/O expansion and remote display
	RS485	On-board Modbus RTU (master/slave)
	Ethernet	Optional with communication module: Modbus TCP (Server), BACnet IP (Server), HTTP
	Plugins	Optional modules available separately
M171P Flush Mounting	CAN expansion bus	On-board For I/O expansion
	RS485	On-board Modbus RTU (master/slave) or BACnet MS/TP (Server)
	Ethernet	On-board Modbus TCP (client/server), BACnet IP (Server), FTP, HTTP
M172O	CAN expansion bus	On-board For I/O expansion and remote display
	RS485-1	On-board Modbus RTU (Slave only) or BACnet MS/TP (Server)
	RS485-2	On-board Modbus RTU (master/slave) or BACnet MS/TP (Server)
	Ethernet	Optional with communication module: Modbus TCP (client/server), BACnet IP (Server), FTP, HTTP
	Plugins	Optional modules available separately
M172P	CAN expansion bus	On-board For I/O expansion and remote display
	RS485-1	On-board Modbus RTU (Slave only) or BACnet MS/TP (Server)
	RS485-2	On-board Modbus RTU (master/slave) or BACnet MS/TP (Server)
	Ethernet	On-board Modbus TCP (client/server), BACnet IP (Server), FTP, HTTP
	Plugins	Optional modules available separately

NOTE: The **Catalog** window shows the devices that can be added (by dragging them) to the corresponding protocol.

NOTE: On the RS485 protocol, you can also connect generic Modbus devices.

Providing HMI Pages

M172P supports HMI Remote so its pages can be downloaded and displayed in **DISPLAY** tab for M171P displays.

This feature is not supported by M171P Flush Mounting. No linked device can upload HMI pages from M171P Flush Mounting device.

Section 5.2

Target Device

Target Device

Overview

In the **Resources** window, double-click the title of the project to display the editor window.

The editor window presents the graphic of the target device and lets you access some settings.

M171 Configuration

In the editor window, you have the possibility to:

- Define the parameter value shown on the main display on idle state by selecting a value in the **Fundamental state display** box.
NOTE: Only available for M171O.
- Set the execution time of the project in milliseconds (ms).
The default setting is 100 ms. The available range is 20...100 ms.
NOTE: Only available for M171O.
- Export parameters and variables defined in **.csv** format.
For more information, refer to Export CSV Files ([see page 61](#)).
- Consult the hardware guide of the device by clicking the  icon.

M172 Configuration

In the editor window, you have the possibility to:

- Export parameters and variables defined in **.csv** format.
For more information, refer to Export CSV Files ([see page 61](#)).
- Consult the hardware guide of the device by clicking the  icon.

Section 5.3

Modbus Objects

Modbus Objects

Overview

Modbus objects allow you to define **EEPROM parameters** (non-volatile memory parameters) and **Status variables**, which can be displayed on the target device and read using the Modbus protocol (*see page 134*).

EEPROM parameters (non-volatile memory parameters) and **Status variables** can be used in the application code. They appear in the **Project** window: **Project** → **Aux Variables** → **Global Shared**.

It is possible to add or remove parameters and variables (with **Add** and **Remove** buttons) in the same way as for variables in the **Project** window.

The **Recalc** button allows you to recalculate the addresses of the selected rows.

EEPROM Parameters

 **EEPROM Parameters** allows you to create the variables which the developer intends to save in non-volatile memory even if the device is powered off.

Refer to Status Variables (*see page 90*) for details about the columns of the editor window.

This table presents the columns of the editor window:

Column	Description
IPA	Pre-assigned index
Address	Resource Modbus address
Name	Resource name which may be used by the developer in the controller application.
Display label	Name displayed in the application menu of the M171O target (4-digit 7 segments).
Device type	Type of data displayed on target and Device.
Application type	Type of data used in controller application.
Size	Significant only in the case of STRING type. Dimension (Length) of the string. Default and max= 31 characters.
Read only	Enables/disables editing of Status variables.
Default value	Default value of the object.
Min	Minimum value of the object.
Max	Maximum value of the object.

Column	Description
Scale	Conversion coefficients between Device type and Application type . Application Type = scale x Device Type + offset.
Offset	
Unit	Unit of measurement of Device Type displayed on Device and if available with icon on target.
Format	Display format for Default Value / Min / Max. For example, XXX.Y display of whole number with decimal point.
Access Level	This column does not apply to M171P/M172. Refer to Visibility of Menu Resources (<i>see page 94</i>).
Description	Free text.
Note	

NOTE: Columns can be hidden or shown by right-clicking its name and selecting **Hide column** or **Show columns** command.

The non-volatile memory is specified for a minimum of 100,000 write cycles.

Using the non-volatile memory for a cyclic write operation may result in quickly exceeding its life cycle limits resulting in an inoperative memory.

NOTICE

Do not use non-volatile memory registers for cyclic write operations.

Failure to follow these instructions can result in equipment damage.

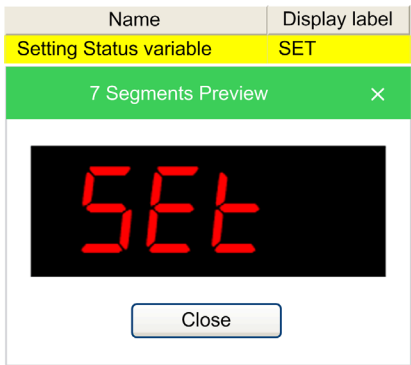
Status Variables



Status variables allow you to define the status variables which can be displayed in the menu of the device.

NOTE: For M171O, each variable has a transcoding on the controller due to the 4-digit / 7-segment display. In the **Display** label box, you can select the transcoding and see a preview on the display by clicking the ellipsis (...).

Some letters are not displayed (for example x and z) so there is a blank space on the display. If the display label is **SET**, **5 E E** appears on the display.



This table presents the columns of the editor window:

Column	Description
IPA	Pre-assigned index
Address	Resource Modbus address
Name	Resource name which may be used by the developer in the controller application.
Display label	Name displayed in the application menu of the M171O target (4-digit 7 segments).
Device type	Type of data displayed on target and Device.
Application type	Type of data used in controller application.
Size	Significant only in the case of STRING type. Dimension (Length) of the string. Default and max= 31 characters.
Read only	Enables/disables editing of Status variables.
Default Value	Default value of the object.
Min	Minimum value of the object.
Max	Maximum value of the object.
Scale	Conversion coefficients between Device Type and Application Type.
Offset	Application Type = scale x Device Type + offset.

Column	Description
Unit	Unit of measurement of Device Type displayed on Device and if available with icon on target.
Format	Display format for Default Value / Min / Max. For example, XXX.Y display of whole number with decimal point.
Access Level	This column does not apply to M171P/M172. Refer to Visibility of Menu Resources (see page 94).
Description	Free text.
Note	

NOTE: Columns can be hidden or shown by right-clicking its name and selecting **Hide column** or **Show columns** command.

Enums



Enums allows you to define enumeration elements which can be used in the **Device Type** column of the editor window.

For more information about **Enums**, refer to Enumerations ([see page 207](#)).

BIOS Parameters



BIOS Parameters allows you to define variations in the default **BIOS parameters** map which is factory-set by Schneider Electric.

Section 5.4

Target Menus

What Is in This Section?

This section contains the following topics:

Topic	Page
Target Menu M171O	93
Target Menu M171P/M172	95

Target Menu M171O

Overview

The target menu consists of a BIOS menu and an Application menu.

The BIOS menu is factory-set.

The following table defines the main functions of the keys/LEDs of the target device.

Key	Press	Description
F5	Short	Switch from BIOS menu to Application menu and conversely.
F1 or F3	Short	Navigate folders and edit values.
F2	Short	Cancel operation (ESC function).
F4	Short	Access to set menu.
F2+F4	Short	Access to Prg menu.
F1/F2/F3/F4	Long	Managed by developer (by using target variable <code>sysKeyFunctions[]</code>).

The LEDs are managed by the developer by using target variable `sysLocalLeds`.

The elements entered in the table in this section are displayed on Device.

Menu Prg

The **Prg** menu can consist of one or more folders, defined by the developer, into which are inserted:

- EEPROM parameters (non-volatile memory parameters).
- Status variables.
- BIOS parameters.
- Inputs and outputs.

Menu Set

The **Set** menu is created in the same way as the **Prg** menu.

The **Set** menu contains the **AL** folder.

Visibility of Menu Resources

The visibility of the resources created by the developer is indicated in the following table:

Access Level column	Visibility on Device	Visibility on target	Note
Always visible	Yes	Yes	Object assigned to a Prg or Set menu
Level 1	Yes	Yes Level 1	
Level 2	Yes	Yes Level 2	
Never visible	Yes	No	Object NOT assigned to any Prg or Set menu
Never visible	Yes Visible in the folder ALL PARAMETERS	No	

Target Menu M171P/M172

Overview

The target menu can be created by using the **CONFIGURATION** tab. In the **Resources** window, right-click **Menus** and select **Add Menu** command.

The BIOS menu is factory-set and is visible from Device.

The main functions of the keys/LEDs of the target device can be programmed by using the **DISPLAY**. LEDs are also programmable from in the **Operators and blocks** window from the **PROGRAMMING**.

In this section, you can define a menu (not visible on the display) and the folders/variables of which it is composed.

The menu can consist of one or more folders, defined by the developer, into which are entered:

- EEPROM parameters (non-volatile memory parameters).
- Status variables.

Section 5.5

I/O Mapping

I/O Mapping

M171O I/O Mapping

You can define the links between variables and physical I/O of M171O:

- **Local:** local variables of the controller module.
- **Extended:** variables of the expansion module.
- **Remote:** variables on the displays.

M171P/M172 I/O Mapping

You can define the links between variables and physical I/O of M171P/M172:

- **Local:** local variables of the M171P/M172 base module.
- **Field:** variables of I/O expansion.

To map variables with physical I/Os, enter the name of the PLC variable in **I/O Mapping → Local → Variable** column.

NOTE: If correctly defined, the variables defined in **Resources** are located in **Project** window: **Project → Aux Variables → Global shared** automatically. The project must be saved without errors for the variables.

Section 5.6

Alarms

Alarms

M171O Alarms

It is possible to define alarm variables which status must be managed by the developer.

Alarm variables can be used in the application code. They appear in the **Project** window: **Project** → **Aux Variables** → **Global shared**.

If the variable assumes a value other than zero, the label is displayed in the **Alarms** folder (AL) of the set menu in M171O.

In the **Resources** window, click **Alarm** to display the Alarm variable list.

This table presents the columns of the editor window:

Column	Description
Name	Resource name which may be used by the developer in the controller application.
Short name	Name displayed in the application menu of the M171 target (4-digit). NOTE: If not filled, the text displayed on the controller will be the 4 first characters of the variable.
Description	Description of the variable.

NOTE: Each variable has a transcoding on the controller due to the 4-digit / 7-segment display. In the **Short name** label box, you can see a preview on the display by clicking the ellipsis (...). Some letters are not displayed (for example x and z) so there is a blank space on the display. If the text is **SET, 5 E E** appears on the display.

M171P/M172 Alarms

In the M171P/M172 target, it is only a Global type USINT declaration.

The alarms for M171P/M172 are only defined to enable the portability of an M171O project.

Section 5.7

Web Site

Web Site

Web Functionalities

The M172 features web functionalities, offering makers of machinery and systems integrators remote access. Having a web-based connection in machines reduces support and maintenance by minimizing call-out charges. End users also benefit, as they can monitor their own systems both locally and from distance, using the graphics interface of any browser.

Main web functionalities:

- Web-based access.
- Remote reading.
- Local and remote system control, including alarms management.
- Preventive and predictive maintenance.
- Email alarm alerts.

'WEB MENU TITLE' WEB TABLE PAGE

+ Add

- Remove

↑ Up

↓ Down

☒ Enable build

Refresh (ms):

1000

(0=disable refresh)

Password:

Page title:

Filename:

Site template:

Choose...

#	Name	Control	Label	Section	Text size	Img filename	Img X	Img Y	Enum values
---	------	---------	-------	---------	-----------	--------------	-------	-------	-------------

Care must be taken and provisions made for use of this product as a control device to avoid inadvertent consequences of commanded machine operation, controller state changes, or alteration of data memory or machine operating parameters.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Configure and install the mechanism that enables the remote HMI local to the machine so that local control over the machine can be maintained regardless of the remote commands sent to the application.
- You must have a complete understanding of the application and the machine before attempting to control the application remotely.
- Take the precautions necessary to assure that you are operating remotely on the intended machine by having clear, identifying documentation within the application and its remote connection.

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

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.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Define a secure password for the Web Functionalities, and do not allow unauthorized or otherwise unqualified personnel to use the features therein.
- Ensure that there is a local, competent, and qualified observer present when operating on the controller from a remote location.
- Configure and install the mechanism that enables the remote HMI local to the machine so that local control over the machine can be maintained regardless of the remote commands sent to the application.
- You must have a complete understanding of the application and the machine before attempting to control the application remotely.
- Take the precautions necessary to assure that you are operating remotely on the intended machine by having clear, identifying documentation within the application and its remote connection.

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

Section 5.8

CAN Expansion Bus

What Is in This Section?

This section contains the following topics:

Topic	Page
CAN Expansion Bus	102
Using an Expansion Module as CAN Expansion Bus Field Slave	104
CAN Expansion Bus for M171P Flush Mounting	109
CAN Custom Device	110
Using a CAN Custom Device	112
CAN Expansion Bus Field - Virtual Master Channels	116

CAN Expansion Bus

Description

M171P/M172 have one on-board CAN Expansion bus port, plus another one available as an external plugin. Each port can be configured as **Not used** (disabled), or **Master (for field)**.

M171P Flush Mounting can be connected using CAN Expansion bus for field mode or for network mode.

CAN EXPANSION BUS	
Mode	☐ Not used ☒ Master (for field)
Baud rate	☒ 500 Kb/s ☐ 250 Kb/s ☐ 125 Kb/s ☐ 50 Kb/s
Master Settings	Node ID (1...122,125): 125 ? Heartbeat time (ms): 0 Sync COBID: 128 Sync Cycle (ms): 0

Master (for Field)

When you configure the CAN Expansion bus port as **Master (for field)**, the controller acts as a CAN Expansion bus master on this port. You can attach CAN Expansion bus slave devices and exchange data with remote I/O.

For a CAN Expansion bus master port, you have to configure:

- Baud rate used in this CAN Expansion bus network (in kb/s).
- Node ID for the master (1...127), by default is 125.
- Heartbeat time in ms, by default 0 (heartbeat producer disabled).
- The `SYNC` COBID to use, by default 128.
- The period for the `SYNC` cycle in ms, by default 0 (sync disabled).

After you added and configured the various CAN Expansion bus slaves, you can link the remote objects of the slave and the internal PLC variables to read or write.

The set of controller objects you can read or write is made of:

- Status variables, created with **CONFIGURATION**.
- Field variables, created with **CONFIGURATION**.

Slave (for Binding)

The binding mode can be configured with the **Installer** software (*see page 67*).

When you configure the CAN Expansion bus port as **Slave**, the M171P/M172P bus port acts as a CAN Expansion bus slave. You can exchange data by Binding I/O with other devices on the CAN Expansion bus network.

Configuring the port:

For a CAN Expansion bus slave port, you have to configure:

- Baud rate used in this CAN Expansion bus network (in kb/s).
- Node ID for the slave (1...127), by default is 1.
- The “virtual network” where this M171P Display is attached; in the tree appears a small colored circle of same color of the chosen network (same color means same network).
- The maximum number of devices that can be bound is 10

The **Binding** object:

Once a CAN Expansion bus port has been configured as **Slave**, the device is able to **SEND** objects on the network. To make the device able to **READ** objects from other devices, it is necessary to add a **Binding** object to the port.

The set of PLC objects you can send or receive is made of:

- **EEPROM parameters** (non-volatile memory parameters), created with **CONFIGURATION**.
- **Status variables**, created with **CONFIGURATION**.

Clicking the **Binding** object displays its configuration page: here is a grid where you want to insert the remote objects to read, and link them to the local destinations.

To do this, click the **Add** button. A window displaying the “public” objects from the other devices on the network appears. Here you can apply search filters and choose which objects to read (multi-selection is also supported).

Once you have inserted the remote objects to read, you have to assign the local destination locations to write, choosing from the list in the **Dest parameter** column or manually inserting the **Address**.

NOTE: It is necessary to rebuild the PLC project to update the list of public objects.

Example:

- **M171 Perf Display_1** reads from **M171 Perf Display_2** the **M171_2_par1** object and puts it in its local **M171_1_par1** object.
- **M171 Perf Display_1** reads from **M171 Perf Display_2** the **M171_2_par2** object and puts it in its local **M171_1_par2** object.

In the Period field, you can configure the period for each parameter; the object is updated every “period” in ms.

Using an Expansion Module as CAN Expansion Bus Field Slave

CAN Expansion Bus with M172 Expansion Modules and M172 Controllers

In this configuration example, you want to use M172 Expansion 28 I/Os as expansion of a M172P device. The same can be done for other logic controllers.

Configure M172P CAN Expansion Bus in **Master** (for field) mode. From the **Catalog** window, select **M172 Expansion 12&28 I/Os** node and drop it on the **CAN Expansion Bus** node.

M172 Expansion 12&28 I/Os configuration is divided into four tabs:

- **GENERAL**: to configure the network parameters.
- **DIGITAL I/O**: to configure the digital I/Os.
- **ANALOG I/O**: to configure the analog I/Os.
- **ADVANCED SETTINGS**: to add and remove variables.

GENERAL tab of **M172 Expansion 12&28 I/Os**:

M172 EXPANSION GENERAL CONFIGURATION

GENERAL	DIGITAL I/O	ANALOG I/O	ADVANCED SETTINGS
Network settings	Node number (1...122)	1	
Advanced <<<			
	Node Guard Period (ms)	200	
	Life time Factor	3	
	Boot time elapsed (ms)	2000	
	☒ USER DEFINED Mode		
	☐ SYNC Mode		
	☐ EVENT Mode		
	☐ CYCLIC Mode	0	ms

DIGITAL I/O tab of M172 Expansion 12&28 I/Os:

M172 EXPANSION DIGITAL I/O CONFIGURATION

GENERAL

DIGITAL I/O

ANALOG I/O

ADVANCED SETTINGS

IOs Configuration

☒ 12 IOs☐ 28 IOs

Digital INPUTS

PLC Var

DataBlock

DI1

DI2

Digital OUTPUTS

PLC Var

DataBlock

DO1

DO2

DO3

DO4

DO5

DO6

EcoStruxure Machine Expert - HVAC knows the **M172 Expansion 12&28 I/Os** dictionary. Each object can be assigned to a **PLC variable**.

Follow this procedure do assign an object to a **PLC variable**:

Step	Action
1	Click  Assign button.
2	Select the PLC variable that you want to assign to the PLC object: Choose PLC variable × Filter: M172P: DI_xDIE1 (BOOL) – DIE1 digital input M172P: DI_xDIE2 (BOOL) – DIE2 digital input M172P: DI_xDIE3 (BOOL) – DIE3 digital input M172P: DI_xDIE4 (BOOL) – DIE4 digital input M172P: DI_xDIE5 (BOOL) – DIE5 digital input M172P: DI_xDIE6 (BOOL) – DIE6 digital input M172P: DI_xDIE7 (BOOL) – DIE7 digital input M172P: DI_xDIE8 (BOOL) – DIE8 digital input M172P: xCmdConsumReset (BOOL) M172P: xWMbTm171VEVPardE30Cir1 (BOOL) OK Cancel NOTE: It is possible to apply a filter to the choice list by entering a string of characters.
3	Click OK button to validate.
4	The PLC Var name is added and its address is displayed in the DataBlock field.

NOTE: Click  **Unassign** button to remove the assignment.

ANALOG I/O tab of M172 Expansion 12&28 I/Os:

M172 EXPANSION ANALOG I/O CONFIGURATION

GENERAL

DIGITAL I/O

ANALOG I/O

ADVANCED SETTINGS

IOs Configuration

☐ 12 IOs

☒ 28 IOs

Analogue OUTPUTS #1, #2

PLC Var

AO1

AO2

DataBlock

Analogue INPUTS Frequency and Counter #1, #2

PLC Var

FDI1 Counter

FDI1 Frequency

FDI2 Counter

FDI2 Frequency

DataBlock

Analogue INPUTS

Temp UM

0 = °C

Analogue INPUT #1

Configuration

0 = NTC(NK103)

PLC Var

AI1

Full Scale Min

0

Full Scale Max

1000

Calibration

0

Sub Configuration

3 = Low Pass Filter enabled, analogue value converted

DataBlock

EIO0000003412 10/2018

107

ADVANCED SETTINGS tab of M172 Expansion 12&28 I/Os:

M172 EXPANSION CONFIGURATION

GENERALDIGITAL I/OANALOG I/OADVANCED SETTINGS

Add

Remove

#	Label	Index	SubIndex	Type	Value	Timeout
1	FullScaleMin_AI1	3d78	0	INT		100
2	FullScaleMax_AI1	3d79	0	INT		100

CAN Expansion Bus with M171P Expansion

In this configuration example, you want to use TM171EP27R as expansion of a M171P device. The same can be done for other logic controllers.

Configure M171P CAN Expansion Bus in **Master** (for field) mode. From the **Catalog** window, it is possible to select **M171P Expansion 27 I/Os** node and drop it on the CAN Expansion Bus node.

M171P Expansion 27 I/Os configuration is similar to CAN Custom configuration (Using a CAN Custom Device (*see page 112*)) with dynamic PDO mapping feature disabled. Available Input/Output objects that can be mapped on PLC variables via PDO are listed in **PDO TX - INPUT** and **PDO RX - OUTPUT**.

CAN Expansion Bus for M171P Flush Mounting

Description

M171P Flush Mounting can be connected using CAN Expansion bus in field mode or in network mode.

Field Mode

To connect M171P Flush Mounting in this mode select M171P Display or **M171 Perf. Blind** CAN Expansion bus node and select the option **Master** (for field) then take M171P Flush Mounting device from **Catalog** tab and drop it over CAN Expansion bus node.

Select **M171 Perf. Flush Mounting_1** child node and configure **Network** settings.

Probes:

M171 Perf. Display_1 can access to **M171 Perf. Flush Mounting_1** on-board probes. To do so select **Probes - Input** tab then it is possible to map a **M171 Perf. Display_1** parameter to let it obtain the value of an on-board **M171 Perf. Display** probe.

Choose one of the probes and click **Assign** button. Take one of the **M171 Perf. Display_1** INT parameter and click **OK** button.

HMI:

It is possible to associate to a M171P Flush Mounting (configured as CAN Expansion bus field slave) an HMI project with local pages. M171P Flush Mounting would be able to show its own target variables and parameters of the CAN Expansion bus master to which it belongs.

Network Mode

The HMI remoting and binding mode can be configured with the **Installer** software ([see page 67](#)).

In this connection mode, M171P Flush Mounting can be linked to one of the remote devices that are available on the network to navigate HMI Remote pages provided by other devices.

Using the **Installer** software, it is possible to do so by indicating one of the available HMI Remote devices of the network.

For example:

You have a CAN Expansion bus network with **M171 Perf. Display_1** and **M171 Perf. Blind_1** then add as first-level node **M171 Perf. Flush Mounting_1** to the network taking it from **Catalog** panel.

Click **CAN Exp bus** node of **M171 Perf. Flush Mounting_1** and select **Master** (for HMI remoting and binding) node, assign unique **Node ID** and select network **CAN Exp bus1**.

Binding of variables between **M171 Perf. Flush Mounting_1** and **M171 Perf. Blind_1** and **M171 Perf. Display_1** is allowed in a network of this type (see chapter CAN Expansion bus - Binding ([see page 103](#)) for more details).

HMI Remote pages:

In CAN Expansion bus network mode, it is possible to configure M171P Flush Mounting in order to be linked to 0 to 10 remote devices that can provide HMI Remote pages to the keyboard.

To add HMI Remote pages select **M171 Perf. Flush Mounting_1** node, then press **Add** on the **HMI Remote pages** box thus all available devices will be displayed and the user can select the pages to navigate.

CAN Custom Device

Description

CAN custom device can be created and added to the Catalog by importing its **EDS** file. Therefore, you can use any third-party CAN Expansion bus device as a slave, as long as it provides a standard, compliant **EDS** file (Electronic Data Sheet) that follows the DS402 CiA specification.

Importing a New CAN Custom Device

To import a new **CAN custom** device, choose **Developer → Import EDS** command.

The **Import EDS** window appears:

Import EDS

Source EDS: C:\ATV600_CANopen_EDS_V2.1\SEATV6x0_0102 Choose...

New Name: ATV6x0_V2.1 1.513

Short Name: ATV6x0_V2p1_1p513

Has dynamic PDO mapping: ☒

Vendor Name: Schneider Electric

Product Name: ATV6x0_V2.1 Revision: 1.513

Description: EDS for ATV6x0

Comments:

Conforamnce class = S20
Product function = Frequency Converter
Supported profiles = 402 v3
Boot-up time = 100ms

Number of Objects

Mandatory: 3 Optional: 39 Manufacturer: 87

OK

Cancel

Here you have to configure:

- The source **EDS** file to import, using the **Choose...** button.
- The full name of the device (by default is **Product name + Revision**).
- The short name must not include any special characters or spaces.
- Dynamic PDO mapping: if you activate this option, you are able to configure manually and modify the default PDO mapping read from the EDS to match the actual mapping of the slave, otherwise the PDO mapping is read-only and determined only by the EDS default values.

After you have chosen the **EDS** file, the window will show a resume of the device characteristic and number of objects (detailed in mandatory, optional, manufacturer).

Deleting a CAN Custom Device

When the device you want to delete is visible in the **Catalog** window (for example when a CAN Expansion bus port is selected and is in **Master** mode), you can right-click on it and choose the **Delete from catalog** command.

Using a CAN Custom Device

Description

When you insert a **CAN custom** device as a CAN Expansion bus slave (for example under a CAN Expansion bus slave port) and click it on the **Resources** window, the editor window displays four tabs.

General Tab

ATV6X0_V2.1 1.513 CONFIGURATION

GENERAL

SDO SET

PDO TX - INPUT

PDO RX - OUTPUT

Network Settings

Node number (1...122)

3

Node Guard Period (ms)

200

Life time Factor

3

Boot time elapsed (ms)

10000

Node heartbeat producer time (ms)

0

Node heartbeat consumer time (ms)

0

Master heartbeat consumer time (ms)

0

Identity object check

☒

PDO Tx communication settings

☐ USER DEFINED Mode

☐ SYNC Mode

☐ EVENT Mode

☒ CYCLIC Mode

1000 ms

PDO Rx communication settings

☐ USER DEFINED Mode

☐ SYNC Mode

☒ EVENT Mode

In the **General** tab, you can configure:

- **Node number** (1...122).
- **Node guard period** in ms (default 200 ms). Value 0 disables node guard for this slave. If not zero, it is the interval of node guarding packets sent by the master to the slave.
- **Life time factor** (default 3x). Value 0 disables node guard for this slave. If not zero, multiplied by the **Node guarding period**, it is the maximum amount of time the master waits for the slave answer for the node guard.
- **Boot time elapsed**: this is the maximum amount of time in ms that the master waits for the slave to become pre-operational at boot (default 10 s) before signaling an error.
- **Node heartbeat producer time** in ms, default is value 0 (heartbeat disabled). If not zero, the master enables the heartbeat error handling for this node.

- **Node heartbeat consumer time** in ms, default is value 0 (heartbeat disabled). If not zero, it is the maximum amount of time the slave waits for the heartbeat produced by the master before timing out. This should be greater than the **Heartbeat time** of the master.
- **Master heartbeat consumer time** in ms, default is value 0 (heartbeat disabled); it is the maximum amount of time the master waits for the heartbeat sent by the slave before timing out. This should be greater than the **Node heartbeat producer time**.
- **Identity object check**: when this option is enabled (the default) the master at boot verifies the slave for its identity, verifying that the `Identity` object fields (object 1018 hex) match with EDS default values (Vendor ID, Product code, Revision, Serial). If the option is not enabled, no verification is done.
- **PDO Tx comm settings**: configure here the transmission mode for PDO Tx; depending on the device features (determined from EDS values), not all options may be available.
- **PDO Rx comm settings**: configure here the transmission mode for PDO Rx; depending on the device features (determined from EDS values), not all options may be available.

SDO Set Tab

ATV6X0_V2.1 1.513 CONFIGURATION

GENERAL
SDO SET
PDO TX - INPUT
PDO RX - OUTPUT

+ Add

- Remove

#	Label	Index	SubIndex	Type	Value	Timeout
1	Transmission Type	1800	2	USINT	255	100
2	Event Timer	1800	5	UINT	1000	100
3	Transmission Type	1802	2	USINT	255	100
4	Event Timer	1802	5	UINT	1000	100
5	Transmission Type	1400	2	USINT	255	100
6	Transmission Type	1402	2	USINT	255	100

In this page, you can insert a list of objects and values to send to the slave at boot for configuration purposes, using SDO packets.

Press the **Add** button, choose the objects to send, and then insert their **Value** in the grid.

Some objects are handled automatically, for example the **Transmission type** and **Event timer** are configured automatically depending on the **PDO Tx comm settings** and **PDO Rx comm settings** in the **General tab**.

PDO Tx - Input and PDO Rx - Input Tabs

In the **PDO Tx - Input** tab, you configure the PDOs (Process Data Object) that the slave transmits, and so the master receives in input. In the **PDO Rx - Output** tab, you configure the PDOs that the slave receives, and so the master sends the output.

If the **CAN custom** device was imported with the **Dynamic PDO mapping** enabled, you are able to edit the PDO mapping by adding and removing objects and manually edit the **PDO** and **Bit** columns. Otherwise, the **Add** and **Remove** buttons are not available and you have to use the PDO configuration as-is.

If you check the **Split single bits** option, the object you choose is inserted as single bits to be linked to BOOL variables (that is the default for digital I/O objects in the DS401 standard).

NOTE: This PDO mapping configuration is not sent to the device, its only purpose is to match a configured PDO mapping on the device.

Then with the **Assign** button you can link each CAN object with the **PLC variable** to read (PDO Tx) or write (PDO Rx).

NOTE: It is necessary to rebuild the PLC project with **PROGRAMMING** to update the list of PLC variables.

PDO TX - INPUT tab of **ATV6X0_V2.1.1.513 CONFIGURATION**:

ATV6X0_V2.1 1.513 CONFIGURATION

GENERALSDO SETPDO TX - INPUTPDO RX - OUTPUT

+ Add

— Remove

➔ Assign

➔ UnAssign

#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size	Label	DataBlock
1	6041	0	1	0	0	Statusword	UINT	16		
2	6044	0	1	16	0	Control Effort	INT	16		
3	2061	2a	3	0	0	NM1 (12741)	UINT	16		
4	2061	2b	3	16	0	NM2 (12742)	UINT	16		
5	2061	2c	3	32	0	NM3 (12743)	UINT	16		
6	2061	2d	3	48	0	NM4 (12744)	UINT	16		

PDO RX - OUTPUT tab of ATV6X0_V2.1.1.513 CONFIGURATION:

ATV6X0_V2.1 1.513 CONFIGURATION

GENERALSDO SETPDO TX - INPUTPDO RX - OUTPUT

+ Add

— Remove

➡ Assign

⬅ UnAssign

#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size	Label	Data Block
1	6040	0	1	0	0	Controlword	UINT	16		
2	6042	0	1	16	0	Target Velocity	INT	16		
3	2061	3e	3	0	0	NC1 (12761)	UINT	16		
4	2061	3f	3	16	0	NC2 (12762)	UINT	16		
5	2061	40	3	32	0	NC3 (12763)	UINT	16		
6	2061	41	3	48	0	NC4 (12764)	UINT	16		

CAN Expansion Bus Field - Virtual Master Channels

Overview

This paragraph describes the criteria used by **CONFIGURATION** to assign virtual node IDs due to the network configuration.

Description

When CAN Expansion bus is in use on a M171P device in **Master** mode (field), three master channels are opened.

First master channel is used to process requests that arrive to its physical node ID (the ID assigned by you in the configuration box). Supervisor PC should be connected using this node ID. CAN Expansion bus physical node ID `addr` must be chosen in a range between 1 to 122 or 125.

Two other virtual master channels are opened on this device and are dedicated to the communication with keyboards (max 2 for each CAN Expansion bus network).

Virtual master node IDs have fixed values:

```
ch_1 = 123  
ch_2 = 124
```

Example: M171P + 2 x M171 Display Graphic

The two M171P Display Graphic displays are both connected to the CAN node.

The CAN node has two default virtual channels that can be connected to a maximum of 2 x M171 Display Graphic.

The default virtual channels are 124 for the first display and 123 for the second M171 Display Graphic.

Click **?** button from the CAN node to view the values.

The default address of the Display for **M171 Perf_2** display is 127, the default virtual channel 124 and the default CAN baudrate 500 kb/s.

Thus, when physically connecting an M171 Display Graphic to an M171P with the default settings, upload HMI from the M171 Display Graphic BIOS menu.

In other cases, such as for Display for **M171 Perf_1** (which has the address 126), set the address 126 and the virtual canal 123 from the M171 Display Graphic BIOS menu.

Section 5.9

RS485

What Is in This Section?

This section contains the following topics:

Topic	Page
RS485	118
Using a TM171E27I/O as RS485 Slave	119
Generic Modbus Object Overview	120
Generic Modbus Object Messages	122
Modbus Custom Devices	124
Using a Modbus Custom Device	126

RS485

Description

M171P Display / M171P Flush Mounting has one on-board RS485 port, plus another one available as an external plugin. Each port can be configured as **Not used** (disabled) or **Modbus Master** (field).
M172 has two on-board RS485 ports.

The first port (RS485-1) is used for Modbus Slave - BACnet MS/TP. The RS485-2 port can be configured as **Modbus Slave - BACnet MS/TP** or **Modbus Master** (for field).

RS485 CONFIGURATION

Mode

☐ Modbus Slave – BACnet MS/TP
☒ Modbus Master (for field)

Baud rate

☐ 9600 b/s
☐ 19200 b/s
☒ 38400 b/s
☐ 57600 b/s
☐ 115200 b/s

Serial Mode

E,8,1 (Even parity, 8 data bits, 1 stop bit) ▼

Field

When you configure the RS485 port as **Master** the target acts as a Modbus RTU master on this port. So you can connect Modbus slave devices.

For a Modbus master port, you must configure:

- Baud rate used in this Modbus network (in b/s).
- Serial mode (parity, data bits, stop bits).

To add a Modbus slaves, right-click **RS485** and select **Add** command. You can also drag a Modbus slave from the **Catalog** window to the **RS485** item in the **Resources** window.

After you added and configured the Modbus slaves ([see page 119](#)), you can link the remote objects of the slave and the internal PLC variables to read or write.

The set of PLC objects you can read or write is made of:

- Status variables.
- Field variables declared in **I/O Mapping** → **Field**.

Using a TM171E27I/O as RS485 Slave

Description

In this configuration example, you want to use **Modicon M171 Perf. Expansion 27 I/Os** as expansion of a M171P Display device. The same can be done for other logic controllers.

Configure M171P Display RS485 in **Modbus Master** (for field) mode. From the **Catalog** window, it is possible to select **Modicon M171 Perf. Expansion 27 I/Os** node and drop it on the **RS485** node.

Modicon M171 Perf. Expansion 27 I/Os configuration is similar to a (Modbus Custom device configuration (*see page 126*)). It is possible to assign available **Modicon M171 Perf. Expansion 27 I/Os** dictionary I/O objects to **M171 Perf. Display** PLC variables.

CONFIGURATION knows the **Modicon M171 Perf. Expansion 27 I/Os** dictionary. **Input** and **Output** objects can be added, removed, assigned, unassigned, or changed in position. Only assigned objects are requested by **M171 Perf. Display** device.

GENERAL tab of **M171P EXPANSION 27 I/Os CONFIGURATION**:

M171P EXPANSION 27 I/Os CONFIGURATION

GENERALINPUTOUTPUT

Settings

Modbus address:

(1 ... 247)

Node number:

(0 ... 127)

Polling time:

Ms (0 = continuous read/write on variation)

TimeOut:

ms

Wait before send:

ms

INPUT tab of **M171P EXPANSION 27 I/Os CONFIGURATION**:

M171P EXPANSION 27 I/Os CONFIGURATION

GENERAL**INPUT**OUTPUT

+

Add

−

Remove

↗

Assign

↖

UnAssign

↑

Up

↓

Down

Parameter	Address	Type	Variable	Type	DataBlock
DIL1	1	BOOL			
SW1	9	BOOL			
AIL1	8336	INT			
Counter	8752	UDINT			
Frequency	8754	UDINT			

OUTPUT tab of M171P EXPANSION 27 I/OS CONFIGURATION:

M171P EXPANSION 27 I/OS CONFIGURATION

GENERAL

INPUT

OUTPUT

+ Add

− Remove

➔ Assign

➔ UnAssign

⬆ Up

⬇ Down

Parameter	Address	Type	Variable	Type	DataBlock
DOL1	1	BOOL			
AOL1	8448	UINT			
LED1	8640	USINT			

Generic Modbus Object Overview

Description

The **Generic Modbus** object is a generic Modbus slave that can be inserted under the RS485 port of the logic controller, when configured as **Modbus master**.

GENERIC MODBUS NODE

GENERAL

Settings

Name:

Generic Modbus_1

Modbus address:

1

(0 ... 247, 0=broadcast)

Node number:

1

(0 ... 127)

To add a **Generic Modbus** object:

Step	Action
1	In the Resources window, click RS485 .
2	• Drag Generic Modbus from the Catalog window to RS485 item in the Resources window. • Or you can right-click RS485 and select Add command. Generic Modbus appears under RS485 item.

You can use the **Generic Modbus** when you want to configure manually and have full control over the single Modbus messages to send to the slave.

Another typical usage is for third-party devices that you plan to use once in your project, and you do not want to put in the catalog for future reuse.

In the main page of the **Generic Modbus** you can configure:

- A name for the object in the project.
- Its Modbus address (in the range 1...247).
- Its Node number (node ID)(in the range 0...127): this value is incremented automatically, and can be used in the PLC program to index the `SysMbMRtuNodeStatus[]` array that contains diagnostic information about each slave node.

Generic Modbus Object Messages

Description

The **Generic Modbus** object alone does nothing; you have to add under it one or more **Modbus messages** that are specific Modbus function requests that are sent on the bus.

To add a function to a **Generic Modbus** object, drag a Modbus function from the **Catalog** window to **Generic Modbus** item in the **Resources** window. You can also right-click **Generic Modbus** and select **Add** command.

Catalog			
Device name	Version	Description	
 Modbus FC-01	1	Read Coils - Function 01 (0x01)	
 Modbus FC-02	1	Read Discrete Inputs - Function 02 (0x02)	
 Modbus FC-03	1	Read Holding Register - Function 03 (0x03)	
 Modbus FC-04	1	Read Input Registers - Function 04 (0x04)	
 Modbus FC-05	1	Write Single Coil - Function 05 (0x05)	
 Modbus FC-06	1	Write Single Register - Function 06 (0x06)	
 Modbus FC-15	1	Write Multiple Coils - Function 15 (0x0F)	
 Modbus FC-16	1	Write Multiple Register - Function 16 (0x10)	

Function		Description	Details	Objects length
1	0x01	Read Coils	Reads one or more read-only discrete output coils	1-bit
2	0x02	Read Discrete Inputs	Reads one or more read-only digital inputs	1-bit
3	0x03	Read Holding Registers	Reads one or more read/write registers	16-bit
4	0x04	Read Input Registers	Reads one or more read-only registers	16-bit
5	0x05	Write Single Coil	Writes one discrete output coil	1-bit
6	0x06	Write Single Register	Writes single analog output holding register	16-bit
15	0x0F	Write Multiple Coils	Writes one or more digital outputs	1-bit
16	0x10	Write Multiple registers	Writes one or more registers	16-bit

The messages are processed in the order they are inserted in the tree.

General Tab

Select the Modbus function added to the tree to display its configuration window:

MODBUS FC

GENERAL

Settings

Start address:

0

(1 ... 65536)

Polling time:

0

ms (0 = Continuous Read)

Time out:

1000

ms

Wait before send:

10

ms

For each message, in its **General** tab you can configure:

- **Start address:** address of the first Modbus object to read or write (1...65536).
- **Polling time:** the message is processed with this period (ms):
 - For writing operations: value 0 means to write it only on variation of the value.
 - For reading operations: value 0 means maximum speed.
- **Time out:** the operation is unsuccessful when this time-out expires (ms).
- **Wait before send:** this is the wait time before sending another request to the salve device.

Coils Tab

Beside the **General** tab, each different message has a second tab where you can configure the list of objects to read or write.

MODBUS FC 01(0X01) – READ COILS

GENERAL
COILS

+ Add

- Remove

➡ Assign

↩ UnAssign

#	Name	ObjType	Label	Address	DataBlock	Description
1	Coil	Bool		0		

Using the **Add** button, insert one row for each Modbus object to read or write, up to 16 elements. The first row has the address configured in the **Address** box in the **General** tab, and the other rows increment and follow.

For each row, press the **Assign** button to choose the PLC object to link and to be read or written with this Modbus message; you cannot leave unassigned rows in the message.

NOTE: Remember to rebuild the PLC project to see an updated list of PLC variables here.

Modbus Custom Devices

Description

You can create and edit **Modbus custom** devices directly.

Therefore, you can use in your project and add in the catalog for future reuse any third-party Modbus slave, characterizing its Modbus map only the first time and simplifying its further use, because you do not have to care about Modbus messages and functions anymore.

Creating a New Modbus Custom Device

To create a new **Modbus custom** device, choose **Developer → Run Modbus custom Editor**; the external **ModbusCustomEditor** tool is launched, with a new empty document.

ModbusCustomEditor

MODBUS CUSTOM EDITOR

New

Open

Save

Device Version

Name:

ModbusCustomDeviceName

Version:

1.0

Description:

Modbus custom device description

Modbus RTU

☒

Modbus TCP

☐

Device Info

Max message size (bit):

2000

Max message size (reg.):

120

Enable overlap of Bit and Reg maps

☐

+ Add

- Remove

↑ Up

↓ Down

#	Address	Label	Type	Read only	Modbus type	Description
1	1	Label1	INT	False	Holding Register (16 bit)	

Here you can configure:

- Name of the device.
- Long description for the device.
- A version number.
- Overlapping of bit and register maps: check this if the device has both a **0** register and a **0** bit (in other words it has different addressing of 16-bit and 1-bit objects). Uncheck this if the address is unique and so duplicated are not allowed, even if the type is different.
- Max message size: insert here the maximum number of registers per message supported by the device.

Then, using the **Add** button, add one row for each Modbus object of the device. You have to insert its address, name, type (note that **Type** and **Read only** columns are linked with the **Modbus type** column) and optionally a long description.

When you finish, save the current device definition; you are prompted for a file name with **.PCT** extension, by default it is proposed the current name+version.

The file is saved in the special **ModbusCustom** folder in the catalog; now you can close the **ModbusCustomEditor** and go back in **CONFIGURATION** to use the new device.

Editing an Existing Modbus Custom Device

To edit an existing **Modbus custom** device, you can:

- Run the **ModbusCustomEditor** with the **Developer → Run Modbus custom Editor** command, and then manually open the PCT file with the standard **File → Open** command.
- When the device you want to edit is visible in the **Catalog** window (for example when an RS485 node is selected and is in **Master** mode), you can right-click on it and choose the **Edit device** command; the **ModbusCustomEditor** is launched and the selected device opened.

Deleting a Modbus Custom Device

To delete an existing **Modbus custom** device when the device is visible in the **Catalog** window, do a right-click on it and choose **Delete from catalog**.

Using a Modbus Custom Device

Description

When you insert the **Modbus custom** device as a Modbus slave (for example under an RS485 port) and click it on the **Resources** window, the Editor window displays three tabs.

General Tab

In the **General** tab you can configure:

- Its **Modbus address** (in the range 1...247).
- Its **Node number** (in the range 0...127); this value is incremented automatically, and can be used in the PLC program to index the `SysMbMRtuNodeStatus[]` array that contains diagnostic information about each slave node.
- **Polling time**: the Modbus messages are processed with this period (ms); for writing operations, value 0 means to write it only on variation of the value, for reading operations value 0 means maximum speed.
- **Timeout**: the operation is unsuccessful when this time-out expires (ms).
- **Wait before send**: this is an additional timeout (to be used with slow slaves that do not answer if the messages are sent too fast).

GENERAL tab of **MODBUS CUSTOM CONFIGURATION**:

MODBUS CUSTOM CONFIGURATION			
GENERAL	INPUT	OUTPUT	
Settings	Modbus address:	1	(1 ... 247)
	Node number:	2	(0 ... 127)
	Polling time:	0	Ms (0 = continuous read/write on variation)
	TimeOut:	1000	ms
	Wait before send:	10	ms

Here you can note that for **Modbus custom** the **Polling time**, **Timeout**, and **Wait before send** are generic for the whole device while for the **Generic Modbus** you can put specific different values for each single message. This is because with the **Modbus custom** the low-level Modbus messages are automatically calculated. However, you cannot “fine-tune” them because these settings are global.

Input and Output Tabs

In the **Input** and **Output** tabs, you can insert one row for each Modbus object to read or write. Press the **Add** button and choose the parameters to exchange (multi-selection is supported). Use the **Assign** button to link them to the PLC object to be read or written to.

INPUT tab of **MODBUS CUSTOM CONFIGURATION**:

MODBUS CUSTOM CONFIGURATION

GENERAL **INPUT** OUTPUT

+ Add

– Remove

➔ Assign

↶ UnAssign

⬆ Up

⬇ Down

Parameter	Address	Type	Variable	Type	DataBlock
Label1	1	INT			

OUTPUT tab of **MODBUS CUSTOM CONFIGURATION**:

MODBUS CUSTOM CONFIGURATION

GENERAL INPUT **OUTPUT**

+ Add

– Remove

➔ Assign

↶ UnAssign

⬆ Up

⬇ Down

Parameter	Address	Type	Variable	Type	DataBlock
Label1	1	INT			

Insert in the **Input** tab the Modbus objects to **READ** from the Modbus slave (and to put into PLC variables). Insert in the **Output** tab the Modbus objects to **WRITE** to the Modbus slave (and to get from the PLC variables).

NOTE: Remember to rebuild the PLC project with Application to see an updated list of PLC variables here.

CONFIGURATION creates the correct Modbus messages analyzing the sequence of addresses and types. If the addresses are consecutive and the types are homogeneous, different objects are grouped in single messages to optimize the communication.

The maximum number of registers configured with the **ModbusCustomEditor** is also considered, along with the maximum number of registers per message of the master (16 for the M171P/M172).

The grouping and generation of the Modbus messages is automatic and recalculated at each compilation.

Section 5.10

Ethernet

Ethernet

Description

M171P Flush Mounting / M172P are provided with an integrated Ethernet port.

M171P/M172O can have one Ethernet port, available as an external communication module.

The Ethernet port can be configured as a Modbus TCP in two ways:

- **Server only:** the controller supports communication from other controller requests.
- **Client/Server:** the controller supports Modbus TCP communication from other controller requests as well as making requests to other controllers.

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.

Configuration

To configure the Ethernet port:

- Select **Client/Server** or **Server only**.
- Enter the additional Modbus TCP sockets (0 by default).

The IP address is stored in **Modbus objects** → **BIOS Parameters**.

Client/Server

The **Client/Server** configuration allows the controller to send requests and to read responses from or to other devices connected on the same Ethernet network.

You can attach Modbus devices and exchange data.

You can add generic Modbus devices (*see page 120*), or custom devices created with the Modbus custom Editor (*see page 124*) in the same way as for RS485.

After you added and configured the Modbus nodes, you can add Generic Modbus Objects Messages (*see page 122*) to define the `READ` or `WRITE` functions.

The set of controller objects you can send or receive is made of:

- **EEPROM Parameters** (non-volatile memory parameters)
- **Status variables**

The configuration page for the **Binding** object in Modbus TCP is the same as the CAN Expansion bus. Refer to chapter CAN Expansion bus - Binding (*see page 103*) for a description and usage of this page.

MODBUS FC

GENERAL

Settings	Start address:	0	(1 ... 65536)
	Polling time:	0	ms (0 = Continuous Read)
	Time out:	1000	ms

The only difference from CAN Expansion bus Binding is that here you have one more column named **Timeout**, where you can configure the specific time-out in ms for each object exchanged.

Section 5.11

Plugins

Communication Modules Range Overview

Overview

The communication modules (plugins) that can be added are displayed in the **Catalog** window.
Drag the communication module into the **Plugins** element.

Example of RS232 configuration window:

RS232 CONFIGURATION

Mode	☐ Modbus Slave ☒ Modbus Master (for field)
Baud rate	☐ 9600 b/s ☐ 19200 b/s ☒ 38400 b/s ☐ 57600 b/s ☐ 115200 b/s
Serial Mode	E,8,1 (Even parity, 8 data bits, 1 stop bit)

Example of Profibus DP configuration window:

PROFIBUS DPV0 CONFIGURATION

GENERAL PB MST TX - INPUT PB MST RX - OUTPUT PI DIAG TX - INPUT PI DIAG RX - OUTPUT

Profibus Station Settings	DP Address (1...126)	1
	Identification Nr.	65535
	Consistency	☐
	Word Swap	☒
	Service Mode Enabled	☐

Communication Modules References

Reference	Description	Terminal type	Compatible controllers
TM171ACAN	CAN	2 screw terminal blocks	M172P M172O M171P ⁽¹⁾
TM171ALON	LonWorks	1 screw terminal block	
TM171AMB	Modbus SL (RS-485)	2 screw terminal blocks	
TM171ARS232	RS232 serial link, Relay output	1 SUB-D 9 1 screw terminal block	
TM171ARS485	Modbus SL, and BACnet MS/TP	2 screw terminal blocks	M172O M171P ⁽¹⁾
TM171AETH	Ethernet, Modbus TCP, and BACnet/IP	1 RJ45	
TM171AETHRS485	Ethernet, Modbus TCP, BACnet/IP, Modbus SL, and BACnet MS/TP	1 RJ45 2 screw terminal blocks	
TM171APBUS	PROFIBUS	1 SUB-D 9	M171P ⁽¹⁾
⁽¹⁾ Not applicable to M171P Flush Mounting			

For further information about communication modules, refer to the Modicon M171A Communication Modules Instruction Sheet [EAV96007](#).

Chapter 6

Technical Reference

Section 6.1

Modbus Protocol

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	135
Data Types	136
Function Codes	137

Overview

Introduction

The transmission mode used is RTU. The frame does not contain message header and end of message bytes.

Slave address	Request code	Data	CRC16
---------------	--------------	------	-------

The data is transmitted in binary code.

CRC16: cyclic redundancy check.

The end of the frame is detected on a silence greater than or equal to three characters.

Principle

Only one device can transmit on the line at a time.

The master manages the exchanges and only it can take the initiative.

It interrogates each of the slaves in succession.

No slave can send a message unless it is asked to do so.

The master repeats the question when there is an incorrect exchange, and declares the interrogated slave unavailable if no response is received within a given time period.

If a slave does not receive a message, it sends an exception response to the master. The master may or may not repeat the request.

Direct slave-to-slave communications are not possible.

For slave-to-slave communication, the application software must therefore be designed to interrogate a slave and send back data received to the other slave.

The 2 types of dialogue are possible between master and slaves:

- The master sends a request to a slave and waits for its response
- The master sends a request to all slaves without waiting for a response (broadcasting principle)

Addresses

Address specification:

- The device Modbus address can be configured from 1 to 247.
- Address 0 coded in a request sent by the master is reserved for broadcasting. Slave devices take account of the request, but do not respond to it.

Data Types

Description

Information is stored in the Slave device in four different types: two types are on/off discrete values (coils and contacts) and two are numerical values (registers).

- Discrete Input Contacts (read only), 1-bit.
- Discrete Output Coils (read/write), 1-bit.
- Analog Input Registers (read only), 16-bit.
- Analog Output Holding Registers (read/write), 16-bit.

To handle more complex data types (like 32-bit integers or floating point), you have to use two or more consecutive registers and read or write them together.

Function Codes

Description

The Modbus protocol specifies different “function codes” for each Modbus message:

- 01 (01 hex): Read Discrete Output Coils.
- 05 (05 hex): Write single Discrete Output Coil.
- 15 (0F hex): Write multiple Discrete Output Coils.
- 02 (02 hex): Read Discrete Input Contacts.
- 04 (04 hex): Read Analog Input Registers.
- 03 (03 hex): Read Analog Output Holding Registers.
- 06 (06 hex): Write single Analog Output Holding Register.
- 16 (10 hex): Write multiple Analog Output Holding Registers.

Part III

Programming

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
7	The PROGRAMMING Tab	141
8	Project Options	163
9	Managing Project Elements	181
10	Editing the Source Code	225
11	Compiling	269
12	Launching the Application	277
13	Debugging	293
14	Language Reference	371
15	Errors Reference	445

Chapter 7

The PROGRAMMING Tab

Section 7.1

Overview

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview of the Programming Window	143
Menu Bar	147
Toolbars	155
Status Bar	162

Overview of the Programming Window

Start-up and Test

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a verification be made and that enough time is allowed to perform complete and satisfactory testing.

WARNING

UNINTENDED EQUIPMENT OPERATION

- Verify that the installation and set up procedures have been completed.
- Before operational tests are performed, remove the blocks or other temporary holding means used for shipment from the component devices.
- Remove tools, meters, and debris from equipment.

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

General Description

The **PROGRAMMING** work environment has various windows for developing the controller application (for example programming in IEC 61131-3 compatible languages), testing, debugging, and controller application downloading to the target device.

In the **PROGRAMMING**, you have two download possibilities:

- Download only the controller application by clicking  **On-line → Download code**.
- Download the project (which includes the controller application, the BIOS and PLC parameters,

and their value by default) by clicking  **Download all** icon in the project toolbar.

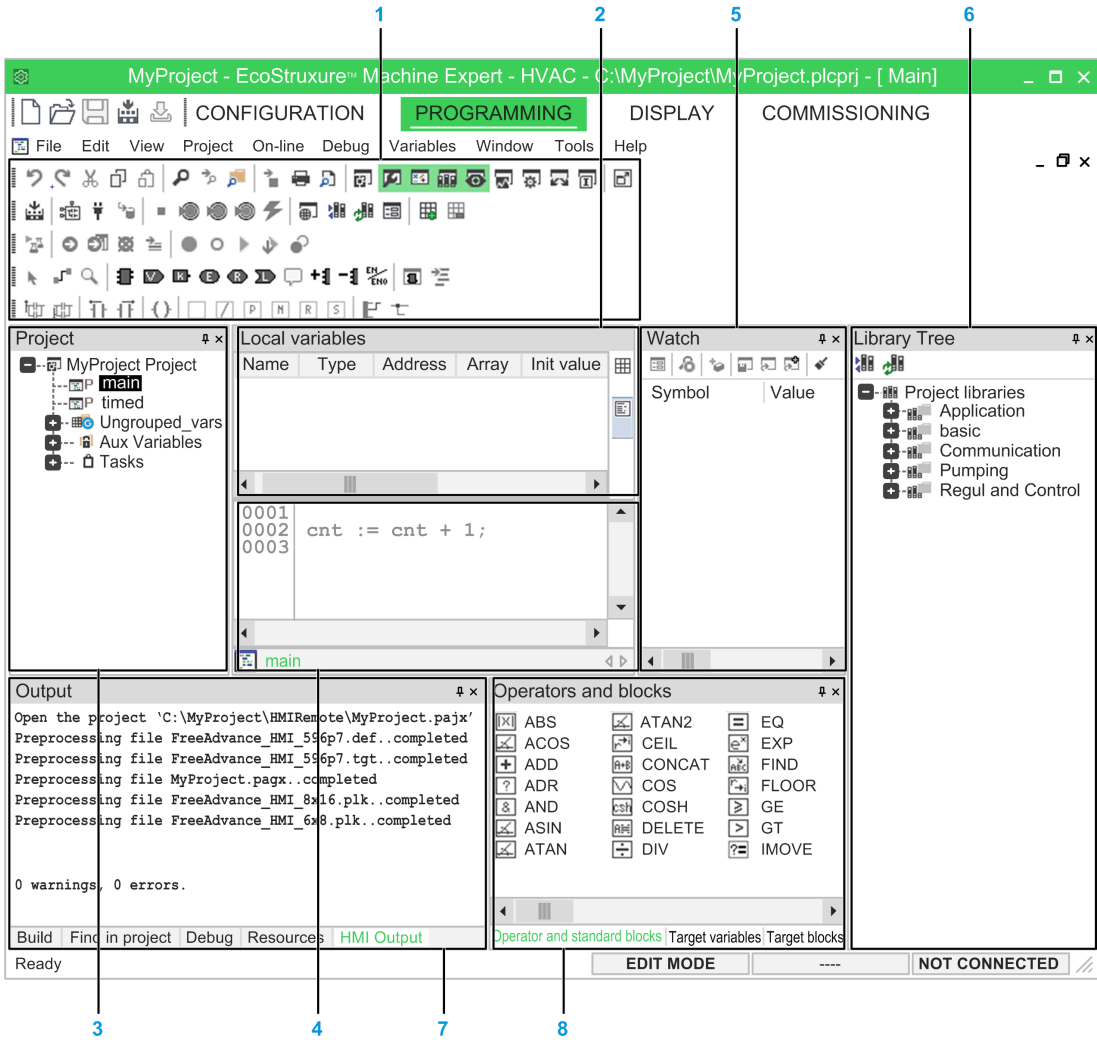
WARNING

AUTOMATIC RESTART OF CONTROLLER

- Do not download your application without first accessing the state of your machine or process.
- Do not download your application without first ascertaining that there is no risk of injury to anyone in or around your machine or process.

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

The windows are listed below:



Marker	Description	
1	Toolbars	Many functions available in menus are displayed here in form of icons in toolbars. These icons help you to create the application. The most used are in Main and Project toolbars. For information about how to manage the toolbars, refer to Toolbars (<i>see page 43</i>).
2	Local variables window	The global and local variables of the code displayed in the source code editor (programs, function blocks, and functions) appear here. For more information about how to use the Local variables window, refer to Variables (<i>see page 188</i>).
3	Project window	This window enables you to: • Manage the application code. • Manage and define complex variables defined by you. • Manage the target device menu. For information about how to use the Project window, refer to Project Window (<i>see page 182</i>).
4	Source code editor	This window enables you to manage, edit, and use file/print source to write in any of the five programming languages defined by the IEC 61131-3 standard. The definition of both global and local variables is supported by specific spreadsheet-like editors. For information about how to use the source code editor, refer to Editing the Source Code (<i>see page 225</i>).
5	Watch window	This window enables you to manage variables debugging by displaying their status in numerical format when the application is running and connected to the target device. For more information about how to use the Watch window, refer to Watch Window (<i>see page 295</i>).
6	Library Tree window	The Library Tree window contains a set of different library objects. Each object is grouped into the folder to which it belongs. These folders are useful to group the library elements logically. To display the properties of an object, right-click its name and select Object Properties command. The Properties Window appears and displays its properties. For information about how to manage libraries, refer to Working with Libraries (<i>see page 174</i>).
7	Output window	This tool window shows the messages relating to the development of the project. For more details, refer to Output window description (<i>see page 146</i>).
8	Operators and blocks window	This tool enables you to manage default function libraries or function libraries created by you. The window is divided into various tabs, one for each library. The following tabs are always available: • Operator and standard blocks: operators (AND, OR, and so on). • Target variables: specific variables of the target device. • Target blocks: specific functions of the target device. Additional tabs are managed by using the menu Project → Library manager.

Output Window

This tool window shows the messages relating to the development of the project.

The window is divided into five tabs:

- **Build:** information related to the file opening, compilation errors, and downloading code to a device.
- **Find in project:** result of the Find in project activity.
- **Debug:** information about debugging activities (for example breakpoints).
- **Resources:** messages related to the target device.
- **HMI Output:** messages related to **DISPLAY** activity.

NOTE: The connection to the target device is also visible in the status bar (*see page 162*).

Menu Bar

Overview

In the following tables, you can see the list of the commands of **PROGRAMMING**. However, since **PROGRAMMING** has a multi-document interface (MDI), you may find some disabled commands or even some unavailable menus, depending on what kind of document is currently active.

File Menu

Command	Icon	Key	Description
New project		-	Creates a new project (<i>see page 53</i>).
Open project		-	Opens an existing project (<i>see page 59</i>).
Import project from target	-	-	Imports sources project from target device.
View project (read only)	-	-	Opens an existing project in read-only mode.
Save project		-	Saves the current open project. (<i>see page 57</i>)
Save project As ...	-	-	Saves the current open project specifying new name, location and extension (<i>see page 57</i>).
Close project	-	-	Closes the open project (<i>see page 59</i>).
New text file	-	-	Opens a blank new generic text file.
Open file	-	Ctrl+O	Opens an existing file, whatever its extension. The file is displayed in the text editor. Anyway, if you open a project file, you currently open the project it refers to.
Save	-	Ctrl+S	Saves the document of the currently active window.
Close	-	-	Closes the document of the currently active window.
Options...	-	-	Opens the Program options dialog box (<i>see page 47</i>).
Print...		Ctrl+P	Prints the document of the currently active window (<i>see page 56</i>).
Print preview		-	Creates a preview of the document of the currently active window, ready to be printed.
Print Project	-	-	Prints the documents making up the project.
Printer Setup	-	-	Opens the Printer setup dialog box.
..recent..	-	-	Lists a set of recently opened project file.
Exit	-	-	Closes EcoStruxure Machine Expert - HVAC.

Edit Menu

Command	Icon	Key	Description
Undo		Ctrl+Z	Cancels last action made.
Redo		Ctrl+Y	Restores the last action canceled by Undo.
Cut		Ctrl+X	Removes the selected items from the active document and stores them in a system buffer.
Copy		Ctrl+C	Copies the selected items to a system buffer.
Paste		Ctrl+V	Pastes in the active document the contents of the system buffer.
Delete line	-	Ctrl+E	Deletes the whole source code line.
Go to symbol		Shift+F12	Goes to variable declaration.
Find in project		Ctrl+Shift+F	Opens the Find in project dialog box.
Bookmarks...	-	-	
Add/toggle	-	Ctrl+F2	Adds a bookmark to mark lines. If a bookmark is already defined, removes it.
Next	-	F2	Goes to next defined bookmark.
Prev	-	Shift+F2	Goes to previous defined bookmark.
Remove all	-	-	Removes the defined bookmarks.
Go to line	-	Ctrl+G	Allows you to move to a specific line in the source code editor.
Find...		Ctrl+F	Asks you to type a string and searches for its first instance within the active document from the current location of the cursor.
Find next		F3	Iterates between the results of the research, found by the Find command.
Replace	-	Ctrl+H	Allows you to automatically replace one or all the instances of a string with another string.
Insert/Move mode		-	Toggle between those two editing modes, used to insert or move blocks.
Connection mode		-	Editing mode which allows you to draw logical wires to connect pins.
Watch mode		-	Editing mode which allows you to add variables to any debugging tool.

View Menu

Command	Icon	Key	Description
Toolbars	-	-	Refer to Toolbars (<i>see page 43</i>).
Main	-	-	Displays or hides the Main toolbar.
Project	-	-	Displays or hides the Project bar.
Debug	-	-	Displays or hides the Debug toolbar.
FBD Bar	-	-	Displays or hides the FBD toolbar.
SFC Bar	-	-	Displays or hides the SFC toolbar.
LD Bar	-	-	Displays or hides the LD toolbar.
Network	-	-	Displays or hides the Network toolbar.
Configuration	-	-	Displays or hides the Configuration bar.
HMI Page	-	-	Displays or hides the HMI Page bar.
HMI Project	-	-	Displays or hides the HMI Project bar.
HMI Profiles	-	-	Displays or hides the HMI Profiles bar.
Commissioning	-	-	Displays or hides the Commissioning bar.
Tool windows	-	-	Refer to Tool Windows Management (<i>see page 44</i>).
Local variables	-	-	Displays or hides the Local variables window.
Project	-	-	Displays or hides the Project window.
Watch	-	-	Displays or hides the Watch window.
Properties Window	-	-	Displays or hides the Properties Window window.
Oscilloscope	-	-	Displays or hides the Oscilloscope window.
PLC run-time status	-	-	Displays or hides the PLC run-time status bar.
Operators and blocks	-	-	Displays or hides the Operators and blocks window.
Library Tree	-	-	Displays or hides the Library Tree window.
Output	-	-	Displays or hides the Output window.
Cross Reference	-	-	Displays or hides the Cross Reference window.
Resources	-	-	Displays or hides the Resources window.
Catalog	-	-	Displays or hides the Catalog window.
HMI Project	-	-	Displays or hides the HMI Project window.
HMI Properties	-	-	Displays or hides the HMI Properties window.
HMI Actions	-	-	Displays or hides the HMI Actions window.
HMI Vars and Parameters	-	-	Displays or hides the HMI Vars and Parameters window.
HMI Templates	-	-	Displays or hides the HMI Templates window.
Commissioning	-	-	Displays or hides the Commissioning window.

Command	Icon	Key	Description
Full screen		Ctrl+U	Expands the currently active document window to fill entire screen (<i>see page 46</i>). (Esc to exit from this mode).
Grid	-	-	Displays or hides a dotted grid in the background of graphical source code editors.
Show comments for objects	-	-	Displays or hides comments for individual objects, not only for networks. Only for LD editor.

Project Menu

Command	Icon	Key	Description
New object	-	-	
New program	-	-	Creates a new program. A dialog is prompted in order to specify the new program properties.
New function block	-	-	Creates a new function block. A dialog is prompted in order to specify the new function block properties.
New function	-	-	Creates a new function. A dialog is prompted in order to specify the new function properties.
New variable	-	-	
Automatic	-	-	Creates a new automatic variable. A dialog is prompted in order to specify the new variable properties.
Mapped variable	-	-	Creates a new mapped variable. A dialog is prompted in order to specify the new variable properties.
Constant	-	-	Creates a new constant. A dialog is prompted in order to specify the new constant properties.
Retain	-	-	Creates a new retain variable. A dialog is prompted in order to specify the new variable properties.
New definition	-	-	
Enumeration	-	-	Create a new enumeration.
Interface	-	-	Define an abstract interface that function blocks can implement.
Structure	-	-	Create a new Structure.
Subrange	-	-	Create a new Subrange.
TypeDef	-	-	Create a new TypeDef.
Copy object	-	-	Copies the object currently selected in the Workspace .
Paste object	-	-	Pastes the previously copied object.
Duplicate object	-	-	Duplicates the object currently selected in the Workspace , and asks you to type the name of the copy.
Delete object	-	Delete	Deletes the currently selected object.

Command	Icon	Key	Description
View PLC Object properties	-	Alt+Enter	Displays properties and description of the selected object.
Object Browser		-	Opens the Object browser, which lets you navigate between objects.
Compile		F7	Launches the compiler.
Recompile all	-	Ctrl+Alt+F7	Recompiles the project.
Generate redistributable source module (RSM)	-	-	Generates an RSM file.
Import objects	-	-	Lets you import an object from a library.
Export objects to library	-	-	Lets you export an object to a library.
Library manager		-	Opens the Library manager.
Refresh all libraries		-	Reloads the libraries linked to the project.
Macros	-	-	
New macro	-	-	Creates a new macro. A dialog is prompted in order to specify the new macro properties.
Copy macro	-	-	Copies the selected macro creating a new one.
Delete macro	-	-	Deletes the selected macro.
Properties	-	-	Displays the selected macro properties.
Select target...	-	-	Lets you select a new target for the project.
Refresh current target	-	-	Lets you update the target file for the same version of the target.
Options...	-	-	Opens the project options dialog.

On-line Menu

Command	Icon	Key	Description
Set up communication...	-	-	Lets you set the properties of the connection to the target (<i>see page 280</i>).
Connect		-	Starts the communication with the device (<i>see page 280</i>).
Download code		F5	Download the configuration and PLC application of the project to the device (<i>see page 289</i>).
Download options...	-	-	Lets you set the properties of the source code downloaded to the target.

Command	Icon	Key	Description
Force image upload	-	-	If the target device is connected, lets you upload the image file.
Force debug symbols upload	-	-	If the target device is connected, lets you upload the debug symbols file.
Halt		-	Stops the PLC execution.
Cold restart		-	Restarts the PLC execution and both retain and non-retain variables are reset.
Warm restart		-	Restarts the PLC execution and non-retain variables are reset.
Hot restart		-	Restarts the PLC execution without any reset on variables.
Reboot target		-	Reboots the target.
Read all logs again	-	-	Reloads the remote logs from target.

Debug Menu

Command	Icon	Key	Description
Simulation mode		-	Open/close the integrated simulation environment.
Start/Stop watch value		-	Starts or stops (toggle) the evaluation of the symbols added in the watch window.
Add symbol to watch	-	F8	Adds a symbol to the Watch window.
Inserts new item into watch		Shift+F8	Inserts a new item into the Watch window.
Add symbol to a debug window	-	F10	Adds a symbol to a debug window.
Inserts new item into a debug window	-	Shift+F10	Inserts a new item into a debug window.
Quick watch symbol	-	F11	View and force the value of the selected symbol.
Live debug mode		-	If debug mode is running, starts or stops (toggle) the live debug mode.
Add/remove text trigger		F9	Adds/removes a text trigger.
Add/remove graphic trigger		Shift+F9	Adds/removes a graphic trigger.

Command	Icon	Key	Description
Remove all triggers		Ctrl+Shift+F9	Removes the active triggers.
Trigger list		Ctrl+I	Lists the active triggers.
Run		-	Restarts program after a breakpoint is hit.
Add/Remove breakpoint		F12	Adds or removes a breakpoint.
Remove all breakpoints		-	Removes the active breakpoints.
Breakpoint list		-	Lists the active breakpoints.
Change current instance	-	-	Changes the current function block instance (live debug mode).

Variables Menu

Command	Icon	Key	Description
Add	-	-	
Automatic	-	-	Creates a new automatic variable. A dialog is prompted in order to specify the new variable.
Mapped variable	-	Ctrl+Shift+M	Creates a new mapped variable. A dialog is prompted in order to specify the new variable.
Constant	-	-	Creates a new constant. A dialog is prompted in order to specify the new constant.
Retain	-	-	Creates a new retain variable. A dialog is prompted in order to specify the new variable.
Insert	-	Ctrl+Shift+Ins	Adds a new row to the grid in the currently active editor.
Delete	-	Delete	Deletes the variable in the selected row of the currently active table.
Create multiple...	-	-	Lets you create a set of multiple variables.
Group	-	-	Opens a dialog box which lets you create and delete groups of variables.

Window Menu

Command	Icon	Key	Description
Cascade	-	-	Displaces the open documents in cascade so that they completely overlap except for the caption.
Tile	-	-	The PLC editors area is split into frames having the same dimensions, depending on the number of currently open documents. Each frame is automatically assigned to one of such documents.
Arrange Icons	-	-	Displaces the icons of the minimized documents in the bottom left-hand corner of the PLC editors area.
Close all	-	-	Closes the open documents.
Windows...	-	Alt+W	Opens a Windows List browser.

Tools Menu

Command	Icon	Key	Description
Custom tools	-	-	Displays up to 16 user defined commands (<i>see page 48</i>).

Help Menu

Command	Icon	Key	Description
Index	-	-	Lists the Help keywords and opens the related topic.
Context	-	F1	Context-sensitive help. Opens the topic related to the currently active window.
Registration	-	-	Register the software (<i>see page 26</i>).
About...	-	-	Credits and version information.

Toolbars

Introduction

The toolbars appear at the top of the EcoStruxure Machine Expert - HVAC window to provide access to frequently used functions.

The description of commands is displayed in the lower left corner of the main window when you place the mouse tracker on the command icon.

Main Toolbar

The main toolbar has the following buttons:

Icon	Description	Shortcut
	Undo Click once to undo the most recent action in the program editor. Click the down arrow and select an action from the list to undo all actions up to and including the selected action. You can undo up to 10 actions.	Ctrl+Z
	Redo Click once to cancel the most recent Undo action. Click the down arrow and select an action from the list to redo all actions up to and including the selected action. You can redo up to 10 actions.	Ctrl+Y
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Find	Ctrl+F
	Find next	F3
	Find in project	Ctrl+Shift+F
	Go to symbol	Shift+F12
	Print Print the content of the current editor window.	Ctrl+P

Icon	Description	Shortcut
	Print preview	-
	Workspace	Ctrl+W
	Output	Ctrl+R
	Operators and blocks	Ctrl+L
	Show or hide Library Tree bar	
	Watch	Ctrl+T
	Oscilloscope	Ctrl+K
	PLC run-time status bar	-
	Cross reference window	-
	Object Properties window	-
	Full screen	Ctrl+U

Project Toolbar

The project toolbar has the following buttons:

Icon	Description	Shortcut
	Compile	F7
	Simulation mode	-
	Connects to the target	-
	Code download	F5

Icon	Description	Shortcut
	Halt	-
	Cold restart	-
	Warm restart	-
	Hot restart	-
	Reboot target	-
	Object browser	-
	Library manager	-
	Refresh all libraries	-
	Object properties	-
	Insert record	-
	Delete record	-

Debug Toolbar

The debug toolbar has the following buttons:

Icon	Description	Shortcut
	Live debug mode	-
	Add/remove trigger	F9
	Add/remove graphic trigger	Shift+F9
	Remove all triggers	Ctrl+Shift+F9

Icon	Description	Shortcut
	Triggers list	Ctrl+I
	Add/remove a breakpoint	F12
	Remove all breakpoints	-
	Run	-
	Step	-
	Breakpoint list	-

FBD Toolbar

The FBD toolbar has the following buttons:

Icon	Description	Shortcut
	Move/Insert	-
	Connection	-
	Watch	-
	New block	-
	Variable	-
	Constant	-
	Expression	-
	Return	-
	Jump	-

Icon	Description	Shortcut
	Comment	-
	Increase pins	Ctrl++
	Decrease pins	Ctrl+-
	Add Enable Input/Output pins	-
	FBD properties	-
	View source	-

SFC Toolbar

The **SFC** toolbar has the following buttons:

Icon	Description	Shortcut
	New step	-
	New transition	-
	New jump	-
	Add a step and a transition above the selected step	-
	Add a step and a transition below the selected step	-
	Add pin to divergent position	-
	Remove pin from divergent position	-
	Add pin to convergent position	-
	Remove pin from convergent position	-
	Add pin to simultaneous divergent transition	-

Icon	Description	Shortcut
	Remove pin from simultaneous divergent transition	-
	Add pin to simultaneous convergent transition	-
	Remove pin from simultaneous convergent transition	-
	Remove space before rightmost pin	-
	Add space before rightmost pin	-
	Move a transition up	-
	Move a transition down	-
	New action	-
	New transition code	-

LD Toolbar

The LD toolbar has the following buttons:

Icon	Description	Shortcut
	Parallel contact before	-
	Parallel contact after	Shift+P
	Serie contact before	-
	Serie contact after	Shift+C
	Coil	Shift+O
	Open object	O

Icon	Description	Shortcut
	Negated object	C
	Positive object	P
	Negative object	N
	Reset object	R
	Set object	S
	Set output line	-
	New branch	-

Network Toolbar

The **Network** toolbar has the following buttons:

Icon	Description	Shortcut
	Insert top	-
	Insert bottom	-
	Insert after	-
	Insert before	-
	View grid	-
	Auto connect	-

Status Bar

Overview

The status bar is located at the bottom right of the EcoStruxure Machine Expert - HVAC window. It indicates the state of communication and the status of the application currently executing on the target device.



For more information, refer to:

- Edit and Debug Mode ([see page 321](#))
- Connection Status ([see page 287](#))
- Application Status ([see page 288](#))

Chapter 8

Project Options

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
8.1	Project Options	164
8.2	Working with Libraries	174

Section 8.1

Project Options

General Information

You can edit significant project options choosing **Project → Options...**

What Is in This Section?

This section contains the following topics:

Topic	Page
General Tab	165
Code Generation Tab	166
Build Output Tab	168
Debug Tab	170
Build Events Tab	171
Cross Reference Tab	172

General Tab

Overview

General tab of the **Project options** window:

The screenshot shows the 'Project options' dialog box with the 'General' tab selected. The dialog has a green title bar with the text 'Project options' and a close button. Below the title bar are four tabs: 'Download', 'Debug', 'Build events', and 'Cross Reference'. The 'General' tab is active, showing a 'Project info' section with four text input fields: 'Project' (containing 'MyProject'), 'Version' (empty), 'Author' (empty), and 'Note' (empty). To the right of the 'Project' and 'Version' fields are labels '(max 10 chars)' and '(example: 1.0)' respectively. Below the 'Project info' section is a 'Compatibility options' section with three checkboxes: 'Use new LD editor' (checked), 'Use customizable workspace' (checked), and 'Use object oriented features' (unchecked). At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

You can set general project information:

- Project name
- Project version
- Project author name
- Project note

In addition, you can set compatibility options:

- **Use new LD editor:** the new Ladder Diagram editor is easier to use, by helping you in common operations working on the diagram is faster and more efficient. By default, this option is active. For more information, refer to Ladder Diagram (LD) Editor ([see page 236](#)).
- **Use customizable workspace:** allows you to manage your project tree in order to reach a more efficient workspace. By default, this option is active. For more information, refer to Project Custom Workspace ([see page 219](#)).
- **Use object oriented features:** not supported.

Code Generation Tab

Overview

Code generation tab of the **Project options** window:

Project options

Download

Debug

Build events

Cross Reference

General

Code generation

Build output

Case sensitivity (IEC default=no)

☐

Check functions and function blocks external variables

☒

Print debug informations

☐

Allow only integer indexes for arrays

☐

Run-time check of array bounds

☐

Run-time check of pointers

☐

Run-time check of division by zero

☐

Run-time check of interfaces

☐

Run-time check of references

☐

Strict pointers check

☐

Strict enumerations check

☐

Enable WAITING statement (extension to standard)

☐

Enable SFC control flags (extension to standard)

☐

Data copy size warning threshold (bytes, 0=disable)

200

Disable warning emission

☐

Disabled warning codes:

+

-

OK

Cancel

Apply

Help

You can edit some properties about code generation:

- **Case sensitivity:** you can set the project as case-sensitive checking this option.
NOTE: By default, this option is not active.
- **Check function and function block external variables:** if this option is disabled, all functions and function blocks can access to global variables without declaring them as external variables.
NOTE: By default, this option is enabled respecting the IEC 61131-3 standard.
- **Print debug information:** displays on the output window the significant debug info.
- **Allow only integer indexes for arrays:** if this option is checked you cannot use BYTE, WORD or DWORD as array indexes.
- **Run-time check of array bounds:** if this option is checked some additional code is included to verify that array indexes are not out of bounds during run-time. This option is available depending on target device.
- **Run-time check of pointers:** if this option is checked the pointers are tested for their validity before their use, calling a user-defined function **checkptr** on the target. This option is available depending on target device.
- **Run-time check of division by zero:** if this option is checked some additional code is included to verify that divisions by zero are not performed on the arrays during run-time. This option is available depending on target device.
- **Run-time check of interfaces:** not supported.
- **Run-time check of references:** not supported.
- **Strict pointers check:** if this option is checked, it is not possible to mix different pointer types and integer values.
- **Strict enumerations check:** if this option is checked, it is not possible to mix enumerative variables and integer types.
- **Enable WAITING statement (extension to standard):** if this option is checked the WAITING construct for the ST language is added as IEC 61131-3 extension. For more information, refer to Waiting Statement ([see page 444](#)).
- **Enable SFC control flags (extension to standard):** if this option is checked, HOLD and RESET flags for SFC POU are enabled.
- **Data copy size warning threshold (bytes, 0=disable):** when arrays or structures are copied, if their dimensions exceed the specified threshold, a message is emitted in order to inform the possible loss of performance of the PLC. If the threshold is set to 0, no messages are emitted.
- **Disable warning emission:** if this option is checked message emissions are not displayed on the output window.
- **Disable warning codes:** if this option is checked some specified message emissions are not displayed on the output window.

Build Output Tab

Overview

Build output tab of the **Project options** window:

The screenshot shows the 'Project options' dialog box with the 'Build output' tab selected. The dialog has a green title bar and a close button. It contains several tabs: 'Download', 'Debug', 'Build events', 'Cross Reference', 'General', 'Code generation', and 'Build output'. The 'Build output' tab is active, showing options for generating listing and EXP files, and downloadable target files. The 'Generate listing file' and 'Generate EXP file' options are checked. The 'Create downloadable target files' option is unchecked. Below this, there are three text input fields for 'PLC application', 'Source code', and 'Debug', each with a default filename: 'MyProject.bin', 'MyProject_source.bin', and 'MyProject_debug.bin' respectively. At the bottom, there are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

Section	Option	Value / Status
Listing	Generate listing file	☒
	Downloadable target files	
Downloadable target files	Create downloadable target files	☐
	PLC application:	MyProject.bin
	Source code:	MyProject_source.bin
	Debug	MyProject_debug.bin
EXP	Generate EXP file	☒

Here you can edit some significant properties of the output files generated by compiling operation.

Listing section:

- **Generate listing file:** if this option is checked the compiler generates a listing file named as **projectname.lst**.

Downloadable target files section:

- **Create downloadable target files:** if this option is checked the compiler generates the binary files that can be downloaded to the target. You can specify custom filenames or use the default names.

Only valid Windows filename are accepted.

- **PLC application** (active only if **Create downloadable target files** is checked): this field specifies the name of the PLC application binary file. The default name is **projectname.bin**.
- **Source code** (active only if **Create downloadable target files** is checked): this field specifies the name of the Source code binary file. The default name is **projectname._source.bin**.
- **Debug** (active only if **Create downloadable target files** is checked): this field specifies the name of the Debug symbol binary file. The default name is **projectname._debug.bin**

Generate EXP file section:

- **Generate EXP file**: if this option is checked the compiler generates an EXP file named as **projectname.exp**

Debug Tab

Overview

Debug tab of the **Project options** window:

The screenshot shows the 'Project options' dialog box with the 'Debug' tab selected. The dialog has a green title bar with a close button. Below the title bar are four tabs: 'General', 'Code generation', 'Build output', and 'Cross Reference'. The 'Debug' tab is active, showing several configuration options with input fields and checkboxes. At the bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

Option	Value
Polling period for debug functions (ms)	20
Number of displayed array elements without alert message	20
Polling period between more variables (ms)	0
Autosave watch list	☐
Enable memory dump (%MW<address>syntax)	☐
Watch internal variables of function blocks	☐

Here you can edit some significant properties of the debug behavior:

- **Polling period for debug function (ms):** set the active sampling period of the debug status.
- **Number of displayed array elements without alert message:** specifies the maximum number of array elements to be added in the watch window without being alerted.
- **Polling period between more variables (ms):** set the delay between sampling of variables.
- **Autosave watch list:** if checked the watch list status is saved into a file when the project is closed.
- **Enable memory dump:** enables the memory dump function for advanced debugging.
- **Watch internal variables of function blocks:** allows you to view internal variables of "VAR" class.

Build Events Tab

Overview

Build events tab of the **Project options** window:

The screenshot shows the 'Project options' dialog box with the 'Build events' tab selected. The dialog has a green title bar and a close button. Below the title bar are four tabs: 'General', 'Code generation', 'Build output', and 'Cross Reference'. The 'Build events' tab is active, showing a list of environment variables at the top: PRJTITLE, PRJPATH, PRJBASENAME, IMGNAME, APPLPATH, TARGETDEFNAME, FIRMWAREFILENAME, PRJRELEASE, PRJVERSION, PRJAUTHOR, PRJCONN, and SIMUL. Below this list are three text input fields for 'Post-build commands:', 'Pre-download commands:', and 'Post-download commands:'. At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

Here you can specify commands that run before the build starts or after the build finishes. You can also use a set of defined environment variables listed on the top of the window.

The environment variables are:

- **PRJTITLE**: project name.
- **PRJPATH**: project folder.
- **PRJBASENAME**: project full path without extension.
- **PRJFULLNAME**: project full path.
- **IMGNAME**: .imgx image file name.
- **TARGETDEFNAME**: project target name.
- **PRJRELEASE**: project name as defined in **General** tab of **Project options**.
- **PRJVERSION**: project version as defined in **General** tab of **Project options**.
- **PRJAUTHOR**: project author as defined in **General** tab of **Project options**.
- **PRJCONN**: current communication settings.
- **APPLPATH**: full application path.
- **SIMUL**: if simulation mode 1, else 0.

Cross Reference Tab

Overview

Cross Reference tab of the **Project options** window:

The screenshot shows the 'Project options' dialog box with the 'Cross Reference' tab selected. The 'Generate cross-reference' checkbox is checked. Below it, the 'Options' section contains several checkboxes for tracing different elements. The 'Trace Global Vars', 'Trace Programs', 'Trace Function Blocks', 'Trace Enums', 'Trace TypeDefs', 'Trace in Programs', and 'Trace in Functions Blocks' are all checked. 'Trace Local Vars', 'Trace Functions', 'Trace Tasks', 'Trace Structs', 'Trace Subrs', and 'Trace in Functions' are unchecked.

Project options					
General		Code generation		Build output	
Download		Debug		Build events	
				Cross Reference	
Generate cross-reference				☒	
Options					
Trace Global Vars	☒	Trace Local Vars	☐		
Trace Programs	☒	Trace Functions	☒		
Trace Function Blocks	☒	Trace Tasks	☐		
Trace Macros	☐	Trace Structs	☐		
Trace Enums	☐	Trace Subrs	☐		
Trace TypeDefs	☐				
Trace in Programs	☒	Trace in Functions	☐		
Trace in Functions Blocks	☒				

OK Cancel Apply Help

Here you can activate the Generate cross-reference function and set the related options.

The cross-reference trace options can be set for:

- **Global Vars:** Global variables
- **Local Vars:** Local variables
- **Programs:** Programs
- **Functions:** Functions
- **Function Blocks:** Function blocks
- **Tasks:** Tasks
- **Macros:** Macros
- **Structs:** Structures

-
- **Enums:** Enumerations
 - **Subrs:** Subranges
 - **TypeDefs:** Typedefs, used to create an alias name for another data type.

The cross-reference trace options can be set in:

- **Programs**
- **Functions**
- **Function Blocks**

Section 8.2

Working with Libraries

General Information

Libraries are a powerful tool for sharing objects between the projects. Libraries are stored in dedicated source file, whose extension is **.pll**.

What Is in This Section?

This section contains the following topics:

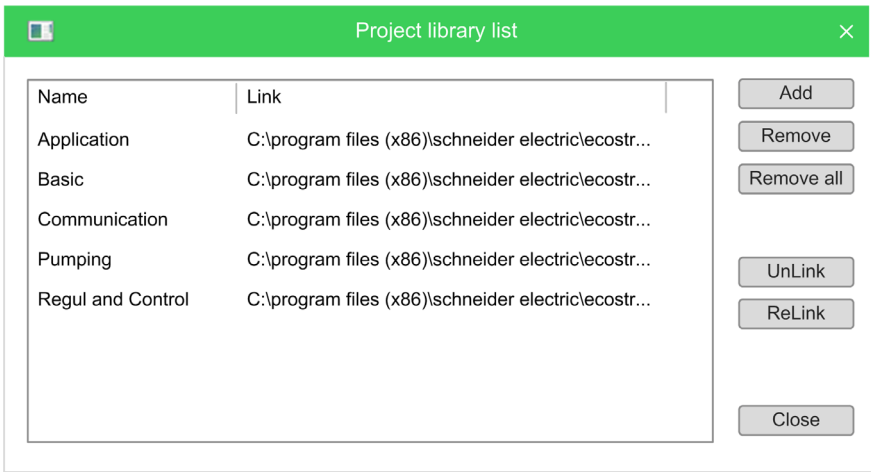
Topic	Page
Library Manager	175
Exporting to a Library	177
Importing from a Library or Another Source	178
Updating Existing Libraries	180

Library Manager

Overview

The library manager lists the libraries currently included in the project. It also allows you to include or remove libraries.

To access the library manager, click **Project → Library manager**.



Including a Library

The following procedure presents how to include a library in a project, which results in the library's objects becoming available to the current project.

Including a library means that a reference to the library's **.pll** file is added to the current project, and that a local copy of the library is made. You cannot edit the elements of an included library, unlike imported objects.

To copy or move a project which includes one or more libraries, make sure that references to those libraries are still valid in the new location:

Step	Action
1	Click Project → Library manager , which opens the Library manager dialog box.
2	Click the Add button, which causes an explorer dialog box to appear, to let you select the .pll file of the library you want to open.
3	When you have found the .pll file, open it either by double-clicking it or by clicking the Open button. The name of the library and its absolute pathname are now displayed in a new row at the bottom of the list.
4	Repeat step 1, 2, and 3 for the libraries you wish to include.
5	When you have finished including libraries, click the Close button.

Removing a Library

Removing a library does not delete the library itself, but removes the reference to it in the project.

The following procedure presents how to remove an included library from the current project:

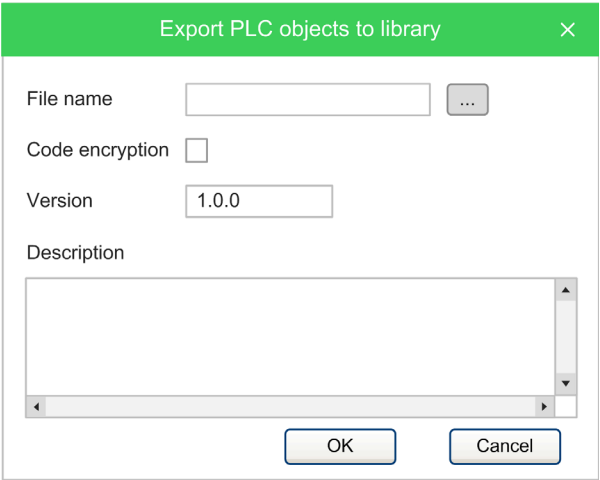
Step	Action
1	Click Project → Library manager menu of the PROGRAMMING main window, which opens the Library manager dialog box.
2	Select the library you wish to remove by clicking its name once. The Remove button is now enabled.
3	Click the Remove button, which causes the reference to the selected library to be deleted from the Project library list.
4	Repeat for the libraries you wish to remove. Alternatively, if you want to remove the libraries, you can click the Remove all button.
5	When you have finished removing libraries, click the Close button.

Exporting to a Library

Overview

You may export an object from the currently open project to a library in order to make that object available to other projects.

The following procedure presents how to export objects to a library:

Step	Action
1	Select the object you want to export in the tree structure of the project tab.
2	Click Project → Export object to library. You can also right-click the object and select Export object to library command. A dialog box appears: 
3	Enter the destination library by specifying the location of its .pll file. You can do this by: • Typing the full pathname in the text box. NOTE: If you enter the name of a non-existing .pll file, EcoStruxure Machine Expert - HVAC creates a new library. • Clicking the  Browse button in order to open an explorer dialog box which allows to browse your disk and the network.
4	You may optionally choose to encrypt the source code of the POU you are exporting in order to protect your intellectual property. You can also modify the version number and add a description.
5	Click OK to confirm the operation, otherwise, click Cancel to quit.

Importing from a Library or Another Source

Overview

You can import an object from a library in order to use it in the current project. When you import an object from a library, the local copy of the object loses its reference to the original library and it belongs exclusively to the current project. You can edit imported objects, unlike objects of included libraries.

Import Example

The following procedure presents how to import objects from a library:

Step	Action																																
1	Click Project → Import objects . This causes an explorer dialog box to appear, which lets you select the .pll file of the library you want to open.																																
2	When you have found the .pll or .plclib file, open it either by double-clicking it or by clicking the Open button. A dialog box of the library explorer appears: Object browser Objects filter ☒ Programs ☒ Function Blocks ☒ Functions ☒ Variables ☒ User types ☐ Operators ☐ Standard functions ☐ Local variables ☐ Basic types Check all Check none Other filters Name 1 OK Location All Library Vars type All Vars group <table><thead><tr><th>Name</th><th>Type</th></tr></thead><tbody><tr><td>AHUPlanModeStrategy</td><td>Function blocks</td></tr><tr><td>AHUTempCntrlStrategy</td><td>Function blocks</td></tr><tr><td>CompAlarmMgmt</td><td>Function blocks</td></tr><tr><td>CompAppLimit</td><td>Function blocks</td></tr><tr><td>CompCntrl_OnOff</td><td>Function blocks</td></tr><tr><td>CompCntrl_Slider</td><td>Function blocks</td></tr><tr><td>CompCntrl_VS</td><td>Function blocks</td></tr><tr><td>CompMgmt</td><td>Function blocks</td></tr><tr><td>CompMgmtVS</td><td>Function blocks</td></tr><tr><td>COPCalculation</td><td>Function blocks</td></tr><tr><td>Counter2Energy</td><td>Function blocks</td></tr><tr><td>DoubleInterpo_5x8</td><td>Function blocks</td></tr><tr><td>EcFanMgmt</td><td>Function blocks</td></tr><tr><td>EnergyTrend</td><td>Function blocks</td></tr><tr><td>FanMgmt</td><td>Function blocks</td></tr></tbody></table> Close Import objects ☐ Enable merge method Select all Select none	Name	Type	AHUPlanModeStrategy	Function blocks	AHUTempCntrlStrategy	Function blocks	CompAlarmMgmt	Function blocks	CompAppLimit	Function blocks	CompCntrl_OnOff	Function blocks	CompCntrl_Slider	Function blocks	CompCntrl_VS	Function blocks	CompMgmt	Function blocks	CompMgmtVS	Function blocks	COPCalculation	Function blocks	Counter2Energy	Function blocks	DoubleInterpo_5x8	Function blocks	EcFanMgmt	Function blocks	EnergyTrend	Function blocks	FanMgmt	Function blocks
Name	Type																																
AHUPlanModeStrategy	Function blocks																																
AHUTempCntrlStrategy	Function blocks																																
CompAlarmMgmt	Function blocks																																
CompAppLimit	Function blocks																																
CompCntrl_OnOff	Function blocks																																
CompCntrl_Slider	Function blocks																																
CompCntrl_VS	Function blocks																																
CompMgmt	Function blocks																																
CompMgmtVS	Function blocks																																
COPCalculation	Function blocks																																
Counter2Energy	Function blocks																																
DoubleInterpo_5x8	Function blocks																																
EcFanMgmt	Function blocks																																
EnergyTrend	Function blocks																																
FanMgmt	Function blocks																																

Step	Action
3	Select the objects you want to import. You can also make simple queries on the objects by using Filters . However, only the Name filter currently applies to libraries. To use it, enter the name of the desired objects, even using the * wildcard, if necessary.
4	Select the objects you want to import then click the Import objects button.
5	When you have finished importing objects, click indifferently OK or Cancel to close the browser.

Undoing Import from a Library

When you import an object in a project, you currently make a local copy of that object. You need to delete the local object in order to undo import.

Merge Function

When you import objects in a project or insert a copied mapped variable, you may encounter an overlapping address or duplicate naming error.

You can choose the behavior that EcoStruxure Machine Expert - HVAC should keep when encountering those problems by setting the corresponding environment options (*see page 49*).

The possible actions are:

Behavior		Ask	Automatic	Take from library	Do nothing
Naming behavior	If different types	✓	✓		✓
	If same type but not variables	✓	✓	✓	
	If both variables	✓	✓	✓	
Address behavior	If address overlaps	✓	✓	✓	
	Copy/paste mapped variable		✓		✓
Ask (default): User has to decide every time an action is required Automatic: A valid name or address is automatically generated by EcoStruxure Machine Expert - HVAC and assigned to the imported object. Take from library: The name or the address is taken from the imported object. Do nothing: The name or the address of the objects in the project are not modified.					

After importing objects, EcoStruxure Machine Expert - HVAC generates a log file in the project folder with detailed info.

Updating Existing Libraries

Overview

If you edit a linked library file, you can refresh its content on the project without closing EcoStruxure Machine Expert - HVAC:

Step	Action
1	Click Project → Refresh all libraries .
2	If the file is correct, EcoStruxure Machine Expert - HVAC updates the linked library content and displays a successful message in the output window, otherwise no modifications are made on the existing linked library.

Chapter 9

Managing Project Elements

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
9.1	Overview	182
9.2	Program Organization Units	183
9.3	Variables	188
9.4	Tasks	199
9.5	Derived Data Types	202
9.6	Browse the Project	211
9.7	Project Custom Workspace	219

Section 9.1

Overview

Project Window

Overview

This chapter explains how to manage with the elements which compose a project, namely: Program Organization Units (POUs), tasks, derived data types, and variables.

The **Project** window allows you to:

- Manage the application code.
- Manage and define complex variables that you define.
- Manage the tasks.

Content of the Project Window

The default **Project** window consists of the following items:

Item	Icon	Description
Project		Name of the project.
main <i>(see page 183)</i>		Initial program.
Var1 <i>(see page 195)</i>		Local variable.
Ungrouped_vars <i>(see page 188)</i>		Group of global variables.
Aux Variables		The shared global resources appear in the Project window but are defined in the Resources window. EEPROM Parameters , Status variables , and I/O Mapping appear here (as they are auto-generated).
Tasks <i>(see page 199)</i>		Creates tasks.

NOTE: The **Project** window content depends on the selected device.

Section 9.2

Program Organization Units

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	183
Creating a Program	184
Creating a Function Block/Function	185
Editing POU's	186
Source Code Encryption/Decryption	186

Overview

Description

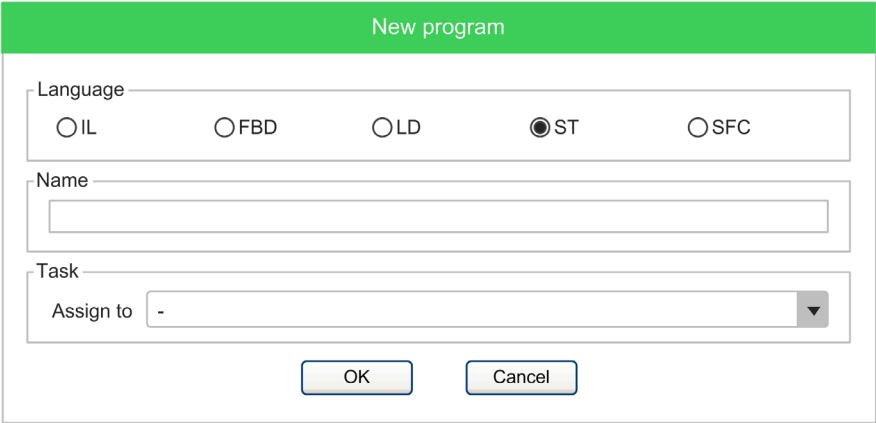
A POU is a Program Organization Unit of type Program, Function, or Function block.

This section presents how to add, edit, and remove POU's in the project.

Creating a Program

Description

To create a program:

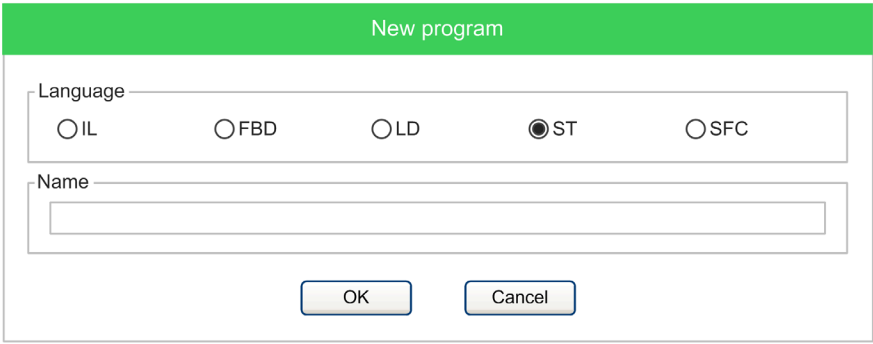
Step	Action
1	Select the target in the Project Window tree view. Click Project → New Object → New program .
2	A dialog box is displayed:  Select the specific language.
3	Enter its name.
4	Optionally, select a task in the list to associate the program to the selected task. NOTE: If you do not select a task at this step, an alert icon  appears below the program icon to indicate that the program is not yet associated to a task. Refer to Associating a Program to a Task (see page 199) to assign the program to the desired task.
5	Click the OK button to confirm.

Alternatively, you can create a new POU from the context menu by selecting a folder or the root element of the project. For more information, refer to Custom Workspace Operations ([see page 222](#)).

Creating a Function Block/Function

Description

To create a Function Block/Function:

Step	Action
1	Select the target in the Project Window tree view. For Function Block, click Project → New Object → New function . For Function, click Project → New Object → New function block .
2	A dialog box is displayed:  Select the specific language. NOTE: Creating functions is available in four programming languages. SFC language is not supported for functions.
3	Enter its name.
4	Click the OK button to confirm.

A function or function block is a (sub)program with inputs and outputs:

- A **function** requires **n** inputs and a single output (**RESULT**) with the same name as the function.
The local memory of the function is initialized each time the function is called.
The function is used within the program by passing the input variables.
- A **function block** requires **n** inputs and **m** outputs. The local memory of each instance of the function block is kept between one call and the next (static memory).
The function block is used within the program as an instance in the same way as the declaration of a variable.

Each function or function block can be used within a program by dragging and dropping the icon into the editor window of the program.

Editing POU's

Overview

To edit a POU, open it by double-clicking it from the project tree. The relative editor opens and lets you modify the source code of the POU.

Changing the Name of the POU

Select a POU from the project tree, then right-click and select **Rename program**, **Rename function block**, or **Rename function** depending on the POU.

Duplicating a POU

Select a POU from the project tree, then click **Project → Duplicate object**.
Enter the name of the new duplicated POU and confirm the operation.

Deleting POU's

Select a POU from the project tree, then click **Project → Delete Object**.
Confirm the operation to delete the POU.

Source Code Encryption/Decryption

Overview

PROGRAMMING can encrypt POU's and require a password to decrypt them, hiding the source code of the POU.

Encrypting a POU

Select a POU from the project tree, right-click, then select **Crypt** in the context menu
Double enter the password and confirm the operation.

PROGRAMMING displays in the project tree a special marker icon that overlays the standard POU icon in order to inform you that the POU is encrypted.

Decrypting a POU

Select a POU from the project tree, right-click, then select **Decrypt** in the context menu

Encrypting all POU's

Select the target from the project tree, right-click, then select **Crypt all objects** in the context menu.
All POU's are encrypted with the same password.

Decrypt all POU

Select the target from the project tree, right-click, then select **Decrypt all objects** in the context menu.

Section 9.3

Variables

Overview

There are two classes of variables in **PROGRAMMING**: global variables and local variables. This section presents how to add to the project, edit, and remove both global and local variables.

What Is in This Section?

This section contains the following topics:

Topic	Page
Global Variables	188
Local Variables	195
Creating Multiple Variables	198

Global Variables

Description

Global variables can be seen and referenced by any POU of the project.

Classes of Global Variables

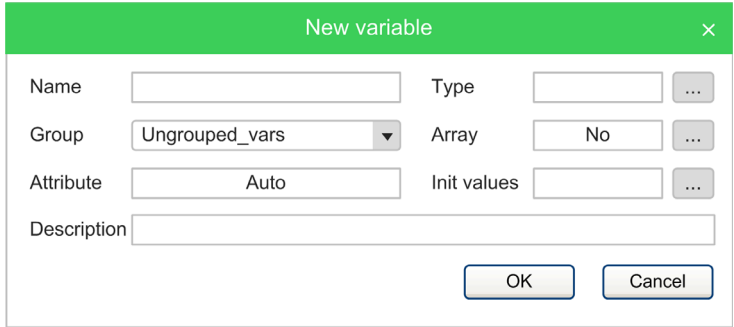
Global variables are organized in special folders of the project tree called **Global variables group**.

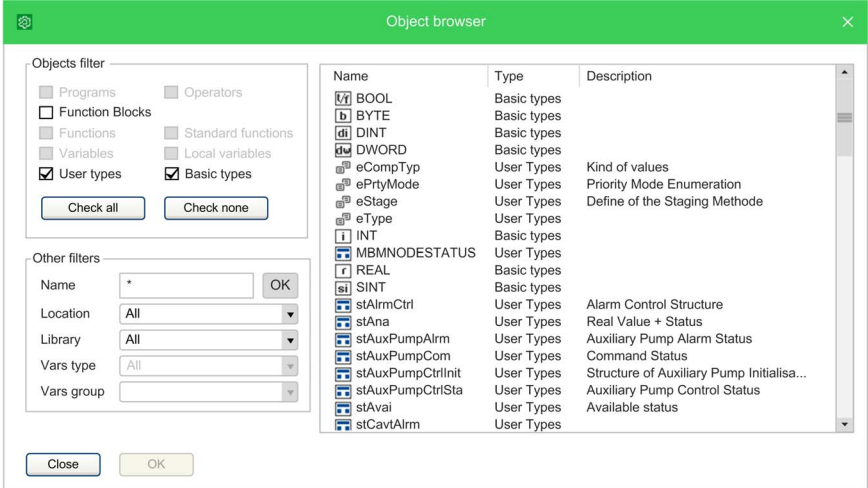
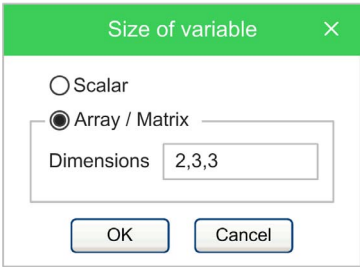
Those variables are classified according to their properties as:

- Automatics: the compiler automatically allocates them to an appropriate location in the target device memory.
- Mapped: they have an assigned address in the target device logical addressing system, which you specify.
- Constants: are declared having the `CONSTANT` attribute; their values cannot be altered by the programming logic.
- Retains: they are declared having the `RETAIN` attribute; their values are stored in a persistent, non-volatile memory area of the target device.

Creating a New Global Variable

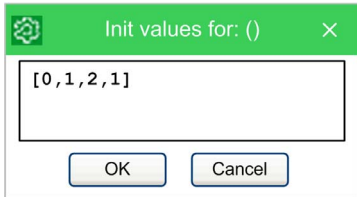
Creating a New Global Variable

Step	Action
1	In order to create a new global variable, you need to define at least one Global variables group in your project. Afterward select it from the project tree then choose the appropriate item of the menu Project → New Object → New variable. Refer to Custom Workspace Operations (<i>see page 222</i>). PROGRAMMING presents a dialog box: 
2	Enter the name of the variable. The variable name must be a valid IEC 61131-3 identifier. Valid variable names can consist of any combination of letters, numbers, and underscores, though they cannot begin with a number.

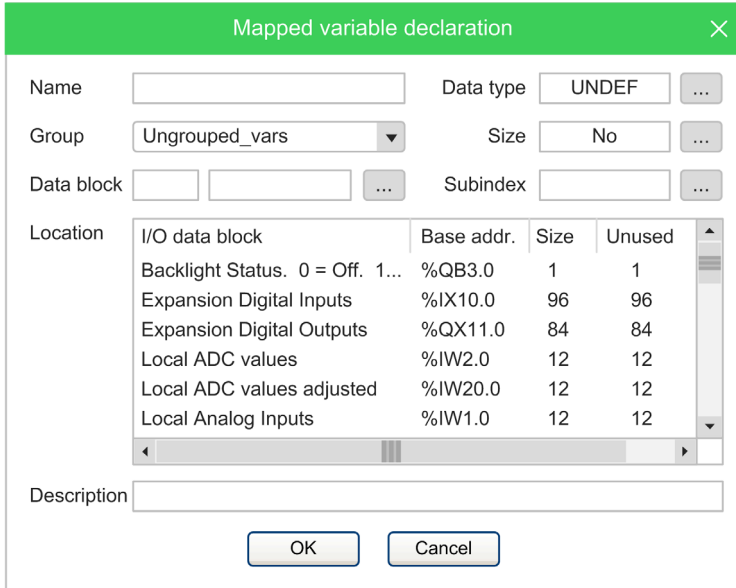
Step	Action
3	Specify the type of the variable either by typing it or by selecting it from the list that PROGRAMMING displays when you click the  Browse button: 
4	If you want to declare an array, you must specify its size by clicking the  Browse button next to the Array field: 

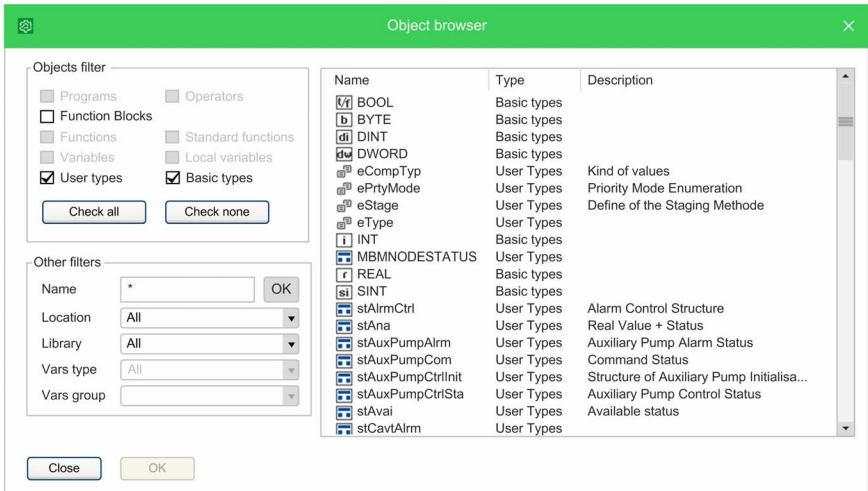
Enter the length of the array. Use comma to separate the length of each dimension (up to 3). For example: 2 or 2,3 or 2,3,3.

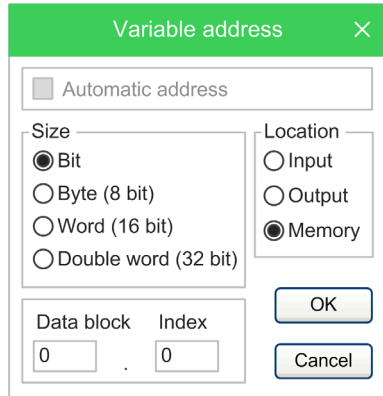
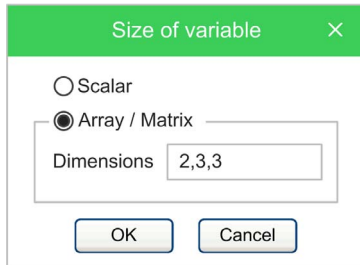
NOTE: A dimension should be greater than 1. For example, entering 2,1 or 1,2 is equivalent to entering 2.

Step	Action
5	You may optionally assign the initial value to the variable or to the single elements of the array by clicking the  Browse button next to the Init values field:  NOTE: Initial values must be separated by a comma.
6	Click the OK button to validate.

Creating a New Global Mapped Variable:

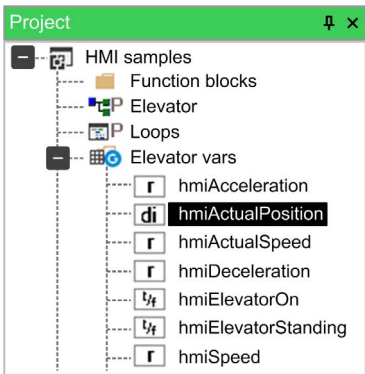
Step	Action
1	To create a newly global mapped variable, you need to define at least one Global variables group in your project. Afterward select it from the project tree then choose the menu Project → New Object → New variable → Mapped Variable. PROGRAMMING presents a dialog box: 

Step	Action
2	Enter the name of the variable. The variable name must be a valid IEC 61131-3 identifier. Valid variable names can consist of any combination of letters, numbers, and underscores, though they cannot begin with a number.
3	Specify the type of the variable either by typing it or by selecting it from the list that  PROGRAMMING displays when you click the Browse button: 
4	You can select the group in the Group list.

Step	Action
5	You are required to specify the address of the variable by doing one of the following operations: - Click the Data block  browser button to open the editor of the address; then enter the desired value:  The 'Variable address' dialog box has a green header with 'Variable address' and a close button. It contains a checkbox for 'Automatic address'. Below it are two sections: 'Size' with radio buttons for 'Bit' (selected), 'Byte (8 bit)', 'Word (16 bit)', and 'Double word (32 bit)'; and 'Location' with radio buttons for 'Input', 'Output', and 'Memory' (selected). At the bottom, there are input fields for 'Data block' (containing '0') and 'Index' (containing '0'), and 'OK' and 'Cancel' buttons. Click OK. - Select, from the Location list, the memory area you want to use: the tool automatically calculates the address of the first free memory location of that area.
6	Depending on the selected location, you can specify its size by clicking the  Browse button next to the Size field:  The 'Size of variable' dialog box has a green header with 'Size of variable' and a close button. It contains two radio buttons: 'Scalar' and 'Array / Matrix' (selected). Below the 'Array / Matrix' radio button is a 'Dimensions' input field containing '2,3,3'. At the bottom are 'OK' and 'Cancel' buttons. Enter the length of the array. Use comma to separate the length of each dimension (up to 3). For example: 2 or 2,3 or 2,3,3. NOTE: A dimension should be greater than 1. For example, entering 2,1 or 1,2 is equivalent to entering 2.
7	You can enter the subindex value.
8	Click the OK button to validate.

Editing a Global Variable

To edit the definition of an existing global variable, open it by double-clicking it, or the folder that it belongs to, from the project tree. The global variables editor opens and lets you modify its definition.



	Name	Type	Address
1	hmiTargetPosition	DINT	%MB1.58
2	hmiActualPosition	DINT	%MD1.62
3	hmiHighSpeed	REAL	%MB1.66
4	hmiAcceleration	REAL	%MD1.70
5	hmiDeceleration	REAL	%MD1.74

Changing the name of the variable:

Select the variable you want to rename from the project tree then right-click its name and select **Rename variable** command. You can also double-click the variable name and modify its name in the editor window. The new name must be updated in all locations where the renamed variable is used.

Duplicating a variable:

Select the variable you want to duplicate from the project tree then right-click its name and select **Duplicate variable** command.

Enter the name of the new duplicated variable and confirm.

Deleting a Global Variable

Select the variable you want to delete from the project tree then right-click its name and select **Delete variable** command.

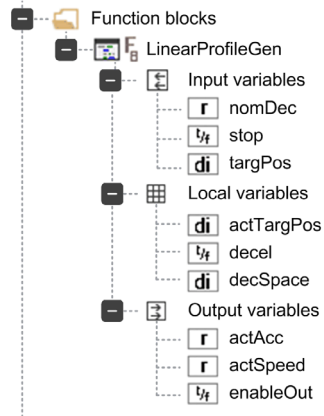
Confirm the operation to delete the variable.

Local Variables

Description

Local variables are declared within a POU (either program, function, or function block), the POU itself being the only project element which can refer to and access them.

Local variables are listed in the project tree under the POU which declares them (only when that POU is open for editing). The local variables are further classified according to their class (for example, as input or output variables):



In order to create, edit, and delete local variables, you have to open the POU for editing and use the local variables editor. The project needs to be saved in order to update the POU branch structure of the project tree, including the modifications applied to the local variables.

For more information, refer to [Opening a Local Variables Editor](#) (see page 263).

Creating Local Variables

Click **Variables** → **Insert** command or click the **Insert record**  icon in the **Project** toolbar. You can also create multiple variables (*see page 198*).

The variable appears in yellow in the **Local variables** window, and you can define its characteristics by clicking the respective boxes.

Created variables can be displayed in table format by selecting the icon on the upper right corner of the window:

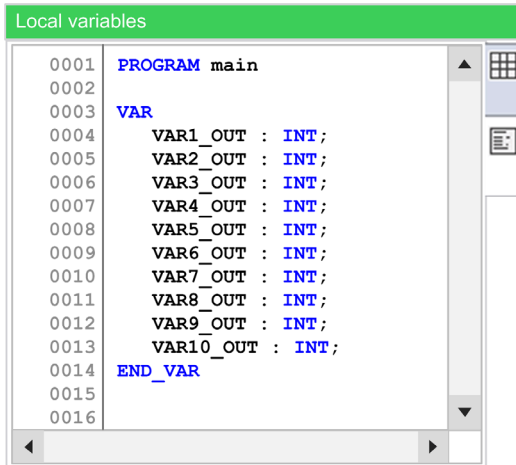
Local variables						
	Name	Type	Address	Array	Init value	Attribute
1	VAR1_OUT	INT	Auto	No		...
2	VAR2_OUT	INT	Auto	No		...
3	VAR3_OUT	INT	Auto	No		...
4	VAR4_OUT	INT	Auto	No		...
5	VAR5_OUT	INT	Auto	No		...
6	VAR6_OUT	INT	Auto	No		...
7	VAR7_OUT	INT	Auto	No		...
8	VAR8_OUT	INT	Auto	No		...
9	VAR9_OUT	INT	Auto	No		...
10	VAR10_OUT	INT	Auto	No		...

The characteristics of the local variables of a program are:

- **Name:** to choose the name of the variable.
- **Type:** to choose from one of the preset options or variables defined by you.
- **Address:** the default setting is **Auto**.
- **Array:** defines whether the variable is array type (if so, define its dimension) or not.
- **Init value:** initial value. It is the value of the variable after each power cycle.
- **Attribute:** to set the variable as **CONSTANT** (variable cannot be overwritten) or **RETAIN**.
- **Description:** to write a description for the variable.

NOTE: Local variable table has different formats for program and function blocks.

Created variables can be displayed also in code format by selecting the second icon on the upper right corner of the window:



The screenshot shows a window titled "Local variables" with a green header bar. Inside the window, a list of variables is displayed in a code-like format. The variables are listed on the left side of the window, and their data types are listed on the right side. The variables are: VAR1_OUT, VAR2_OUT, VAR3_OUT, VAR4_OUT, VAR5_OUT, VAR6_OUT, VAR7_OUT, VAR8_OUT, VAR9_OUT, and VAR10_OUT. All variables are of type INT. The window has a scrollbar on the right side and a small icon in the top right corner.

```
0001 PROGRAM main
0002
0003 VAR
0004     VAR1_OUT : INT;
0005     VAR2_OUT : INT;
0006     VAR3_OUT : INT;
0007     VAR4_OUT : INT;
0008     VAR5_OUT : INT;
0009     VAR6_OUT : INT;
0010     VAR7_OUT : INT;
0011     VAR8_OUT : INT;
0012     VAR9_OUT : INT;
0013     VAR10_OUT : INT;
0014 END_VAR
0015
0016
```

The local variables appear in the **Project** window, below the program folder, identified by an icon.

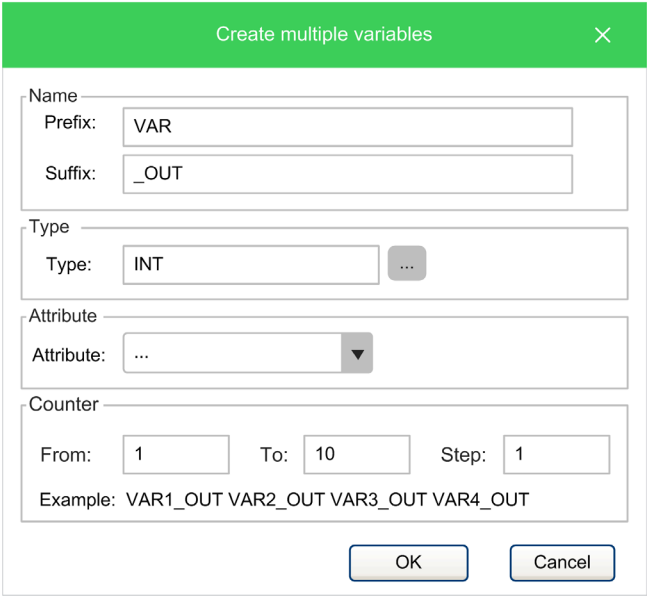
NOTE: The local variables are reinitialized each time the POU is executed.

Creating Multiple Variables

Description

PROGRAMMING allows you to create multiple variables simultaneously.

Creating Multiple Variables:

Step	Action
1	Open the POU for editing.
2	Select Variables → Create multiple . A dialog box appears: 
3	Specify the prefix and the suffix of the new variables names.
4	Specify the type of the variables either by typing it or by selecting it from the list that is displayed  Browse button. when you click the
5	If needed, select the Attribute in the list.
6	Insert the number of the variables you want to create specifying the start index, the end index, and the step value. You can see an example of the generated variable names at the bottom of the dialog.

Section 9.4

Tasks

What Is in This Section?

This section contains the following topics:

Topic	Page
Associating a Program to a Task	199
Task Configuration	201

Associating a Program to a Task

Overview

For a program to run, it must be associated to a task.

The following types of task are available:

- **Boot** task is executed only once at PLC start up.
- **Init** task executed at each download of the application and on starting up the system (after Boot).
NOTE: The associated program initializes slaves and messages according to the configuration, with fixed values that are independent of the run time.

WARNING

AUTOMATIC RESTART OF CONTROLLER

- Do not download your application without first accessing the state of your machine or process.
- Do not download your application without first ascertaining that there is no risk of injury to anyone in or around your machine or process.

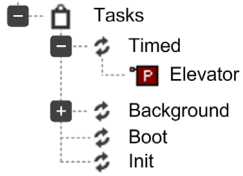
Failure to follow these instructions can result in death, serious injury, or equipment damage.

- **Timed** task runs at regular intervals which you can set. The default setting depends on target type.
NOTE: Modbus messages do not interfere with this task.
- **Background** task runs with low priority after the execution of the **Timed** tasks.
- **Modbus** task executed to implement Modbus Master, calling relative function blocks, and to send messages (Only for M171O).

Associating a Program to a Task Type

Each new project has the **main** program associated to the **Background** task. The **main** program can be removed and/or associated to other tasks.

To associate a program to a task:

Step	Action
1	Right-click on the task where you want to add the program from the project tree then choose the Add program command.
2	Select the program you want to be executed by the task from the list which shows up and confirm your choice.
3	The program has been assigned to the task: 

Managing a Program into a Task

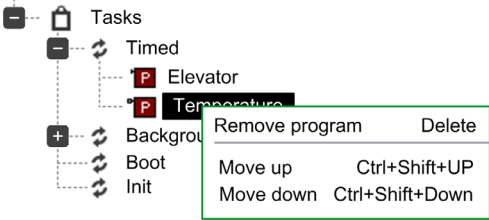
You can assign more than one program to a task.

Programs are executed sequentially, as they are assigned and visible in the tree.

When you right-click a program associated to a task, three actions are available:

- **Remove program (Delete)**
- **Move up (Ctrl+Shift+Up)**
- **Move down (Ctrl+Shift+Down)**

Move up and **Move down** allow you to change the execution order of the program within the same task:



Section 9.5

Derived Data Types

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	202
Typedefs	203
Structures	205
Enumerations	207
Subranges	209

Overview

Description

The **Definitions** section of the **Workspace** window lets you define derived data types.

The derived data type is a complex classification that identifies one or various data types and is composed of primitive data types.

You have the flexibility to create those specific types that have advanced properties and uses in addition to the primitive data types.

PROGRAMMING can manage:

- Typedefs (*see page 203*)
- Structures (*see page 205*)
- Enumeration (*see page 207*)
- Subranges (*see page 209*)

Typedefs

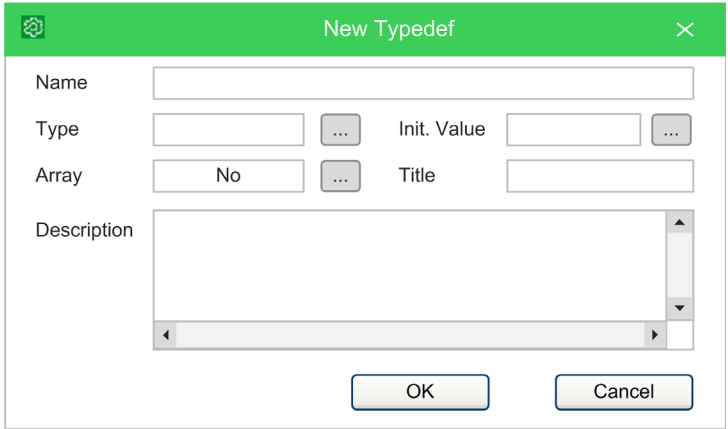
Description

The following paragraphs present how to manage **TypeDef**.

For more information about **TypeDefs**, refer to TypeDefs description ([see page 374](#)).

Creating a New Typedef

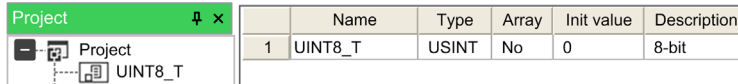
To create a new **TypeDef**:

Step	Action
1	New TypeDef can be added in the Project window tree: • Right-click on the name of the project. • Click Add → New Definition → TypeDef. A dialog box is displayed: 
2	Enter the name of the TypeDef. The TypeDef name must be a valid IEC 61131-3 identifier. Valid names can consist of any combination of letters, numbers, and underscores, though they cannot begin with a number.
3	Specify the type of the TypeDef either by typing it or by selecting it from the list that is displayed when you click the  Browse button.

Step	Action
4	If you want to declare an array, you must specify its size by clicking the  Browse button next to the Array field: Size of variable Scalar Array / Matrix Dimensions2,3,3 OK Cancel Enter the number of elements of the array. Use comma to separate the number of elements of each dimension (up to 3 dimensions). For example: 2 or 2,3 or 2,3,3. NOTE: A dimension must be greater than 1 to be relevant. For example, entering 2,1 or 1,2 is equivalent to entering 2.
5	You may optionally assign the initial value to the variable or to the single elements of the array by clicking the  Browse button next to the Init values field: Init values for: () [0,1,2,1] OK Cancel NOTE: Initial values must be separated by a comma.
6	You can specify: • An optional title. • An optional description.
7	Click Ok to validate.

Editing a Typedef

To edit a **TypeDef**, double-click it from the **Project** window tree. The **TypeDef** is displayed in the window where you can modify the values:



	Name	Type	Array	Init value	Description
1	UINT8_T	USINT	No	0	8-bit

To modify a value, select it in the table then

- Enter a new value, or
- Click the browser button. A window appears to enter a new value.

To edit the properties of an existing **TypeDef**, right-click it from the **Project** window tree and select **Edit properties** to open the associated editor.

To directly modify the name of the **TypeDef**, click it in the **Project** window tree then click it again to open the name field. Enter the new name and press **Enter** to validate.

To edit the properties of an existing **TypeDef**, right-click it from the **Project** window tree and select **Edit properties**.

To display the properties of an existing **TypeDef**, right-click it from the **Project** window tree and select **View properties** to open the associated **Properties Window**.

Deleting a Typedef

To delete an existing **TypeDef**, right-click it from the **Project** window tree and select **Delete**.

Structures

Description

The following paragraphs present how to manage structures.

For more information about **Structure**, refer to Structures description ([see page 375](#)).

Creating a New Structure

To create a new **Structure** in the **Project** window tree:

Step	Action
1	To create a new Structure, do one of the following operations: ● Right-click on the name of the project then click Add → New Definition → Structure. ● Select the project in the Project window tree then, in the menu, click Project → New object → New Definition → Structure A dialog box is displayed: New Structure Name COMPLEX Title Complex Number Version 1.0.0 Description Complex Number OK Cancel
2	Enter the name of the Structure .
3	You can specify: ● An optional title. ● An optional version number. ● An optional description.
4	Click Ok to validate.

Editing a Structure

To edit an existing **Structure**, double-click it from the **Project** window tree.

Project		Name	Pos.	Type	Array	Init value
Project	1	Re	0	REAL	No	0
	2	Im	1	REAL	No	0

Right-click to insert or delete elements.

To modify a value, select it in the table then:

- Enter a new value, or
- Click the browser button. A window appears to enter a new value.

To edit the properties of an existing **Structure**, right-click it from the **Project** window tree and select **Edit properties** to open the associated editor.

To directly modify the name of the **Structure**, click it in the **Project** window tree then click it again to open the name field. Enter the new name and press **Enter** to validate.

To display the properties of an existing **Structure**, right-click it from the **Project** window tree and select **View properties** to open the associated **Properties Window**.

Deleting a Structure

To delete an existing **Structure**, right-click it from the **Project** window tree and select **Delete**.

Enumerations

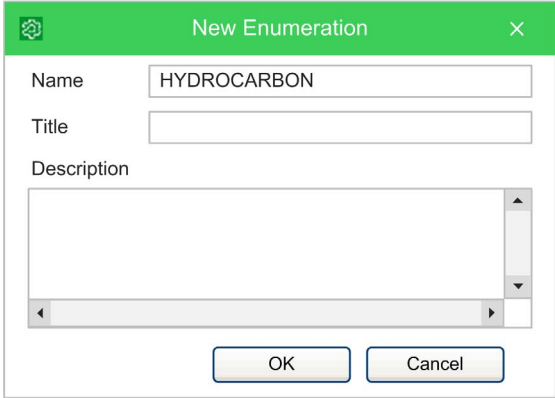
Description

The following paragraphs show you how to manage enumerations.

For more information about **Enumeration**, refer to Enumerated Data Types ([see page 374](#)).

Creating a New Enumeration

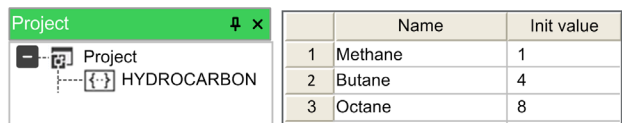
To create a new **Enumeration**:

Step	Action
1	To create a new Enumeration, do one of the following operations: - Right-click on the name of the project then click Add → New Definition → Enumeration. - Select the project in the Project window tree then, in the menu, click Project → New object → New Definition → Enumeration A dialog box is displayed: 
2	Enter the name of the Enumeration .

Step	Action
3	You can specify: • An optional title. • An optional description.
4	Click Ok to validate.

Editing an Enumeration

To edit an existing **Enumeration**, double-click it from the **Project** window tree.



Right -click to insert or delete element.

To modify a value, select it in the table then:

- Enter a new value, or
- Click the browser button. A window appears to enter a new value.

To edit the properties of an existing **Enumeration**, right-click it from the **Project** window tree and select **Edit properties** to open the associated editor.

To directly modify the name of the **Enumeration**, click it in the **Project** window tree then click it again to open the name field. Enter the new name and press **Enter** to validate.

To display the properties of an existing **Enumeration**, right-click it from the **Project** window tree and select **View properties** to open the associated **Properties Window**.

Deleting an Enumeration

To delete an existing **Enumeration**, right-click it from the **Project** window tree and select **Delete**.

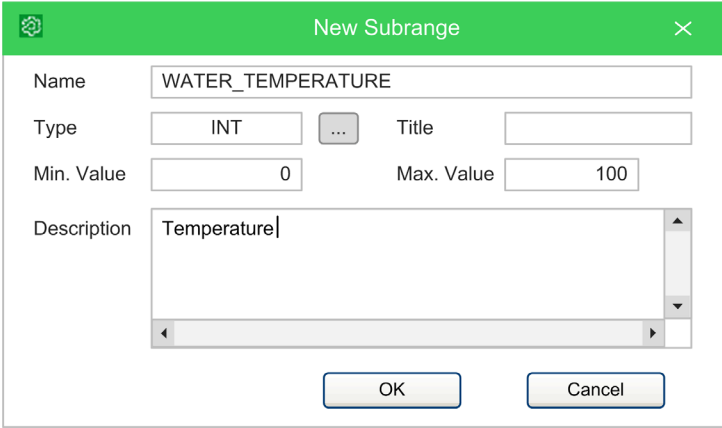
Subranges

Description

The following paragraphs present how to manage subranges.
For more information about **Subrange**, refer to Subranges description (*see page 375*).

Creating a New Subrange

To create a new **Subrange**:

Step	Action
1	To create a new Subrange, do one of the following operations: ● Right-click on the name of the project then click Add → New Definition → Subrange. ● Select the project in the Project window tree then, in the menu, click Project → New object → New Definition → Subrange A dialog box is displayed: 
2	Enter the name of the Subrange .
3	Specify the type of the Subrange either by typing it or by selecting it from the list that is displayed  when you click the Browse button.
4	You can specify: ● The minimum value. ● The maximum value.
5	You can specify: ● An optional title. ● An optional description.
6	Click Ok to validate.

Editing a Subrange

To edit an existing **Subrange**, double-click it from the **Project** window tree.

Project						
		Name	Type	Min	Max	Description
1	WATER_TEMPERATURE	INT	0	100	Temperature	

To modify a value, select it in the table then:

- Enter a new value, or
- Click the browser button. A window appears to enter a new value.

To edit the properties of an existing **Subrange**, right-click it from the **Project** window tree and select **Edit properties** to open the associated editor.

To directly modify the name of the **Subrange**, click it in the **Project** window tree then click it again to open the name field. Enter the new name and press **Enter** to validate.

To display the properties of an existing **Subrange**, right-click it from the **Project** window tree and select **View properties** to open the associated **Properties Window**.

Deleting a Subrange

To delete an existing **Subrange**, right-click it from the **Project** window tree and select **Delete**.

Section 9.6

Browse the Project

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	211
Object Browser	212
Search with the Find in Project Command	217

Overview

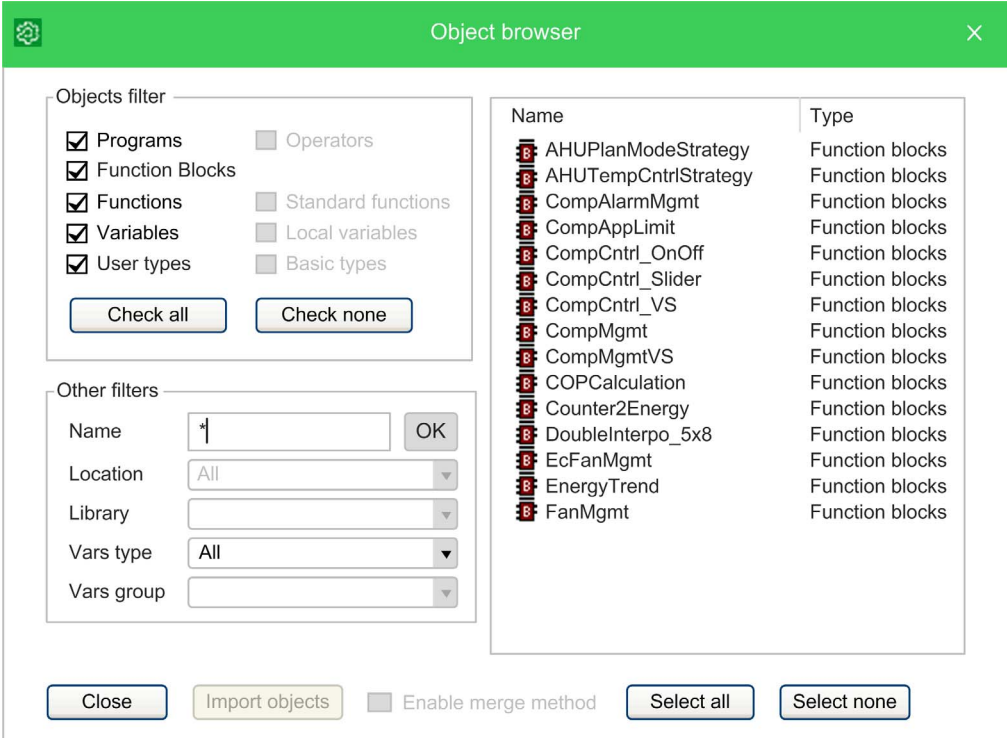
Description

PROGRAMMING provides two tools to search for an object within a project: the **Object browser** and the **Find in project** feature.

Object Browser

Description

PROGRAMMING provides a tool for browsing the objects of your project: the **Object Browser**.



This tool is context-dependent, this implies that the kind of selectable objects and the available operations on objects depend on the context.

Object browser can be opened in the following ways:

- **Browser mode:**
In **PROGRAMMING**, click the menu command **Project → Object Browser**.
- **Import object mode:**
Right-click the project name in the **Project** window and select **Import objects**, the **Object browser** opens after opening the selected object.
- **Select object mode:**
For example, to add a program to a task, right-click a task item in the **Project** window and select **Add program** command. It opens the **Object browser** window.

User interaction with **Object browser** is similar for the three modes and is presented in the next paragraph.

Common Features and Usage of Object Browser

This section presents the features and the usage of the **Object browser**.

Objects filter:

Objects filter

☒ Programs	☐ Operators
☒ Function Blocks	
☒ Functions	☐ Standard functions
☐ Variables	☐ Local variables
☐ User types	☐ Basic types

Check all Check none

This is the main filter of the **Object browser**. You can select one of the available (enabled) object items.

In this example, **Programs**, **Function Blocks**, **Functions** are selected, so objects of this type are displayed in the object list. **Variables** and **User types** objects can be selected, but objects of that type are not currently displayed in the object list.

You can also click the **Check all** button to select the available objects at one time or can click the **Check none** button to deselect the objects at one time.

Other filters:

Other filters

Name

Location

Library

Vars type

Vars group

Selected objects can be also filtered by name, symbol location, specific library, type of variable, and group of variables.

Filters are all additive and are immediately applied after setting.

Name	
Function	Filters objects on the base of their name.
Allowed values	All the strings of characters.
Use	Type a string to display the specific object whose name matches the string. Use the * wildcard if you want to display all the objects whose name contains the string in the Name text box. Type * if you want to disable this filter. Press Enter when edit box is focused or click the OK button to apply the filter.
Applies to	All object types.

Symbol location	
Function	Filters objects on the base of their location.
Allowed values	All, Project, Target, Library, Aux. Sources.
Use	All= Disables this filter. Project= Objects declared in the PROGRAMMING project. Target= Firmware objects. Library= Objects contained in a library. In this case, use simultaneously also the Library filter. Aux sources= Displays auxiliary sources only.
Applies to	All objects types.

Library	
Function	Filters objects contained in library. The value of this filter is relevant only if the Symbol location filter is set to Library .
Allowed values	All, libraryname1, libraryname2, ...
Use	All= Displays objects contained in any library. LibrarynameN= Displays only the objects contained in the library named librarynameN.
Applies to	All objects types.

Vars Type	
Function	Filters global variables and system variables (also known as firmware variables) according to their type.
Allowed values	All, Normal, Constant, Retain
Use	All= Displays all the global and system variables. Normal= Displays normal variables only. Constant= Displays constants only. Retain= Displays retain variables only.
Applies to	Variables.

Vars Group	
Function	Filters variables according to their group.
Allowed values	Analog_inputs, Analog_Outputs, ...
Use	Displays the variables that belongs to the selected group.
Applies to	Variables.

Object list:

Object list shows all the filtered objects. The list can be ordered in ascending or descending order by clicking the header of the column. It is possible to order items by **Name**, **Type**, or **Description**.

Double-clicking an item allows you to perform the default associated operation (the action is the same as the **OK**, **Import object**, or **Open source** button actions).

When item multiselection is allowed, **Select all** and **Select none** buttons are visible.

It is possible to select all objects by clicking the **Select all** button. **Select none** deselects all objects.

If at least one item is selected on the list operation, buttons are enabled.

Resize:

Object browser window can be resized, the cursor changes along the border of the window and allows you to resize it. When reopened, **Object browser** window keeps the same size and position of the previous usage.

Using Object Browser

In order to use the object browser to look over through the elements of the project, choose the menu item **Project → Object Browser**.

Available objects:

In this mode, you can list objects of these types:

- Programs.
- Function Blocks.
- Functions.
- Variables.
- User types.

These items can be checked or unchecked in the **Objects filter** section to show or to hide the objects of the chosen type in the list.

Other types of objects (Operators, Standard functions, Local variables, Basic types) cannot be browsed in this context, therefore they are unchecked and disabled.

Available operations:

Open source, default operation for double-clicking an item	
Function	Opens the editor by which the selected object was created and displays the relevant source code.
Use	If the object is a program, or a function, or a function block, this button opens the relevant source code editor. If the object is a variable, then this button opens the variable editor. Select the object whose editor you want to open, then click the Open source button.

Export to library	
Function	Exports an object to a library.
Use	Select the objects you want to export, then click the Export to library button.

Delete objects	
Function	Allows you to delete an object.
Use	Select the object you want to delete, then click the Delete object button.

Using Object Browser for Import

Object browser is also used to support object importations in the project from an external library. In order to use the object browser to import external library to the project, choose the menu item **Project → Import objects**.

Available objects:

In this mode you can list objects of these types:

- Programs.
- Function blocks.
- Functions.
- Variables.
- User types.

These items can be checked or unchecked in **Objects filter** section to show or to hide the objects of the chosen type in the list.

Other types of objects (Operators, Standard functions, Local variables, Basic types) cannot be imported so they are unchecked and disabled.

Available operations:

Import objects is the only operation supported in this mode. It is possible to import selected objects by clicking the **Import objects** button or by double-clicking one of the objects in the list.

Using Object Browser for Object Selection

Object browser dialog is useful for many operations that require the selection of a single PLC object. The Object browser can be used to select the program to add to a task, to select the type of a variable, to select an item, to find in the project, and so on.

Available objects:

Available objects are strictly dependent on the context. For example, in the program assignment to a task operation, the only available objects are program objects.

Not all available objects may be selected by default.

Available operations:

In this mode, it is possible to select a single object by double-clicking the list or by clicking the **OK** button; then the dialog is automatically closed.

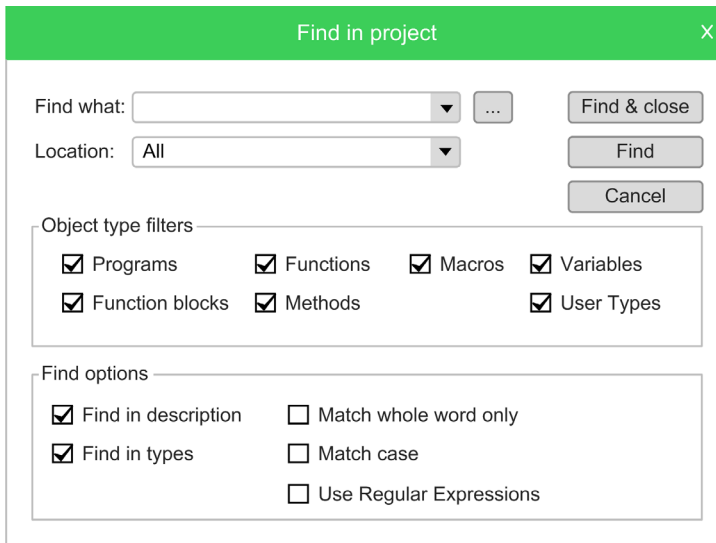
Search with the Find in Project Command

Description

The **Find in project** command retrieves all the instances of a specified character string in the project.

In order to use this functionality, choose the menu item **Edit → Find in project**.

PROGRAMMING displays the following dialog box:



Step	Action
1	In the Find what box, type the name of the object you want to search. Otherwise, click the  Browse button on the right of the box, and select the name of the object from the list of all the existing items.
2	Select one of the values listed in the Location box to specify a constraint on the location of the objects to be monitored.
3	The frame named Object type filters contains 7 check boxes, each of which, if ticked, enables research of the string among the object it refers to.
4	Tick the relevant options check boxes in the Find options frame.

Step	Action																		
5	Click Find to start the search; otherwise click Cancel to quit. The results are displayed in the Find in project tab of the Output window. Output Searching for 'cnt' <table><tr><td>Pulse(7) -</td><td>ST</td><td>cnt</td></tr><tr><td>Pulse(14) -</td><td>LD</td><td>cnt</td></tr><tr><td>Pulse(16) -</td><td>ST</td><td>cnt</td></tr><tr><td>Pulse(18) -</td><td>LD</td><td>cnt</td></tr><tr><td>Pulse(26) -</td><td>ST</td><td>cnt</td></tr><tr><td>Pulse(LV) - cnt : UINT</td><td></td><td></td></tr></table> 6 occurrence(s) have been found. Build Find in project Debug Resources HMI Output	Pulse(7) -	ST	cnt	Pulse(14) -	LD	cnt	Pulse(16) -	ST	cnt	Pulse(18) -	LD	cnt	Pulse(26) -	ST	cnt	Pulse(LV) - cnt : UINT		
Pulse(7) -	ST	cnt																	
Pulse(14) -	LD	cnt																	
Pulse(16) -	ST	cnt																	
Pulse(18) -	LD	cnt																	
Pulse(26) -	ST	cnt																	
Pulse(LV) - cnt : UINT																			

Section 9.7

Project Custom Workspace

What Is in This Section?

This section contains the following topics:

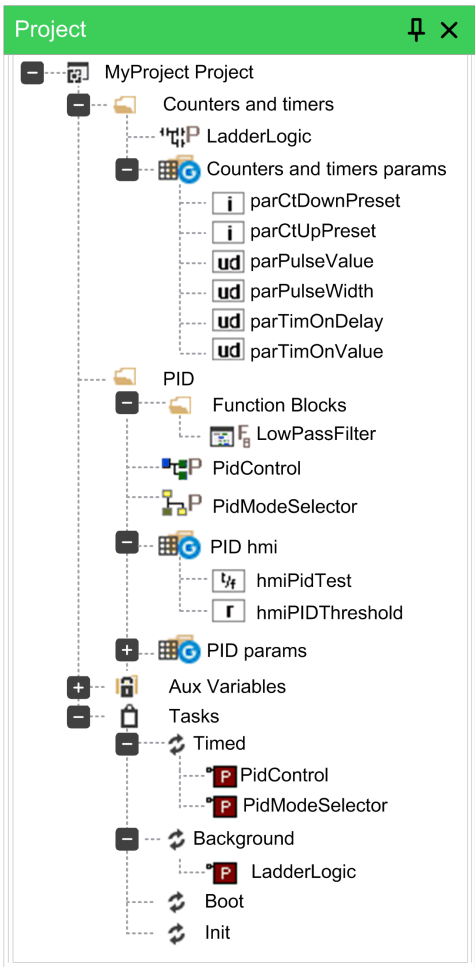
Topic	Page
Overview	220
Enable Custom Workspace Into an Existing Project	221
Workspaces Migration	221
Custom Workspace Basic Units	221
Custom Workspace Operations	222
Workspace Elements with Limitations	223

Overview

Description

The custom workspace functionalities allow you to organize your **Project** window tree according to your needs, in order to obtain more efficiency in the management of the project.

The organizational units of the custom workspace are logical having no effects on the PLC code.



Enable Custom Workspace Into an Existing Project

Description

To enable this feature, click the **Use customizable workspace** check box in **Project → Options... → General** tab. Once enabled, the project needs to be reloaded.

For more information, refer to Project Info (*see page 165*).

By default this feature is enabled and customized according to the target device.

Workspaces Migration

Description

Whenever Custom Workspace feature is switched, **PROGRAMMING** reorders the workspace maintaining the user customization by this logic:

Static (old) workspace to custom (new)

Fixed logic units (for example function blocks folder) are converted into new dynamic folders with the same names. Fixed global group units (for example: Mapped variables) are converted into new global dynamic groups with the same names. The global variables that do not belong to any group are grouped into a new group called **Ungrouped global vars**.

Custom (new) workspace to static (old)

The custom units are destroyed and the POU's and global variables are grouped into the default fixed units (for example: function blocks folder and Mapped Variables).

Custom Workspace Basic Units

Description

In the new custom workspace you can work using two different main logic units:

-  **Folder:** this is an optional logical unit that can contain POU's, folders (you can nest folders into another one), and global variables group.
-  **Global variables group:** this is a mandatory logical unit that can only contain global variables. In order to create a global variable, you need to have almost one global variables group defined into your custom workspace.

Custom Workspace Operations

Description

Different operations can be performed in order to optimize the organization of your project.

Creating a folder:

In order to create a folder select the root item of the project tree or, if you want to nest it, an existing folder then right-click **Add → New folder**.

This operation adds a new customizable folder unit ready to be renamed. Default folder name is **New folder**.

Creating a Global variables Group:

To create a global variables group, select the root item of the project tree or, if you want to nest it, an existing folder, select it, then right-click and select **Add → New global variables group**.

This operation adds a new customizable folder unit ready to be renamed. Default folder name is **New var group**.

Rename a unit (folder or Global variables group):

In order to rename a global variables group or a folder, select it, then right-click and select **Rename**.

This operation makes the name of the unit ready to be renamed.

Deleting a unit (folder or Global variables group):

In order to delete a global variables group or a folder, select it, then right-click and select **Delete**.

If the units contains any child, you are prompted for three possibilities:

Step	Action
1	Delete all child elements too (this may impact the PLC).
2	Do not delete child elements, they are moved upwards following the project tree.
3	Cancel the operation and do nothing.

Export all children to library:

To export all elements of a global variables group or a folder, select it, then right-click and select **Export all children to library**.

This operation allows you to export recursively all child elements of the selected item into a library. For more information about new library, refer to Exporting to a Library ([see page 177](#)).

Moving Unit:

You can simply drag units to a different location of the tree in order to organize your project workspace. All children are moved if the parent item is moved, following the original structure.

Moving variables is also possible both from project tree (single selection) and from the variable grid (single and multiple selections). For more information about variables editor, refer to Variables Editor ([see page 262](#)).

Workspace Elements with Limitations

Description

Some elements of the workspace are fixed and not customizable. They are automatically generated by **PROGRAMMING** and no special custom operations are allowed on:

- **Root Project Element:**
You cannot move, rename, or delete this element. It can contain customizable units as children.
- **POUs Children Elements:**
These elements are generated following the structure of the POU they belong to. You cannot move, rename, or delete these elements directly from the tree. For more information about POUs, refer to Program Organization Units ([see page 183](#)).
- **SFC Children Elements:**
These elements follow the previously mentioned rules but especially for the SFC children nodes the rename or delete operations are not allowed also on the POUs that belong to Actions or Transitions elements. For more information about SFC language, refer to Sequential Function Chart (SFC) Editor ([see page 254](#)).
- **Aux Variables Element:**
You cannot move, rename, or delete this element and its children. They are automatically generated by **PROGRAMMING**.
- **Tasks Element:**
You cannot move, rename, or delete these elements. They are automatically generated by **PROGRAMMING**. For more information, refer to Tasks ([see page 199](#)).

Chapter 10

Editing the Source Code

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
10.1	Overview	226
10.2	Instruction List (IL) Editor	227
10.3	Function Block Diagram (FBD) Editor	230
10.4	Ladder Diagram (LD) Editor	236
10.5	Structured Text (ST) Editor	251
10.6	Sequential Function Chart (SFC) Editor	254
10.7	Variables Editor	262

Section 10.1

Overview

Overview

PLC Editors

PROGRAMMING includes five source code editors, which support the whole range of programming languages according to the IEC 61131-3 Standard:

- Instruction List (IL) (*see page 227*).
- Function Block Diagram (FBD) (*see page 230*).
- Ladder Diagram (LD) (*see page 236*).
- Structured Text (ST) (*see page 251*).
- Sequential Function Chart (SFC) (*see page 254*).

The editors, both graphical and text one, support tooltips. By enabling them (*see page 47*), **PROGRAMMING** shows some information about symbols on mouse-over.

Section 10.2

Instruction List (IL) Editor

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	227
Editing Functions	228
Reference to PLC Objects	228
Automatic Error Location	228
Bookmarks	229

Overview

Description

The IL editor allows you to write code and modify POU's using Instruction List (IL):

```
0001 MUL sysIq
0002 SHR 16#04
0003 ADD addIqSq
0004
0005 MUL sysIq
0006 SHR 16#04
0007 ADD addIqSq
```

Editing Functions

Description

The IL editor is endowed with functions common to most editors running on a Windows platform, namely:

- Text selection.
- **Edit → Cut** 
- **Edit → Copy** 
- **Edit → Paste** 
- **Edit → Replace**
- Drag-and-drop of selected text.

Reference to PLC Objects

Description

If you need to add a reference to an existing PLC object, you have two options:

- You can type directly the name of the PLC object.
- You can drag it to a suitable location. For example, global variables can be taken from the **Workspace** window, whereas standard operators and embedded functions can be dragged from the **Libraries** window, whereas local variables can be selected from the local variables editor.

Automatic Error Location

Description

The IL editor also automatically displays the location of compiler errors. To know where a compiler error occurred, double-click the corresponding error line in the **Output** bar.

Bookmarks

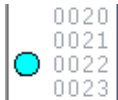
Description

You can set bookmarks to mark frequently accessed lines in your source file. You can remove a bookmark when you no longer need it.

Setting a Bookmark

Move the insertion point to the line where you want to set a bookmark, then press **Ctrl+F2**.

The line is marked in the margin by a light-blue circle.



Bookmarks are managed in **Edit** → **Bookmarks**.... The available commands are:

- **Add/toogle** (Ctrl+F2)
- **Next** (F2)
- **Prev** (Shift+F2)
- **Remove all**

Jumping to Next Bookmark

Press **F2** repeatedly until you reach the desired line.

Jumping to Previous Bookmark

Press **Shift+F2** repeatedly until you reach the desired line.

Removing a Bookmark

Move the cursor to anywhere on the line containing the bookmark, then press **Ctrl+F2**.

Section 10.3

Function Block Diagram (FBD) Editor

What Is in This Section?

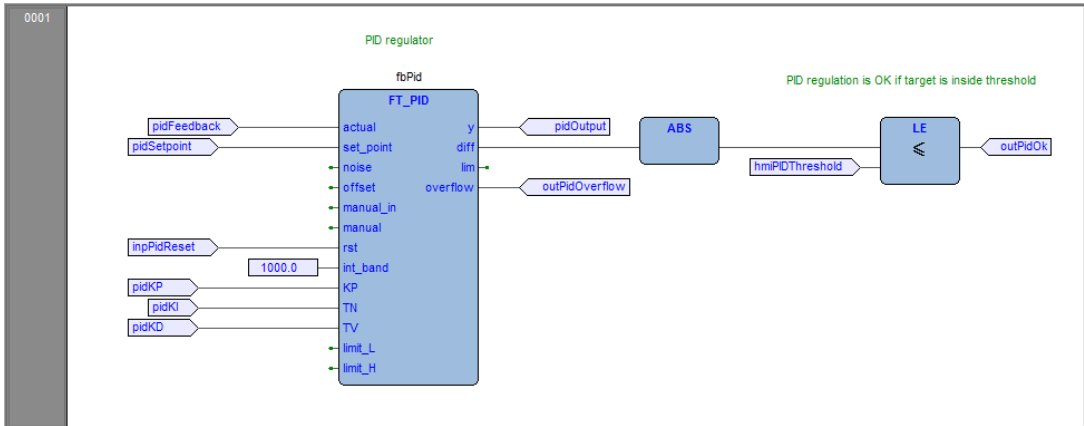
This section contains the following topics:

Topic	Page
Overview	230
Creating a New FBD Document	231
Adding/Removing Networks	231
Labeling Networks	232
Inserting and Connecting Blocks	233
Editing Networks	235
Modifying Properties of Blocks	235
Getting Information on a Block	235
Automatic Error Retrieval	235

Overview

Description

The FBD editor allows you to code and modify POUs using Function Block Diagram (FBD):



For more details, refer to Function Block Diagram language reference ([see page 409](#)).

Creating a New FBD Document

Description

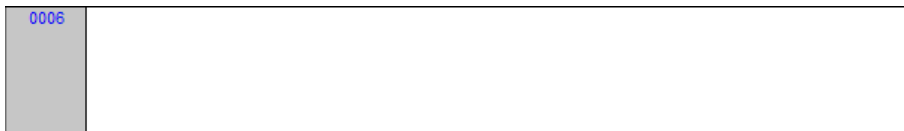
For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit (*see page 184*).
- Creating a Function Block/Function (*see page 185*).
- Editing POUs (*see page 186*).

Adding/Removing Networks

Description

Every POU coded in FBD consists of a sequence of networks. A network is defined as a maximal set of interconnected graphic elements. The upper and lower bounds of every network are fixed by two straight lines while each network is delimited on the left by a gray area containing the network number:



You can perform the following operations on networks:

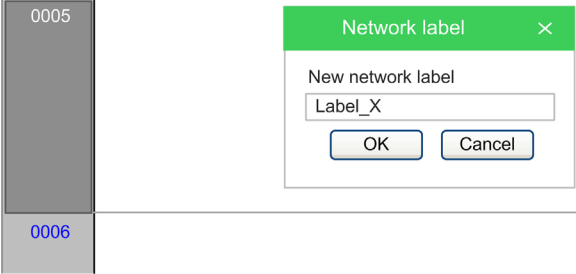
- To add a new blank network, click **Scheme** → **Network** → **New** and select the position of the new network: **Top**, **Bottom**, **Before**, or **After**.
- To delete a network, select it and press **Delete**.
- To display a background grid which helps you to align objects, click **View** → **Grid** .
- To add a comment, click **Scheme** → **Object** → **New** → **Comment**.

Labeling Networks

Description

You can modify the usual order of execution of networks through a jump statement, which transfers the program control to a labeled network.

To assign a label to a network:

Step	Action
1	Select the network.
2	Apply one of the following operations: • Click Scheme → Network → Label. • Double-click the gray area containing the network number.
3	A dialog box appears, which lets you enter the label you want to associate with the selected network: 
4	Click OK. The label is displayed in the top left-hand corner of the selected network. 

Inserting and Connecting Blocks

Overview

This paragraph presents how to build a network.

Inserting Blocks

Add a block to the blank network, by applying one of the following operations:

- Open the **Object browser** window, by applying one of the following operations:
 - In the menu, click **Scheme → Object → New → Function Block**.
 - In the FBD toolbar, click .

If the block is a constant, a return statement, or a jump statement, you can directly click the relevant buttons in the **FBD** toolbar (*see page 158*)

Then choose one item from the list and click **OK**.

- Drag the selected object from the suitable location. For example, global variables can be taken from the **Workspace** window, whereas standard operators and embedded functions can be dragged from the **Libraries** window, whereas local variables can be selected from the local variables editor.

Repeat until you have added all the blocks to the network.

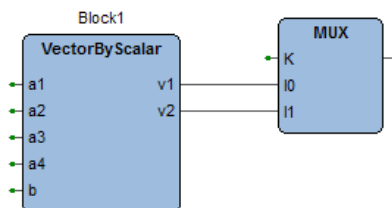
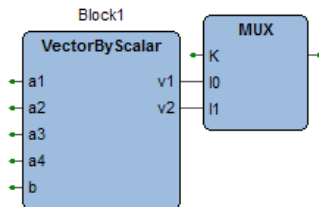
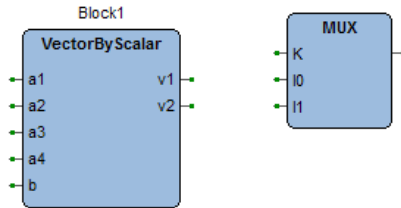
Connecting Blocks

To connect blocks manually:

- Enable the manual connection mode by applying one of the following operations:
 - In the menu, click  **Edit → Connection mode**.
 - In the FBD toolbar, click .
 - Press **Space** key.
- Click once the source pin, then move the mouse pointer to the destination pin: the FBD editor draws a logical wire from the former to the latter.

To connect blocks automatically:

- Enable the automatic connection mode by applying one of the following operations:
 - In the menu, click  **Scheme → Auto connect**.
 - In the code editor, right-click and click  **Auto connect**.
- Then select one block, drag it close to the other one to let the corresponding pins coincide. The FBD editor automatically draws the logical wires.



Deleting Blocks

To delete a block, select it and press **Delete** key.

When you delete a block, its connections are not removed automatically, but they become invalid and they are redrawn red. Click **Scheme → Delete invalid connection**.

Editing Networks

Description

The FBD editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

- Selection of a block.
- Selection of a set of blocks by pressing **Shift+Left** button and by drawing a frame including the blocks to select.
- **Edit → Cut** , **Edit → Copy** , **Edit → Paste**  operations of a single block as well as of a set of blocks.
- Drag-and-drop.

Modifying Properties of Blocks

Description

- Click  **Scheme → Increment pins** to increment the number of input pins of some operators and embedded functions.
- Click  **Scheme → Enable EN/ENO pins** to display the enable input and output pins.
- Click **Scheme → Object → Instance name** or click **Scheme → Object properties** to modify the name of an instance of a function block.

For more information, refer to Modifying Properties of Blocks ([see page 245](#)) in Ladder Diagram (LD) Editor section.

Getting Information on a Block

Description

You can always get information on a block by selecting it and then applying one of the following operations:

- Click **Scheme → Object → Open source**  to open the source code of a block.
- Click **Scheme → Object properties** to see properties and input/output pins of the selected block.

Automatic Error Retrieval

Description

The FBD editor also automatically displays the location of compiler errors. To reach the block where a compiler error occurred, double-click the corresponding error line in the **Output** bar.

Section 10.4

Ladder Diagram (LD) Editor

What Is in This Section?

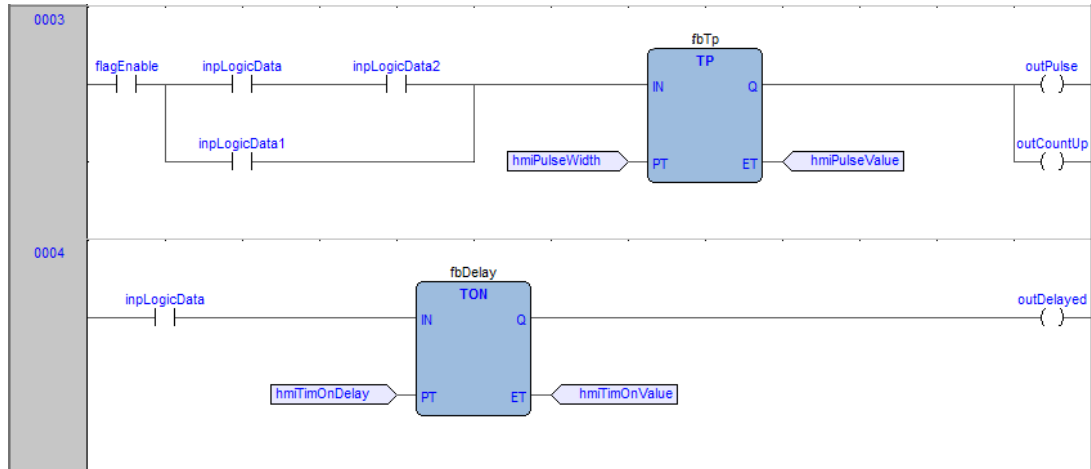
This section contains the following topics:

Topic	Page
Overview	237
Creating a New LD Document	237
Adding/Removing Networks	238
Labeling Networks	239
Inserting Contacts	240
Inserting Coils	242
Inserting Blocks	243
Editing Coils and Contacts Properties	244
Editing Networks	244
Modifying Properties of Blocks	245
Getting Information on a Block	246
Automatic Error Retrieval	246
Inserting Variables	247
Inserting Constants	247
Inserting Expression	248
Comments	249
Branches	250

Overview

Description

The LD editor allows you to code and modify POU's using Ladder Diagram (LD):



For more details, refer to Ladder Diagram language reference ([see page 414](#)).

Creating a New LD Document

Description

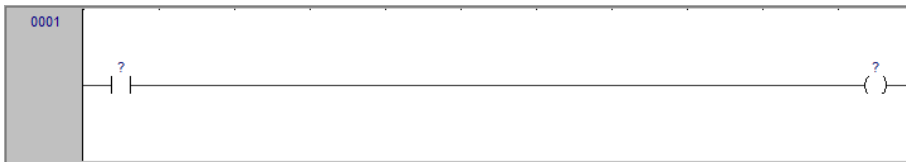
For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit ([see page 184](#)).
- Creating a Function Block/Function ([see page 185](#)).
- Editing POUs ([see page 186](#)).

Adding/Removing Networks

Description

Each POU coded in LD consists of a sequence of networks. A network is defined as the set of interconnected graphic elements. The upper and lower bounds of every network are fixed by two straight lines while each network is delimited on the left by a gray area containing the network number.



On each LD network, the right and the left power rail are represented, according to the LD language indication.

On the new LD network, a horizontal line links the two power rails. It is called the “power link”. On this link, all the LD elements (contacts, coils, and blocks) of the network are placed.

You can perform the following operations on networks:

- To add a new blank network, click **Scheme → Network → New**, or click one of the equivalent

buttons  in the **Network** toolbar.

- To display a background grid which helps you to align objects, click **View → Grid** .

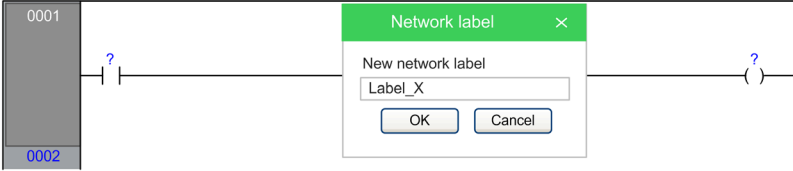
- To add a comment, click **Scheme → Object → New Comment**  or press **Shift+M**.

Labeling Networks

Description

You can modify the usual order of execution of networks through a jump statement, which transfers the program control to a labeled network.

To assign a label to a network:

Step	Action
1	Select the network.
2	Apply one of the following operations: - Click Scheme → Network → Label. - Double-click the gray area containing the network number.
3	A dialog box appears, which lets you enter the label you want to associate with the selected network: 
4	Click OK . The label is displayed in the top left-hand corner of the selected network: 

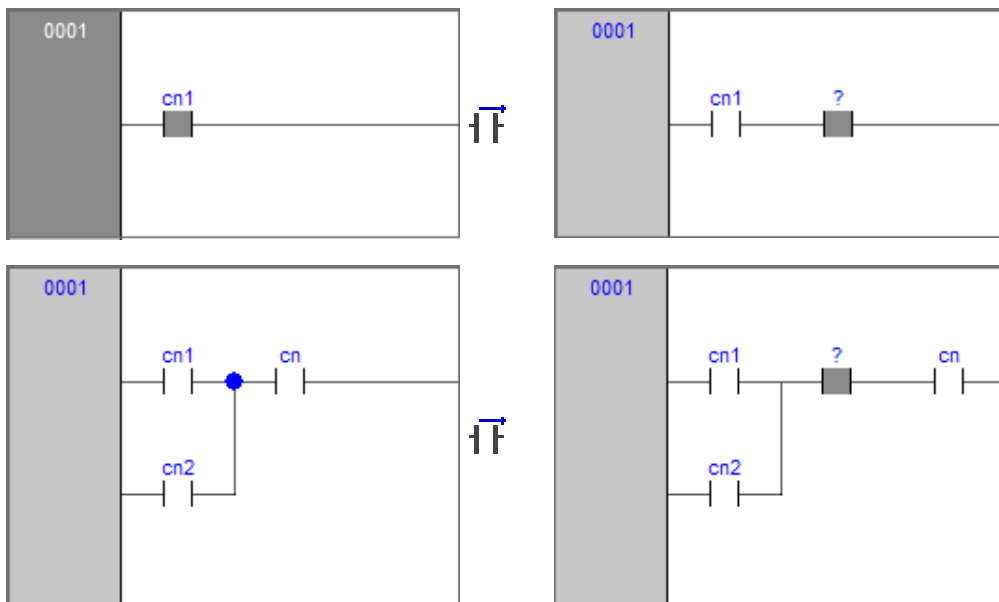
Inserting Contacts

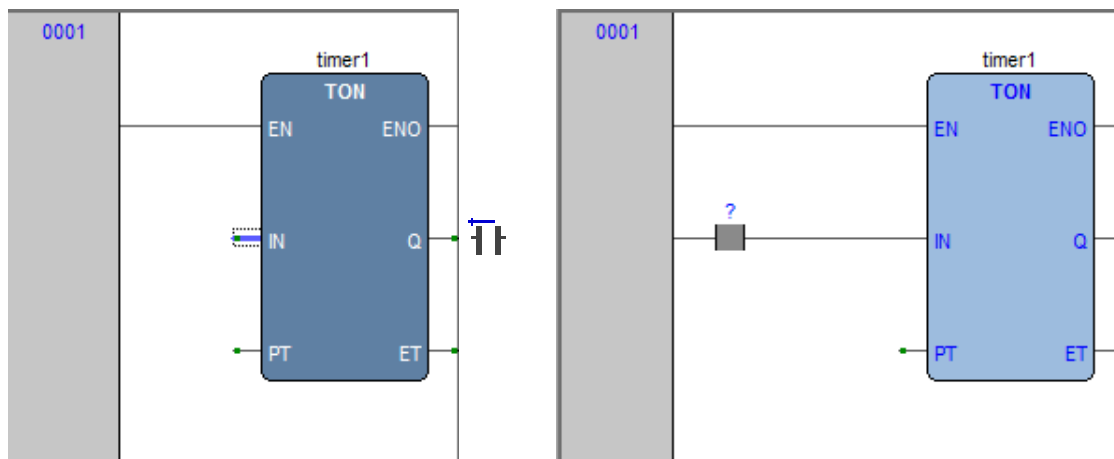
Description

To insert new contacts on the network, apply one of the following procedures:

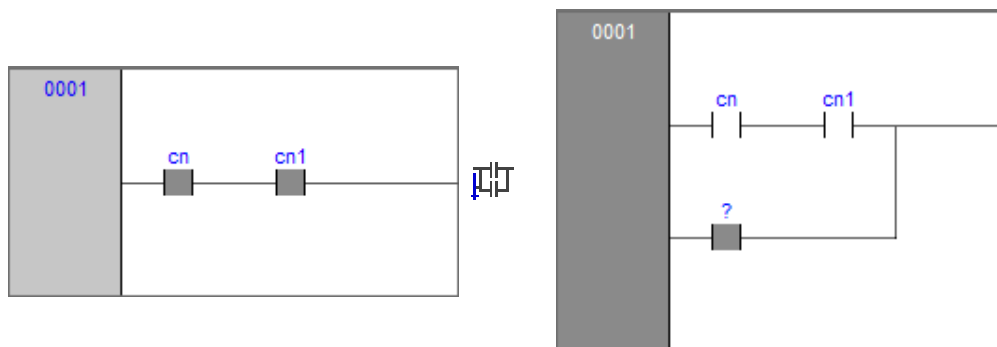
- Drag a boolean variable to the desired place over an object. For example, global variables can be taken from the **Workspace** window, whereas local variables can be selected from the local variables editor. Contacts inserted with drag and drop will always be inserted in series after the destination object.
- Select a contact, a block, a pin of block, or a connection point that acts as the insertion point. Insert the new contact choosing between the connection type (serial or parallel) and choosing the position (before or after the currently selected object) by using the **Scheme → Object → New**.

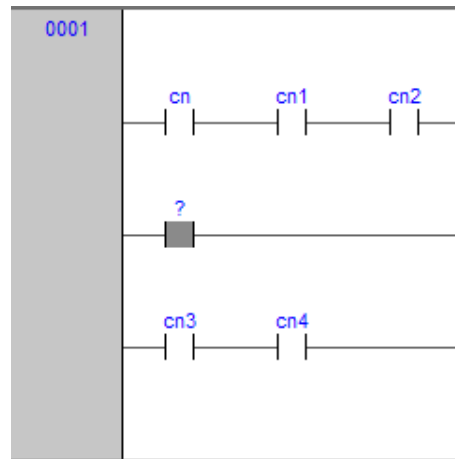
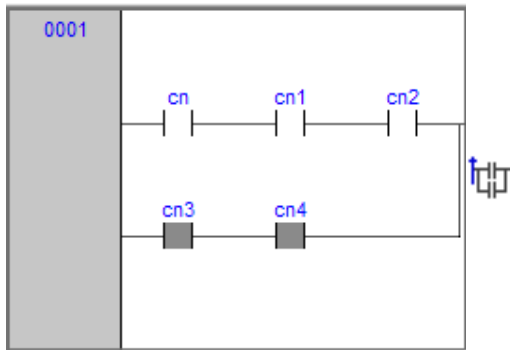
For serial insertion, the new contact is inserted on the left or right side of the selected contact/block or in the middle of the selected connection depending on the element selected before the insertion. Examples of serial insertion:





For parallel insertions, several contacts can be selected before performing the insertion; the new contact is inserted above or below the group of selected contacts. Examples of parallel insertions





Inserting Coils

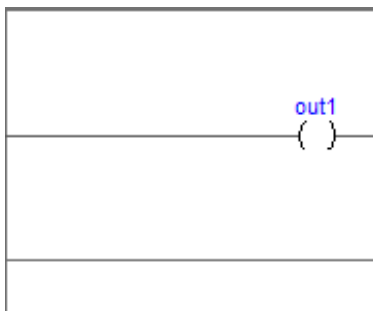
Description

To insert new coils on the network, apply one of the following operations:

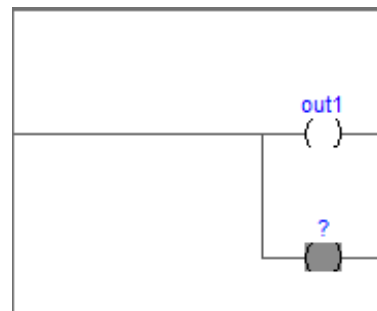
- Drag a boolean variable on the network, over an existing output of the network (coil, return, jump). For example, global variables can be taken from the **Workspace** window, whereas local variables can be selected from the local variables editor.

- Click  **Scheme** → **Object** → **New** → **Coil**.

The new coil is inserted to the right power rail. If other coils, return or jumps are already present in the network, the new coil is added in parallel with the previous ones.







Inserting Blocks

Description

To insert blocks on the network, apply one of the following operations:

- Select a contact, connection, or block then click  **Scheme** → **Object** → **New** → **Block**, then the **Browser object** window appears. Choose one item from the list.
- Drag the selected object (from the **Workspace** window, the **Libraries** window or the local variables editor) over the desired connection.

If the object has at least one **BOOL** input and one **BOOL** output pins, they are connected to the power link (and it will possible to add **EN/ENO** pins later with the provided command); otherwise the **EN/ENO** pins are automatically added.

Operators, functions, and function blocks can only be inserted into an **LD** network on the main power link, or on the power link of a branch (so they cannot be inserted in parallel of a contact); it is also not possible to create a contact in parallel of a block.

If a block has a **BOOL** input pin, it is possible to create another logical sub-network of contacts and blocks before it; otherwise, you can connect only variables, constants, or expressions (that nevertheless can be connected to **BOOL** pins) to non-**BOOL** input pins.

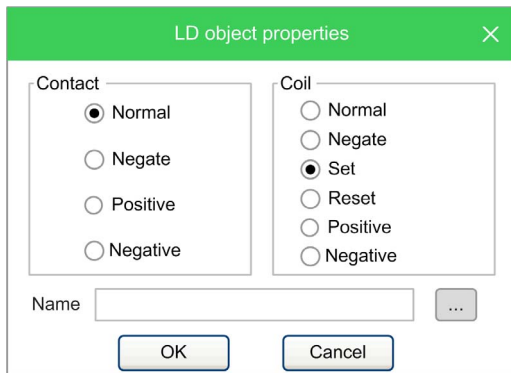
Editing Coils and Contacts Properties

Description

The type of a contact (normal, negated, positive, negative) or a coil (normal, negated, set, reset, positive, negative) can be changed by one of the following operations:

- Double-click the element (contact or coil).
- Select the element and then press the **Enter** key.
- Select the element, activate the pop-up menu, then select **Properties**.

A relevant dialog box appears. Select the desired element type from the list displayed and then click **OK**.



Otherwise, select the desired contact or coil, and modify its type using the six provided buttons in the **LD** toolbar or the six commands in the **Scheme** menu.

Editing Networks

Description

The **LD** editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

- Selection of a block.
- Selection of a set of adjacent contacts by pressing **Ctrl+Left** button on each contact to select; if the selection spans across different parallel branches, more contacts are automatically added in the selection.
- **Edit → Cut** , **Edit → Copy** , **Edit → Paste**  operations of a single block as well as of a set of blocks.
- **Drag-and-drop** of the selected object or group to move it inside or outside the current network.

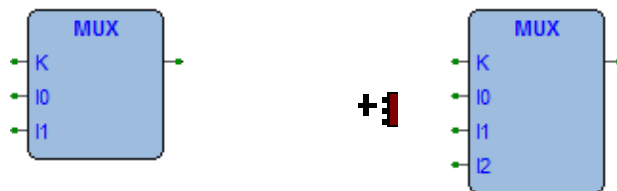
Adding, moving, deleting, or copy/pasting objects will automatically recalculate the layout of the network objects; because of this, it is not possible to manually “draw” connection lines or freely placing objects without connecting them to the network.

Modifying Properties of Blocks

Description

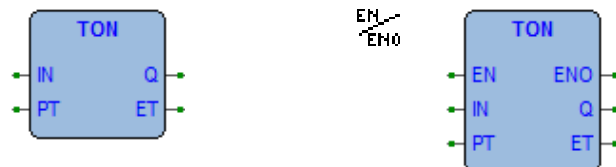
- Click **Scheme → Increment pins**  to increment the number of input pins of some operators and embedded functions.

NOTE: You can also remove pins by clicking **Decrease pins** .



- Click  **Scheme → Enable EN/ENO pins** to display the enable input and output pins. **EN/ENO** pins can be removed only if the selected block has at least one **BOOL** input and one **BOOL** output; otherwise, they are automatically added when creating the block and it will not be possible to remove them (the **Enable EN/ENO pins** command is disabled). If a block has more than one **BOOL** output pin, it is possible to choose which pin brings the **signal** out of the block and so continue the power link: select the desired output pin and click the

Scheme → Set output line  menu command.



- Click **Scheme** → **Object properties** to modify the name of an instance of a function block.

FBD object properties

Instance nameTON_1

☐ En/Eno

Function blocks

I/O	Neg	Name	Type	Description
In		IN	BOOL	Timer input source
In	n/a	PT	UDINT	Preset time value [ms]
Out		Q	BOOL	Timer output
Out	n/a	ET	UDINT	Timer current value [ms]

OK

Cancel

Getting Information on a Block

Description

You can always get information on a block that you added to an LD document, by selecting it and then applying one of the following operations:

- Click **Scheme** → **Object** → **Open source**  to open the source code of a block.
- Click **Scheme** → **View PLC Object properties** in the menu to see properties and input/output pins of the selected block.

Automatic Error Retrieval

Description

The LD editor also automatically displays the location of compiler errors. To reach the block where a compiler error occurred, double-click the corresponding error line in the **Output** bar.

Inserting Variables

Description

To connect a variable to an input or output pin of a block, apply one of the following procedures:

- Select the pin of a block, and then click the  **Scheme → Object → New → Variable** menu command; then double-click the new variable object (or press **Enter** key) and enter the variable name.
- Drag the selected variable (from the **Workspace** window, the **Libraries** window or the local variables editor) over the desired pin of a block.

Inserting Constants

Description

To connect a numeric constant to an input pin of block, select the pin and click the  **Scheme → Object → New → Constant** menu command; then double-click the new constant object (or press **Enter** key) and enter the numeric constant value.

Inserting Expression

Description

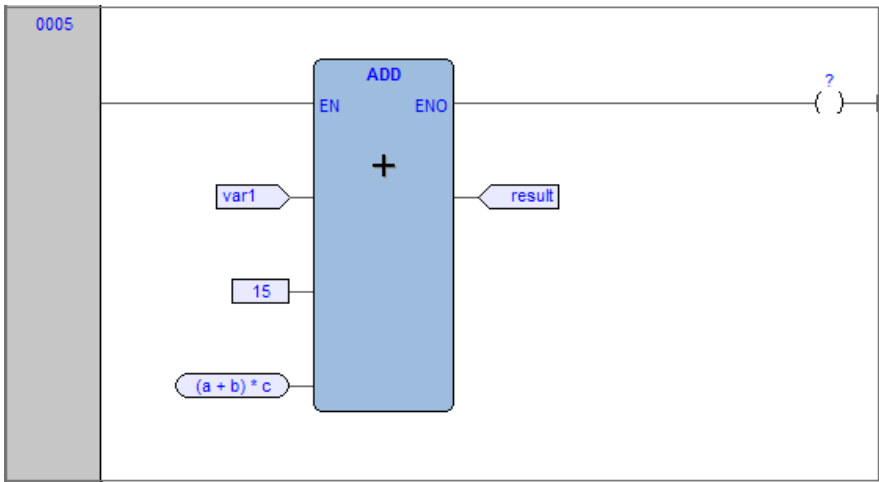
To connect a complex expression to an input pin of block, select the pin and click the  **Scheme** → **Object** → **New** → **Expression** menu command; then double-click the new expression object (or press **Enter** key) and enter any **ST** expression.

For example:

$(a+b) * c$

TO_INT (n)

ADR (x)

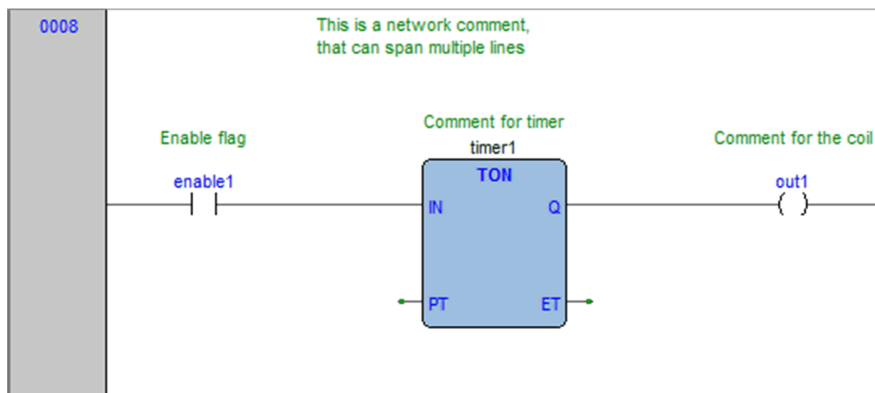


Comments

Description

It is possible to insert two types of comments:

- **Network** comments: activate the network by clicking the header on the left or inside the grid (but without selecting any object), and then click the  **Scheme → Object → New → Comment** menu command. The network comment is displayed at the top of the network, and if necessary is expanded to show all the text lines of the comment.
- **Object** comments: they are activated with the menu command in **View → Show comments for objects**. Above any contact, function block, or coil, the description of the associated PLC variable (if present) is initially displayed. With the **Comment** command, you can modify it to enter a specific object comment that overrides the PLC variable description.



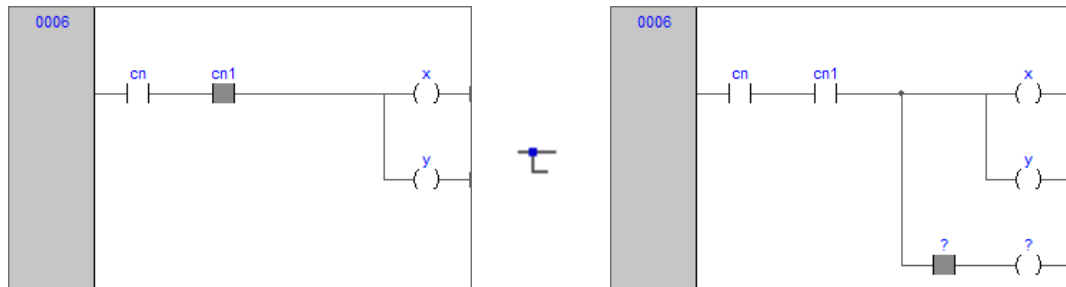
Branches

Description

The main power line can be branched to create sub-networks that can be further branched themselves. To add a branch, select the object after you want to create the branch and then click the  **Scheme → Object → New → Branch** menu command.

The start of the new branch is marked as a large dot on the source line; deleting the objects on a branch deletes the branch itself.

Selecting an object on a branch effectively selects the branch, so for example selecting a contact on a branch and then clicking the  **Scheme → Object → New → Coil** adds the coil on the branch instead of adding it on the main power line.



Section 10.5

Structured Text (ST) Editor

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	251
Creating and Editing ST Objects	252
Editing Functions	252
Reference to PLC Objects	252
Automatic Error Location	252
Bookmarks	253

Overview

Description

The ST editor allows you to code and modify POUs using Structured Text (ST):

```
0001
0002 IqDW := sysIq * sysIq ;
0003 addIqSq := addIqSq + SHR( IqDW, 16#04 ) ;
0004
0005 IdDW := sysId * sysId ;
0006 addIdSq := addIdSq + SHR( IdDW, 16#04 ) ;
0007
0008 IF a > b THEN
0009     a := c;
0010     n := a * b * c;
0011 END_IF;
0012
```

For more details, refer to Structured Text language reference ([see page 419](#)).

Creating and Editing ST Objects

Description

For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit (*see page 184*).
- Creating a Function Block/Function (*see page 185*).
- Editing POUs (*see page 186*).

Editing Functions

Description

The ST editor is endowed with functions common to most editors running on a Windows platform, namely:

- Text selection.
- **Edit → Cut**  .
- **Edit → Copy**  .
- **Edit → Paste**  .
- **Edit → Replace**.
- Drag-and-drop of selected text.

Reference to PLC Objects

Description

If you need to add to your ST code a reference to an existing PLC object, you have two options:

- You can type directly the name of the PLC object.
- You can drag it to a suitable location. For example, global variables can be taken from the **Workspace** window, whereas embedded functions can be dragged from the **Libraries** window, whereas local variables can be selected from the local variables editor.

Automatic Error Location

Description

The ST editor also automatically displays the location of compiler errors. To know where a compiler error has occurred, double-click the corresponding error line in the **Output** bar.

Bookmarks

Description

You can set bookmarks to mark frequently accessed lines in your source file. Once a bookmark is set, you can use a keyboard command to move to it. You can remove a bookmark when you no longer need it.

Setting a Bookmark

Move the insertion point to the line where you want to set a bookmark, then press **Ctrl+F2**.

The line is marked in the margin by a light-blue circle:



Bookmarks are managed in **Edit** → **Bookmarks**.... The available commands are:

- **Add/toogle** (**Ctrl+F2**)
- **Next** (**F2**)
- **Prev** (**Shift+F2**)
- **Remove all**

Jumping to Next Bookmark

Press **F2** repeatedly until you reach the desired line.

Jumping to Previous Bookmark

Press **Shift+F2** repeatedly until you reach the desired line.

Removing a Bookmark

Move the cursor to anywhere on the line containing the bookmark, then press **Ctrl+F2**.

Section 10.6

Sequential Function Chart (SFC) Editor

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	254
Creating a New SFC Document	254
Inserting a New SFC Element	255
Connecting SFC Elements	255
Assigning an Action to a Step	256
Specifying a Conditional Transition	258
Assigning Conditional Code to a Transition	259
Specifying the Destination of a Jump	261
Editing SFC Networks	261

Overview

Description

The SFC editor allows you to code and modify POUs using Sequential Function Chart (SFC):

For more information about SFC editor features, refer to SFC Toolbar ([see page 159](#)).

For more details, refer to Sequential Function Chart language reference ([see page 430](#)).

Creating a New SFC Document

Description

For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit ([see page 184](#)).
- Creating a Function Block/Function ([see page 185](#)).
- Editing POUs ([see page 186](#)).

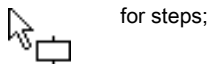
Inserting a New SFC Element

Description

You can insert three type of SFC elements:

- Click  **Scheme → Object → New → Step.**
- Click  **Scheme → Object → New → Transition.**
- Click  **Scheme → Object → New → Jump.**

In either case, the mouse pointer changes to:



for steps;



for transitions;



for jumps.

Connecting SFC Elements

Description

Follow this procedure to connect SFC blocks:

- Click  **Edit → Connection mode**, or simply press the space bar on your keyboard. Click once the source pin, then move the mouse pointer to the destination pin: the SFC editor draws a logical wire from the former to the latter.
- Alternatively, you can enable the auto connection mode by clicking  **Scheme → Auto connect**. Then take the two blocks, and drag them close to each other to let the respective pins coincide, which makes the SFC editor draw automatically the logical wire.

Assigning an Action to a Step

Description

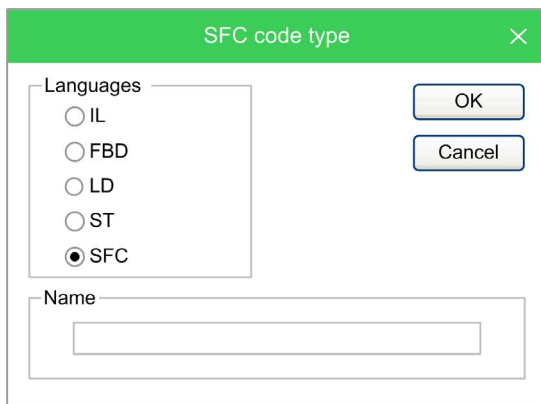
This paragraph explains how to implement an action and how to assign it to a step.

Writing the Code of an Action

Start by opening an editor, applying one of the following procedures:

- Click  **Scheme** → **Code object** → **New action**.
- Right-click on the name of the SFC POU in the **Workspace** window  **New action**.

In either case, **PROGRAMMING** displays a dialog box:



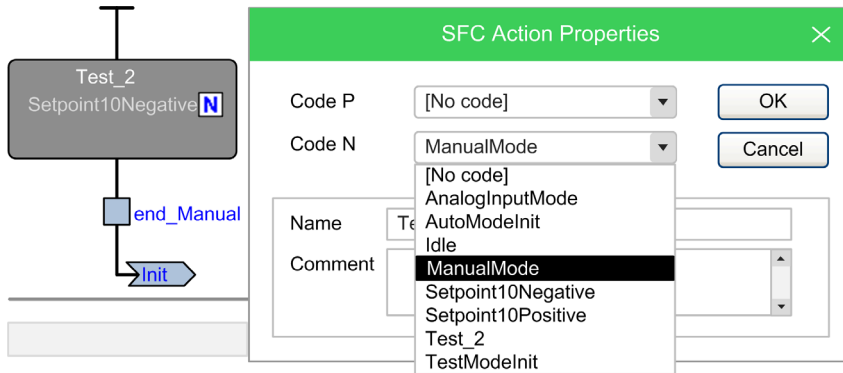
Select one of the languages and type the name of the new action in the text box at the bottom of the dialog box. Then either confirm by clicking **OK**, or quit by clicking **Cancel**.

If you click **OK**, **PROGRAMMING** opens automatically the editor associated with the language you selected in the previous dialog box and you are ready to type the code of the new action.

You are not allowed to declare new local variables, as the module you are now editing is a component of the original SFC module, which is the POU where local variables can be declared. The scope of local variables extends to all the actions and transitions making up the SFC diagram.

Assigning an Action to a Step

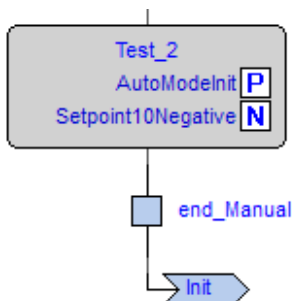
When you have finished writing the code, double-click the step you want to assign the new action to. This causes the following dialog box to appear.



From the list displayed in the **Code N** box, select the name of the action you want to execute if the step is active. You may also choose, from the list displayed in the **Code P (Pulse)** box, the name of the action you want to execute each time the step becomes active (that is, the action is executed only once per step activation, regardless of the number of cycles the step remains active). Confirm the assignments by clicking **OK**.

In the SFC schema, actions to step assignments are represented by letters on the step block:

- Action **N** by letter N;
- Action **P** by letter P.



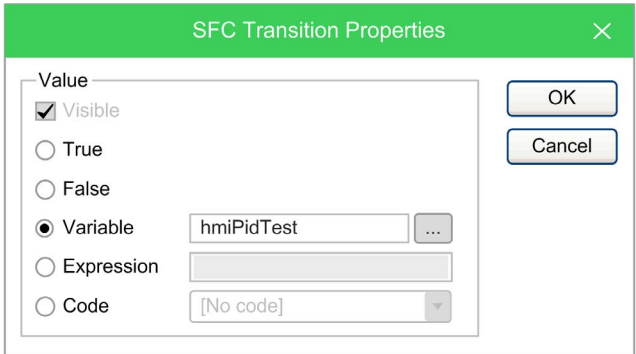
If later you need to edit the source code of the action, you can double-click these letters. Alternatively, you can double-click the name of the action in the **Actions** folder of the **Workspace** window.

Specifying a Conditional Transition

Description

A transition condition can be assigned through a constant, a variable, or a piece of code. This paragraph explains how to use the first two means while conditional code is discussed in the next paragraph.

First of all, double-click the transition you want to assign a condition to. This causes the following dialog box to appear:

The image shows a dialog box titled "SFC Transition Properties" with a green header bar and a close button (X) in the top right corner. The dialog is divided into two main sections. The left section, labeled "Value", contains five radio button options: "Visible" (which is checked), "True", "False", "Variable", and "Expression". Below the "Variable" option, there is a text input field containing "hmiPidTest" and a small "..." button to its right. Below the "Expression" option, there is an empty text input field. Below the "Code" option, there is a text input field containing "[No code]" and a small dropdown arrow to its right. The right section of the dialog contains two buttons: "OK" and "Cancel".

Select **True** if you want this transition to be constantly cleared, **False** if you want the PLC program to keep executing the preceding block.

Instead, if you select **Variable** the transition depends on the value of a boolean variable. Click the corresponding bullet to make the text box to its right available, and to specify the name of the variable.

You can also make use of the objects browser, that you can invoke by clicking the  **Browse** button.

Click **OK** to confirm, or **Cancel** to quit without applying changes.

Assigning Conditional Code to a Transition

Description

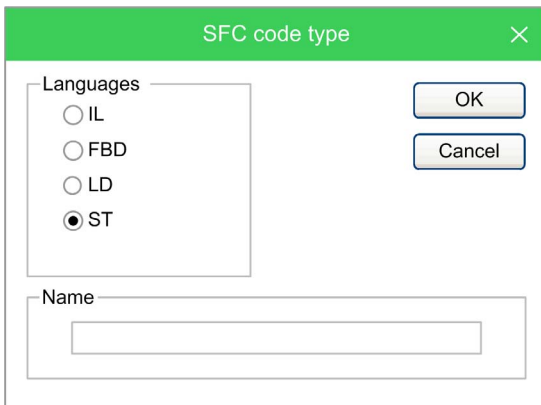
This paragraph explains how to specify a condition through a piece of code, and how to assign it to a transition.

Writing the Code of a Condition

Start by opening an editor, applying one of the following procedures:

- Click  **Scheme** → **Code object** → **New transition code**.
- Right-click on the name of the SFC POU in the **Workspace** window  **New transition**.

In either case, **PROGRAMMING** displays a dialog box:



The dialog box is titled "SFC code type" with a close button (X) in the top right corner. It contains two main sections. The first section, labeled "Languages", has a list of radio buttons: IL, FBD, LD, and ST. The ST option is selected. To the right of this list are "OK" and "Cancel" buttons. The second section, labeled "Name", contains a text input field.

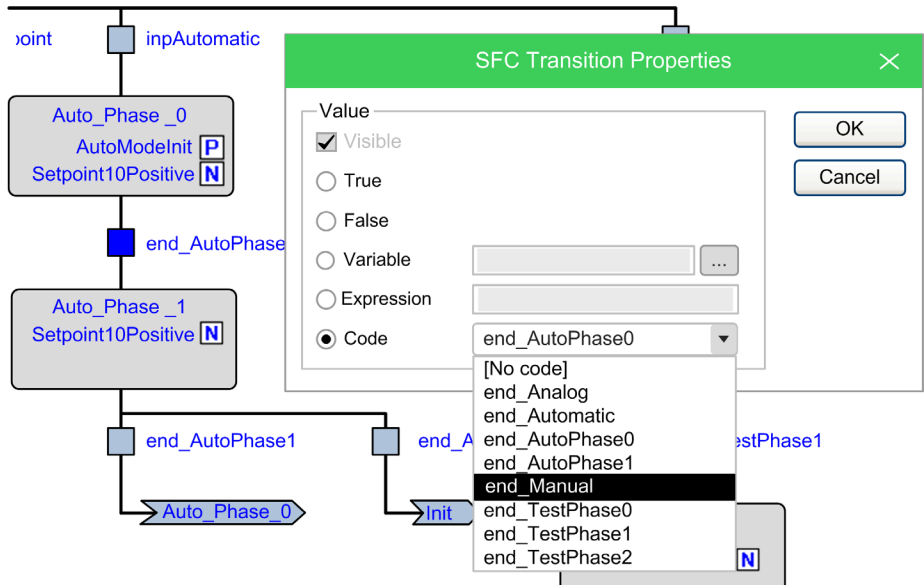
Select one of the languages and type the name of the new condition in the text box at the bottom of the dialog box. Then either confirm by clicking **OK**, or quit by clicking **Cancel**.

If you click the **OK** button, **PROGRAMMING** opens automatically the editor associated with the language you selected in the previous dialog box and you can type the code of the new condition.

You are not allowed to declare new local variables, as the module you are now editing is a component of the original SFC module, which is the POU where local variables can be declared. The scope of local variables extends to all the actions and transitions making up the SFC diagram.

Assigning a Condition to a Transition

When you have finished writing the code, double-click the transition you want to assign the new condition to. This causes the following dialog box to appear:



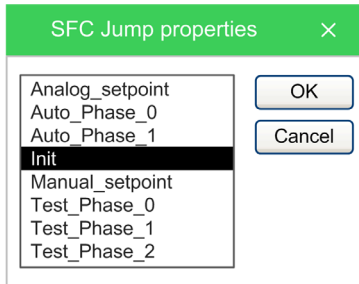
Select the name of the condition you want to assign to this step. Then confirm by clicking **OK**.

If later you need to edit the source code of the condition, you can double-click the name of the transition in the **Transitions** folder of the **Workspace** window.

Specifying the Destination of a Jump

Description

To specify the destination step of a jump, double-click the jump block in the **Chart** area. This opens the dialog box presented below, listing the name of all the existing steps. Select the destination step, then either click **OK** to confirm or **Cancel** to quit.



Editing SFC Networks

Description

The SFC editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

- Selection of a block.
- Selection of a set of blocks by pressing **Ctrl + left** button.
- **Edit → Cut** , **Edit → Copy** , **Edit → Paste**  operations of a single block as well as of a set of blocks.
- Drag-and-drop.

Section 10.7

Variables Editor

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	262
Opening a Variables Editor	263
Creating a New Variable	264
Editing Variables	264
Deleting Variables	267
Sorting Variables	267
Copying Variables	267

Overview

Description

PROGRAMMING includes a graphical editor for both global and local variables that supplies an interface for declaring and editing variables: the tool takes care of translating the contents of these editors into syntactically correct IEC 61131-3 source code.

As an example, consider the contents of the Global variables editor represented in the following figure:

	Name	Type	Address	Array	Init value	Attribute
1	gA	BOOL	Auto	No	TRUE	---
2	gB	REAL	Auto	[0...4]		---
3	gC	REAL	%MD60.20	No	1.0	---
4	gD	INT	Auto	No	-74	CONSTANT

The corresponding source code is represented like this:

```
VAR_GLOBAL
  gA : BOOL := TRUE;
  gB : ARRAY[ 0..4 ] OF REAL;
  gC AT %MD60.20 : REAL := 1.0;
END_VAR
VAR_GLOBAL CONSTANT
  gD : INT := -74;
END_VAR
```

Opening a Variables Editor

Opening the Global Variables Editor

To open the Global variables editor, double-click **Global variables** in the project tree:

Project

MyProject Project

Programs

Function blocks

Functions

Global variables

Automatic variables

Mapped variables

Constants

Retain variables

	Name	Type	Address
1	parCtDownPreset	UDINT	Auto
2	parCtUpPreset	UDINT	Auto
3	parPulseValue	UDINT	Auto
4	parPulseWidth	UDINT	Auto
5	parTimOnDelay	UDINT	Auto
6	parTimOnValue	UDINT	Auto
7	pidKD	REAL	Auto
8	pidKI	REAL	%MD1

NOTE: If you use the customizable workspace ([see page 165](#)), the global variables are accessible under the icon  **Global variables** group.

Opening a Local Variables Editor

To open a local variables editor, double-click the Program Organization Unit that contains the local variables you want to edit:

Project

MyProject Project

Programs

LadderLogic

PidControl

PidModeSelector

Temperature_Control

Function blocks

Local variables

	Name	Type	Address
1	fbDelay	TON	Auto
2	fbCtu	CTU_DINT	Auto
3	fbCtd	CTU_DINT	Auto
4	fbTp	TP	Auto
5	ettp	UDINT	Auto

NOTE: If you use the customizable workspace ([see page 165](#)), the local variables are accessible under the POU, in the **Local variables** group.

Creating a New Variable

Description

Create a new variable, by applying one of the following operations:

- In the menu, click **Variables → Insert**.
- In the Project toolbar, click .
- Press the **Ctrl+Shift+Insert** keys.

Editing Variables

Description

Follow this procedure to edit the declaration of a variable in a variables editor (the following steps are optional and you will skip most of them when editing a variable):

Step	Action																														
1	Edit the name of the variable by entering the new name in the corresponding cell: <table><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th><th>Array</th></tr><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td><td>No</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td>Ungrouped_vars</td><td>[0...4]</td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyMapped_vars</td><td>No</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyConstants_vars</td><td>No</td></tr></table>		Name	Type	Address	Group	Array	1	gA	BOOL	Auto	Ungrouped_vars	No	2	gB	REAL	Auto	Ungrouped_vars	[0...4]	3	gC	REAL	%MD60.20	MyMapped_vars	No	4	gD	INT	Auto	MyConstants_vars	No
	Name	Type	Address	Group	Array																										
1	gA	BOOL	Auto	Ungrouped_vars	No																										
2	gB	REAL	Auto	Ungrouped_vars	[0...4]																										
3	gC	REAL	%MD60.20	MyMapped_vars	No																										
4	gD	INT	Auto	MyConstants_vars	No																										
2	Modify the variable type, either by editing the type name in the corresponding cell or by clicking the button in that cell and select the desired type from the list that pops up: <table><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th></tr><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td>Ungrouped_vars</td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyMapped_vars</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyConstants_vars</td></tr></table>		Name	Type	Address	Group	1	gA	BOOL	Auto	Ungrouped_vars	2	gB	REAL	Auto	Ungrouped_vars	3	gC	REAL	%MD60.20	MyMapped_vars	4	gD	INT	Auto	MyConstants_vars					
	Name	Type	Address	Group																											
1	gA	BOOL	Auto	Ungrouped_vars																											
2	gB	REAL	Auto	Ungrouped_vars																											
3	gC	REAL	%MD60.20	MyMapped_vars																											
4	gD	INT	Auto	MyConstants_vars																											

Step	Action																																				
3	Edit the address of the variable by clicking the button in the corresponding cell and entering the required information in the window that shows up. In the case of global variables, this operation may change the position of the variable in the project tree: <table><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th></tr><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td>Ungrouped_vars</td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyMapped_vars</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyConstants_vars</td></tr></table> Variable address ☐ Automatic address Size ☐ Bit ☐ Byte (8 bit) ☐ Word (16 bit) ☒ Double word (32 bit) Location ☐ Input ☐ Output ☒ Memory Data block 60 Index 20 OK Cancel		Name	Type	Address	Group	1	gA	BOOL	Auto	Ungrouped_vars	2	gB	REAL	Auto	Ungrouped_vars	3	gC	REAL	%MD60.20	MyMapped_vars	4	gD	INT	Auto	MyConstants_vars											
	Name	Type	Address	Group																																	
1	gA	BOOL	Auto	Ungrouped_vars																																	
2	gB	REAL	Auto	Ungrouped_vars																																	
3	gC	REAL	%MD60.20	MyMapped_vars																																	
4	gD	INT	Auto	MyConstants_vars																																	
4	In the case of global variables, you can assign the variable to a group, by selecting it from the list which opens when you click the corresponding cell. This operation changes the position of the variable in the project tree: <table><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th><th>Array</th></tr><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td><td>No</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td></td><td>[0...4]</td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyConstants_vars</td><td>No</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyMapped_vars</td><td>No</td></tr><tr><td></td><td></td><td></td><td></td><td>Ungrouped_vars</td><td></td></tr></table>		Name	Type	Address	Group	Array	1	gA	BOOL	Auto	Ungrouped_vars	No	2	gB	REAL	Auto		[0...4]	3	gC	REAL	%MD60.20	MyConstants_vars	No	4	gD	INT	Auto	MyMapped_vars	No					Ungrouped_vars	
	Name	Type	Address	Group	Array																																
1	gA	BOOL	Auto	Ungrouped_vars	No																																
2	gB	REAL	Auto		[0...4]																																
3	gC	REAL	%MD60.20	MyConstants_vars	No																																
4	gD	INT	Auto	MyMapped_vars	No																																
				Ungrouped_vars																																	

Step	Action																																													
5	Choose whether a variable is an array or not. If it is, edit the size of the variable: <table><thead><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th><th>Array</th><th>Init value</th></tr></thead><tbody><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td><td>No</td><td>TRUE</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td>Ungrouped_vars</td><td>[0...4]</td><td></td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyMapped_vars</td><td>No</td><td>1.0</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyConstants_vars</td><td>No</td><td>-74</td></tr></tbody></table> Size of variable Scalar Array / Matrix Dimensions 5 OK Cancel		Name	Type	Address	Group	Array	Init value	1	gA	BOOL	Auto	Ungrouped_vars	No	TRUE	2	gB	REAL	Auto	Ungrouped_vars	[0...4]		3	gC	REAL	%MD60.20	MyMapped_vars	No	1.0	4	gD	INT	Auto	MyConstants_vars	No	-74										
	Name	Type	Address	Group	Array	Init value																																								
1	gA	BOOL	Auto	Ungrouped_vars	No	TRUE																																								
2	gB	REAL	Auto	Ungrouped_vars	[0...4]																																									
3	gC	REAL	%MD60.20	MyMapped_vars	No	1.0																																								
4	gD	INT	Auto	MyConstants_vars	No	-74																																								
6	Edit the initial values of the variable: click the button in the corresponding cell and enter the values in the window that pops up: Init values for: gB (REAL) [0,1,2,1,0] OK Cancel																																													
7	Assign an attribute to the variable (for example, <code>CONSTANT</code> or <code>RETAIN</code>), by selecting it from the list which opens when you click the corresponding cell: <table><thead><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th><th>Array</th><th>Init value</th><th>Attribute</th></tr></thead><tbody><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td><td>No</td><td>TRUE</td><td>---</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td>Ungrouped_vars</td><td>[0...4]</td><td>[0,1,2,1,0]</td><td>---</td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyMapped_vars</td><td>No</td><td>1.0</td><td>CONSTANT</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyConstants_vars</td><td>No</td><td>-74</td><td>RETAIN</td></tr></tbody></table>		Name	Type	Address	Group	Array	Init value	Attribute	1	gA	BOOL	Auto	Ungrouped_vars	No	TRUE	---	2	gB	REAL	Auto	Ungrouped_vars	[0...4]	[0,1,2,1,0]	---	3	gC	REAL	%MD60.20	MyMapped_vars	No	1.0	CONSTANT	4	gD	INT	Auto	MyConstants_vars	No	-74	RETAIN					
	Name	Type	Address	Group	Array	Init value	Attribute																																							
1	gA	BOOL	Auto	Ungrouped_vars	No	TRUE	---																																							
2	gB	REAL	Auto	Ungrouped_vars	[0...4]	[0,1,2,1,0]	---																																							
3	gC	REAL	%MD60.20	MyMapped_vars	No	1.0	CONSTANT																																							
4	gD	INT	Auto	MyConstants_vars	No	-74	RETAIN																																							
8	Type a description for the variable in the corresponding cell. In the case of global variables, this operation may modify the position of the variable in the project tree: <table><thead><tr><th></th><th>Name</th><th>Type</th><th>Address</th><th>Group</th><th>Array</th><th>Init value</th><th>Attribute</th><th>Description</th></tr></thead><tbody><tr><td>1</td><td>gA</td><td>BOOL</td><td>Auto</td><td>Ungrouped_vars</td><td>No</td><td>TRUE</td><td>---</td><td>Global variable A</td></tr><tr><td>2</td><td>gB</td><td>REAL</td><td>Auto</td><td>Ungrouped_vars</td><td>[0...4]</td><td>[0,1,2,1,0]</td><td>---</td><td>Global variable B</td></tr><tr><td>3</td><td>gC</td><td>REAL</td><td>%MD60.20</td><td>MyMapped_vars</td><td>No</td><td>1.0</td><td>---</td><td>Global variable C</td></tr><tr><td>4</td><td>gD</td><td>INT</td><td>Auto</td><td>MyConstants_vars</td><td>No</td><td>-74</td><td>CONSTANT</td><td>Global variable D</td></tr></tbody></table>		Name	Type	Address	Group	Array	Init value	Attribute	Description	1	gA	BOOL	Auto	Ungrouped_vars	No	TRUE	---	Global variable A	2	gB	REAL	Auto	Ungrouped_vars	[0...4]	[0,1,2,1,0]	---	Global variable B	3	gC	REAL	%MD60.20	MyMapped_vars	No	1.0	---	Global variable C	4	gD	INT	Auto	MyConstants_vars	No	-74	CONSTANT	Global variable D
	Name	Type	Address	Group	Array	Init value	Attribute	Description																																						
1	gA	BOOL	Auto	Ungrouped_vars	No	TRUE	---	Global variable A																																						
2	gB	REAL	Auto	Ungrouped_vars	[0...4]	[0,1,2,1,0]	---	Global variable B																																						
3	gC	REAL	%MD60.20	MyMapped_vars	No	1.0	---	Global variable C																																						
4	gD	INT	Auto	MyConstants_vars	No	-74	CONSTANT	Global variable D																																						
9	Save the project to persist the changes you made to the declaration of the variable.																																													

Deleting Variables

Description

In order to delete one or more variables, select them in the editor. You can use the **Ctrl** or the **Shift** keys to select multiple elements:

	Name	Type	Address	Group	Array	Init value
1	G_I_iTe	INT	Auto	Ungrouped_vars	No	
2	G_I_iSe	INT	Auto	Ungrouped_vars	No	
3	G_I_iDi	INT	Auto	Ungrouped_vars	No	1
4	G_I_iC	BOOL	Auto	Ungrouped_vars	No	
5	G_I_iAl	BOOL	Auto	Ungrouped_vars	No	
6	gA	BOOL	Auto	Ungrouped_vars	No	TRUE
7	gB	REAL	Auto	Ungrouped_vars	[0...4]	[0,1,2,1,0]
8	gC	REAL	%MD60.20	MyMapped_vars	No	1.0
9	gD	INT	Auto	MyConstants_vars	No	-74

Delete selected variable(s), by applying one of the following operations:

- In the menu, click **Variables → Delete**.

- In the Project toolbar, click .
- Press the **Delete** key.

You cannot delete the **RESULT** of an IEC 61131-3 **FUNCTION**.

Sorting Variables

Description

You can sort the variables in the editor by clicking the column header of the field you want to use as the sorting criterion.

Copying Variables

Description

The variables editor allows you to copy and paste elements. You can either use keyboard shortcuts

or the **Edit → Copy** , **Edit → Paste**  menu.

NOTE: Overlapping addresses problems may occur by copying mapped variables.
PROGRAMMING can automatically assign available address to the pasted variable and fix the overlap. In order to enable this functionality, refer to Software Options (*see page 47*) and Merge Function (*see page 179*) for further details.

Chapter 11

Compiling

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	Overview	270
11.2	Compiling the Project	271
11.3	Compiler Output	273
11.4	Command-Line Compiler	276

Section 11.1

Overview

Overview

Description

Compilation consists of transforming the PLC source code into another programming language (the target language) such as binary code, which can be executed by the processor on the target device.

Section 11.2

Compiling the Project

What Is in This Section?

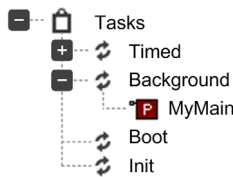
This section contains the following topics:

Topic	Page
Overview	271
Image File Loading	272

Overview

Prerequisite

Before starting the compilation, make sure that at least one program has been assigned to a task:



Otherwise, compilation aborts with a meaningful error message:



Compiling the Project

Start compiling the project, by applying one of the following operations:

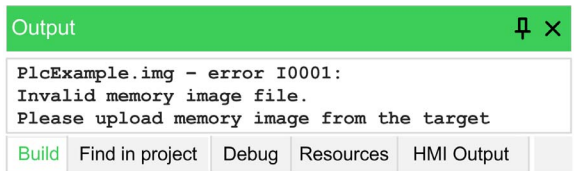
- In the menu, click  **Project → Compile**.
- In the Project toolbar, click .
- Press the **F7** key.

NOTE: PROGRAMMING automatically saves the changes to the project before starting the compilation.

Image File Loading

Description

Before performing the compilation, the compiler needs to load the image file (*.img file), which contains the memory map of the target device. If the target is connected when compilation is started, the compiler seeks the image file directly on the target. Otherwise, it loads the local copy of the image file from the working folder. If the target device is disconnected and there is no local copy of the image file, compilation cannot be carried out: you are then required to connect to a working target device.



Section 11.3

Compiler Output

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	273
Compiler Errors	274

Overview

Description

As the previous step is accomplished, the compiler performs the compilation, then displays a report in the **Output** window. The last string of the report has the following format:

```
m warnings, n errors
```

Condition	Description
$n > 0$	Compiler error(s). The PLC code contains one or more serious detected errors, which require intervention. No executable code was generated.
$n = 0, m > 0$	Emission of warning(s). The PLC code contains one or more minor detected errors, which the compiler automatically worked around. However, you are informed that the PLC program may act differently from what you intend. You should correct these minor errors by editing and recompiling the application until no other messages are reported.
$n = m = 0$	The compilation was successful in that no errors or warnings were detected.

Compiler Errors

Description

When your application contains one or more errors, information is displayed in the **Output** window for each of those detected errors.

Output

```
Preprocessing Global shared completed.

0 warnings, 0 errors.

Preprocessing user defined data .. completed.
Compiling programs .. completed.
Compiling function blocks .. completed.
Compiling functions .. completed.
Preprocessing user defined data .. completed.

Code generation ..
Preprocessing EmbeddedElements completed.
aborted.
MAIN(1) - error A4097: N1 => Object not found

0 warnings, 1 errors.
```

Build

Find in project

Debug

Resources

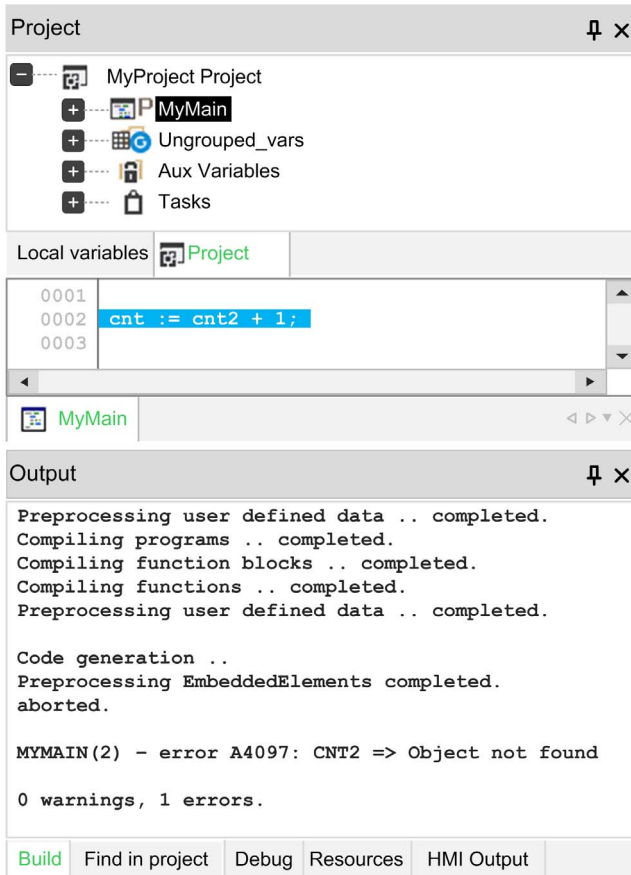
HMI Output

For each detected error, the information includes:

- The name of the Program Organization Unit affected by the error;
- The number of the source code line which procured the error;
- The type of error:
 - error: serious error
 - warning: minor error
- The error code;
- The error description.

For more information, refer to Compile Time Error Messages ([see page 446](#)).

If you double-click the error message in the **Output** window, **PROGRAMMING** opens the source code and highlights the line containing the detected error:



Section 11.4

Command-Line Compiler

Overview

Description

The compiler can be used independently from the integrated software environment: in the directory of EcoStruxure Machine Expert - HVAC, you can find an executable file, **EWc.exe**, which can be invoked (for example, in a batch file) with a number of options.

In order to get information about the syntax and the options of this command-line tool, launch the executable without parameters.

Chapter 12

Launching the Application

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
12.1	Overview	278
12.2	Setting Up the Communication	279
12.3	On-Line Status	287
12.4	Downloading the Application	289
12.5	Simulation	290
12.6	Control the PLC Execution	291

Section 12.1

Overview

Overview

Description

In order to download and debug the application, you have to establish a connection with the target device. This chapter focuses on the operations required to connect to the target and to download the application. A separate chapter is dedicated to Debugging ([see page 293](#)).

Section 12.2

Setting Up the Communication

What Is in This Section?

This section contains the following topics:

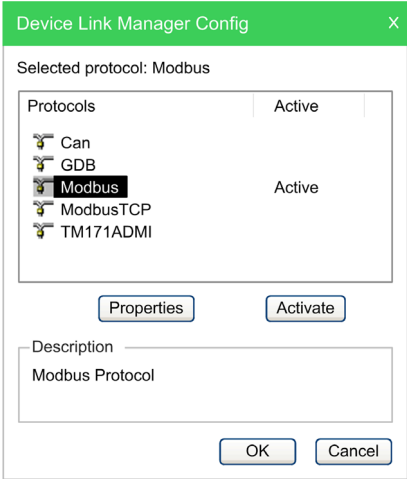
Topic	Page
Overview	280
Saving the Last Used Communication Port	286

Overview

Description

In order to establish the connection with the target device, verify all connections and network communication.

Follow this procedure to set up and establish the connection to the target device:

Step	Action
1	Click On-line → Set up communication... menu of the PROGRAMMING tab. This causes the following dialog box to appear. 
2	Select the appropriate protocol: • CAN (see page 281) • GDB (see page 281) • Modbus (see page 281) • ModbusTCP (see page 282) • TM171ADMI (see page 284)
3	Click Activate button to activate the protocol.
4	Click Properties button to modify the properties of the activated protocol.
5	Fill in all the protocol-specific settings. For example: the address or the communication timeout (that is how long PROGRAMMING must wait for an answer from the target before displaying a communication error message).
6	Click OK button to apply the changes you made to the communication settings.

Now you can establish communication by clicking  **On-line → Connect** menu.

CAN

Select **CAN** in the case of CAN connection:

Parameter	Description	Values	Default	Note
Baud_CAN_OB	CAN protocol baud rate On-board	2 = 500 kbaud 3 = 250 kbaud 4 = 125 kbaud 5 = 125 kbaud 6 = 50 kbaud	2	-
Addr_CAN_OB	On-board CAN serial address	1...127	1	The address is determined by the sum of this value the value of the DIP switches

GDB

The GDB protocol is not implemented in the M171/M172 controllers.

The GDB protocol is reserved for internal software use.

Modbus

Select **Modbus** in the case of USB/RS485 connection:

Parameter	Description	Values	Default	Note
Baud_RS485_OB	Modbus protocol baud rate On-board	0 = 9600 baud 1 = 19200 baud 2 = 38400 baud 3 = 57600 baud 4 = 76800 baud 5 = 115200 baud	2	-
Addr_RS485_OB	On-board RS485 serial address	1...255	1	The address is determined by the sum of this value the value of the DIP switches
Proto_RS485_OB	On-board RS485 protocol selection	2 = uNET 3 = Modbus/RTU 4 = BACnet MS/TP	3	-
Databit_RS485_OB	RS485 data bit number On-board	8	8	Fixed at 8
Stopbit_RS485_OB	On-board RS485 stop bit number	1 = 1 stop bit 2 = 2 stop bit	1	-
Parity_RS485_OB	On-board RS485 protocol parity	0 = NULL 1 = ODD 2 = EVEN	2	-

Modbus TCP

Select the **ModbusTCP** protocol in the case of Ethernet connection, using the relevant communication module if necessary.

In the protocol properties window:

- The **IP or hostname** box is for entering either an IP address (the default setting for M171P/M172P is 10.0.0.100) or a host name on a local network.
- The TCP/IP communication **Port** box is set by default to 502.

Connect the PC Ethernet cable to M171P/M172P.

Configure the TCP/IPv4 connection in the Ethernet port properties of your PC with the address (10.0.0.101):

Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

☐ Obtain an IP address automatically

☒ Use the following IP address:

IP address: 10 .000 .000 .101

Subnet mask: 255 .255 .255 .255

Default gateway: . . .

☐ Obtain DNS server address automatically

☒ Use the following DNS server addresses:

Preferred DNS server: . . .

Alternate DNS server: . . .

☐ Validate settings upon exit

Advanced...

OK Cancel

NOTE: The default M171P/M172P configuration is 10.0.0.100: the PC Ethernet port is thus configured with an address different to the default address (for example 10.0.0.101, the first three fields must be the same, the fourth different).

Click the **OK** button: the PC is configured to dialog with M171P/M172P via the Ethernet port.

M171P/M172P has a number of BIOS parameters for managing the connection between the target and EcoStruxure Machine Expert - HVAC but, unlike M171O, it does not have a default menu displayed on the on-board or remote display.

Passive Ethernet Plugin:

The Ethernet passive plug-in configuration parameters involve the configuration of the TCP/IP communication port (for example 502), the IP address, the gateway, and the subnet mask.

The “Default Gateway” parameters are not relevant in the local point-to-point network.

For connections via a router the “Default Gateway”, parameters must be set according to the network configuration, as in the following example:

Parameter	Description	Value	Parameter	Description	Value
Ip_1_ETH_PI	Ethernet passive Plug-in IP address (first part)	192	DefGtwy_1_ETH_PI	Default Gateway (first part)	192
Ip_2_ETH_PI	Ethernet passive Plug-in IP address (second part)	168	DefGtwy_2_ETH_PI	Default Gateway (second part)	168
Ip_3_ETH_PI	Ethernet passive Plug-in IP address (third part)	0	DefGtwy_3_ETH_PI	Default Gateway (third part)	0
Ip_4_ETH_PI	Ethernet passive Plug-in IP address (fourth part)	100	DefGtwy_4_ETH_PI	Default Gateway (fourth part)	1

M171P Flush Mounting **Specific HMI Management:**

In addition to the BIOS parameters, M171P Flush Mounting manages the HMI menu:

Parameter	Description	Values	Default	Note
Hmi_language	Display language (local or remote)	0...65535	0	-
HMIList_current	Current HMI	0 = Remote HMI 1 / 1 = Remote HMI 2 2 = Remote HMI 3 / 3 = Remote HMI 4 4 = Remote HMI 5 / 5 = Remote HMI 6 6 = Remote HMI 7 / 7 = Remote HMI 8 8 = Remote HMI 9 / 9 = Remote HMI 10 10 = not used 11 = Local HMI	11	Local HMI is identified on the display as network In Connection as HMI Remote HMI is identified In Connection as Remote HMI

Ten remote menus are available. The first menu parameters are listed below. The others are similar:

Parameter	Description	Values	Default	Note
HmiList_ID_1	Remote HMI 1 navigation ID list	0...254	0	-
HmiList_Res_1	Remote HMI 1 navigation resource type	1 = RTU (RS485 Modbus RTU) 2 = TCP (Modbus TCP) 3 = CAN (CAN)	3 = CAN	-
HmiList_Addr_1	Remote HMI 1 navigation resource address for CAN, RTU, and TCP (IP part 1)	0...255	0	For example: CAN: 2.500000 RS485: 1.38400.P81 Modbus TCP: 010.000.000.100
HmiList_Addr_2	Remote HMI 1 navigation resource address for TCP (IP part 2)	0...255	0	
HmiList_Addr_3	Remote HMI 1 navigation resource address for TCP (IP part 3)	0...255	0	
HmiList_Addr_4	Remote HMI 1 navigation resource address for TCP (IP part 4)	0...255	0	
HmiList_File_1	Remote HMI navigation file 1 (DOS 8.3 uppercase format)	Alphanumeric string, 8 characters	*****	The default name is HMIREM.KBD

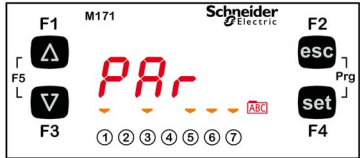
TM171ADMI

Select TM171ADMI in the case of connection with the M171O with the TM171ADMI programming cable.

M171O has parameters in the **CF** folder in the controller for managing the connection between the target and EcoStruxure Machine Expert - HVAC.

If the target is "empty", for example there is no controller application on the device, M171O displays the message **F r E E**. Otherwise, a controller application exists on M171O and the message PLC appears on the display. Simultaneously press the **UP** and **DOWN** keys to view the message.

Follow this procedure to modify a parameter:

Step	Action	Result
1	From the main display, press the set key and the esc key simultaneously to open the programming menu:	The programming menu is opened. The label of the first subfolder is displayed (PAR in this case):
		
2	Press set key to open the Parameters menu.	The label of the first subfolder is displayed (CL).
3	Press the UP and DOWN keys to scroll the other labels until you find the one indicated by CF .	-
4	Press set key to open the folder.	The label of the first parameter is displayed.
5	Press the UP and DOWN keys to scroll through the various parameters until you find the connection parameters.	-
6	Press set key to view the value of the parameter.	The value of the parameter is displayed.
7	Press the UP and DOWN keys to modify this value.	-
8	Press set key to validate the new value of the parameter. NOTE: Press esc to take you back to the previous folder without saving the value entered.	-
9	Press esc key to go back to the main display.	-
10	Use the UP and DOWN keys to scroll the other parameters and repeat the procedure (step 5 to 9) to view the values and - if necessary - edit them.	-

The parameters values needed for correct connection between the M171O target and EcoStruxure Machine Expert - HVAC:

Parameter	Description	Values	Default	Parameter Visibility Level	Note
CF01 ⁽¹⁾	Select COM1 (TTL) protocol	0 = reserved 1 = Modbus	1	2	Must be set to 1
CF30	Modbus protocol controller address	1...255	1	3	Check that the set values correspond to those defined by the tab On-line → Set up → Communication → Properties
CF31 ⁽²⁾	Modbus protocol baud rate	0, 1, 2 = not used 3 = 9600 baud 4 = 19200 baud 5 = 38400 baud 6 = 57600 baud 7 = 115200 baud	3	3	
CF32	Modbus protocol controller parity	1 = EVEN 2 = NONE 3 = ODD	1	3	
(1) COM1 = The TTL port and the RS485 port cannot be used simultaneously.					
(2) CF31	5 = 38400 baud (RS485: not supported) 6 = 57600 baud (RS485: not supported) 7 = 115200 baud (RS485: not supported)				

For other parameters and to manage parameter visibility levels, refer to *Modicon M171 Optimized Logic Controller Hardware Guide*.

Saving the Last Used Communication Port

Description

When you connect to target devices using a serial port (COM port), you usually use the same port for all devices (many PCs have only one COM port). You may save the last used COM port and let **PROGRAMMING** use that port to override the project settings: this feature is useful when you share projects with other developers, which may use a different COM port to connect to the target device.

In order to save your COM port settings, enable the **Use last port** option in **File → Options... → General** tab.

Section 12.3

On-Line Status

What Is in This Section?

This section contains the following topics:

Topic	Page
Connection Status	287
Project Status	288

Connection Status

Description

The state of communication is displayed in a box next to the right border of the **Status** bar.

If you have not yet attempted to connect to the target, the state of communication is set to **Not connected**.

NOT CONNECTED

When you connect to the target device, the state of communication becomes one of the following:

- ERROR

Error: the communication cannot be established. You should verify both the physical link and the communication settings (*see page 279*).

- CONNECTED

Connected: the communication has been established.

Project Status

Description

Next to the communication status there is another box which indicates the status of the project currently executing on the target device.

When the connection status is `Connected`, the project status takes on one of the following values (depending on source code):

- **NO CODE**

`No code`: no project is executing on the target device.

- **DIFF. CODE**

`Diff. code`: the project currently executing on the target device differs from the one currently open in the software; moreover, no debug information consistent with the running project is available: thus, the values displayed in the watch window or in the oscilloscope are not reliable and the debug mode cannot be activated.

- **DIFF. CODE (SYM)**

`Diff. code, Symbols OK`: the project currently executing on the target device is not the same as the one currently open in the software; however, some debug information consistent with the running project is available (for example, the project has been previously downloaded to the target device from the same PC): the values displayed in the watch window or in the oscilloscope are reliable, but the debug mode still cannot be activated.

- **SOURCE OK**

`Source OK`: the project currently executing on the target device is the same as the one currently open in the software: the debug mode can be activated.

Section 12.4

Downloading the Application

Overview

Description

A compiled PLC application must be downloaded to the target device in order to have the processor execute it. The steps presents how to download an application code into a target device.

Step	Action
1	Connect the target device to the PC where EcoStruxure Machine Expert - HVAC is running.
2	Click  On-line → Download code. Result: PROGRAMMING verifies whether the project has unsaved changes. If so, it automatically starts the compilation of the application. The binary code is sent to the target device. After the end of the download, the controller automatically resets. Then the downloaded code is executed by the processor on the target device.

WARNING

AUTOMATIC RESTART OF CONTROLLER

- Do not download your application without first accessing the state of your machine or process.
- Do not download your application without first ascertaining that there is no risk of injury to anyone in or around your machine or process.

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

Section 12.5

Simulation

Overview

Description

Depending on the target device you are interfacing with, you may be able to simulate the execution of the PLC application with integrated simulation environment of **PROGRAMMING: Simulation**.

In order to start the simulation, click  **Debug → Simulation mode**.

Refer to the *Simulation manual* for instructions on the simulator.

Section 12.6

Control the PLC Execution

Control the PLC Execution

Overview

The PLC application execution can be controlled using the related functions.

These functions are accessible in the Project toolbar (*see page 156*) and in the On-line menu (*see page 151*).

The controller will start executing program logic when power is applied to the equipment. It is essential to know in advance how the outputs will affect the process or machine being controlled.

At start up, the controller will attempt to start executing program logic when power is applied to the equipment, regardless of the reason the controller had previously stopped. It is essential to know in advance how an unconditional start will affect the process or machine being controlled.

WARNING

UNINTENDED MACHINE START-UP

- Conduct a thorough risk analysis to determine the effects, under all conditions, of an unconditional start of the application.
- Verify the state of security of your machine or process environment before applying power to the controller or starting the application with EcoStruxure Machine Expert - HVAC software.

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

WARNING

UNINTENDED EQUIPMENT OPERATION

- Never assume that your controller is in a certain controller state before commanding a change of state, configuring your controller options, uploading a program, or modifying the physical configuration of the controller and its connected equipment.
- Before performing any of these operations, consider the effect on all connected equipment.
- Before acting on a controller, always positively confirm the controller state, checking for the presence of output forcing, and reviewing the controller status information via EcoStruxure Machine Expert - HVAC.

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

Halt

You can stop the PLC execution by clicking  **On-line → Halt.**

Cold Restart

The PLC application execution will be restarted and both retain and non-retain variables will be reset.

You can cold restart the PLC execution by clicking  **On-line → Cold restart.**

Warm Restart

The PLC application execution will be restarted and only non-retain variables will be reset.

You can warm restart the PLC execution by clicking  **On-line → Warm restart.**

Hot Restart

The PLC application execution will be restarted and no variables will be reset.

You can hot restart the PLC execution by clicking  **On-line → Hot restart.**

Reboot Target

You can reboot the target by clicking  **On-line → Reboot target.**

Chapter 13

Debugging

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
13.1	Overview	294
13.2	Watch Window	295
13.3	Oscilloscope	303
13.4	Edit and Debug Mode	321
13.5	Live Debug	322
13.6	Triggers	326
13.7	Graphic Triggers	347

Section 13.1

Overview

Overview

Description

PROGRAMMING provides several debugging tools, which help you to verify whether the application behaves as intended.

All these debugging tools basically allow you to watch the value of selected variables while the PLC application is running.

PROGRAMMING debugging tools can be gathered in two classes:

- Asynchronous debuggers. They read the values of the variables you selected with successive queries issued to the target device. Both the manager of the debugging tool (that runs on the PC) and, potentially, the task which is responsible to answer those queries (on the target device) run independently from the PLC application. The values of two distinct variables being sampled in the same moment are not necessarily in concordance with each other with respect to the PLC application execution (one or more cycles may have occurred). For the same reason, the evolution of the value of a single variable is not consistent with its actual value in the controller memory, especially when it is updated rapidly within the application.
- Synchronous debuggers. They require the definition of a trigger in the PLC code. They refresh simultaneously all the variables they have been assigned every time the processor reaches the trigger, as no further instruction can be executed until the value of all the variables is refreshed. As a result, synchronous debuggers obviate the limitations affecting asynchronous ones.

This chapter presents how to debug your application using both asynchronous and synchronous tools.

Section 13.2

Watch Window

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	295
Opening and Closing the Watch Window	296
Adding Items to the Watch Window	296
Removing a Variable	299
Refreshment of Values	299
Changing the Format of Data	300
Working with Watch Lists	301
Autosave Watch List	302

Overview

Description

The **Watch** window allows you to monitor the values of a set of variables. Being an asynchronous tool, the **Watch** window does not establish synchronization of values. Values of the variables in the **Watch** window may refer to different execution cycles of the corresponding task.

The **Watch** window contains an item for each variable that you added to it. The information displayed in the **Watch** window includes the name of the variable, its value, its type, and its location in the PLC application.

Watch			
			
Symbol	Value	Type	Location
 HMIPIDTEST	FALSE	BOOL	global
 HMIPIDTHRESHOLD	0.2	REAL	global
 PARCTDOWNPRESET	100	INT	global
 BASETIME	0	UDINT	@FAST:PIDMODESELECTOR

Opening and Closing the Watch Window

Description

Open/close the **Watch** window, by applying one of the following operations:

- In the menu, click **View** → **Tool windows** → **Watch**.
- In the Main toolbar, click .
- Press the **Ctrl+T** key.

Closing the **Watch** window means simply hiding it, not resetting it. If you close the **Watch** window and then open it again, you will see that it still contains all the variables you added to it.

Adding Items to the Watch Window

Description

To display a variable in the **Watch** window, you need to add it to the watch list.

NOTE: All the variables of the project can be added to the **Watch** window, regardless of where they were declared.

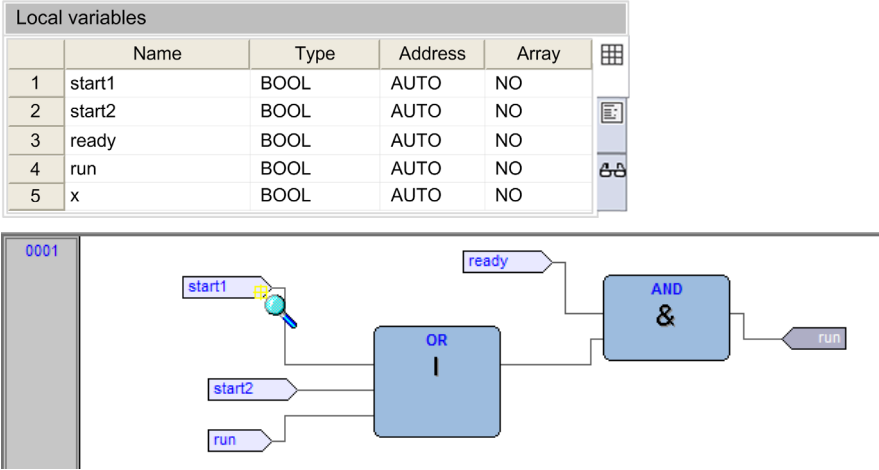
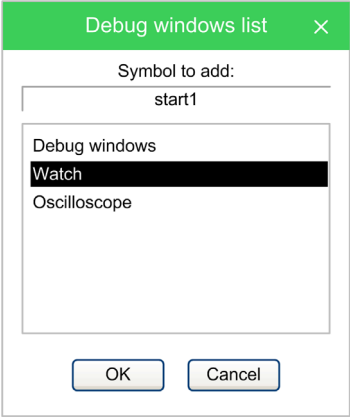
Adding a Variable from a Textual Source Code Editor

To add a variable to the **Watch** window from a textual (IL or ST) source code editor:

Step	Action								
1	Double-click the variable you wish to display in the Watch window. <pre>0001 x := x + hmiFrequency; 0002 0003 hmiSinVal := SIN(x) * hmiAmplitude; 0004 hmiCosVal := COS(x) * hmiAmplitude; 0005 0006 hmiStep := hmiStep + 1; 0007 0008</pre>								
2	Drag the selected variable in the Watch window: Watch           <table><tr><th>Symbol</th><th>Value</th><th>Type</th><th>Location</th></tr><tr><td>▲ HMISINVAL</td><td>0.982414</td><td>REAL</td><td>@FAST:LOOPS</td></tr></table>	Symbol	Value	Type	Location	▲ HMISINVAL	0.982414	REAL	@FAST:LOOPS
Symbol	Value	Type	Location						
▲ HMISINVAL	0.982414	REAL	@FAST:LOOPS						

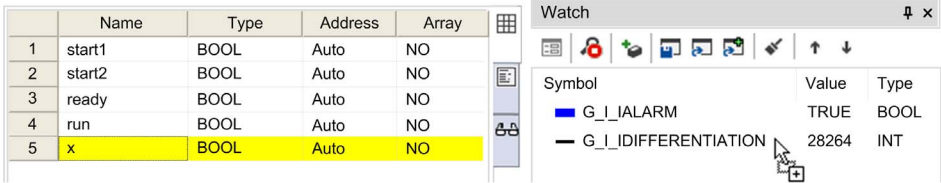
Adding a Variable from a Graphical Source Code Editor

To add a variable to the **Watch** window from a graphical (LD, FBD, or SFC) source code editor:

Step	Action
1	Click  Edit → Watch mode.
2	Click the variable to display in the Watch window. 
3	A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked.  To add the variable in the Watch window, select Watch and click OK.
4	Once you have added to the Watch window all the variables you want to display, click  Edit → Insert/Move mode, the mouse cursor turns to its original shape.

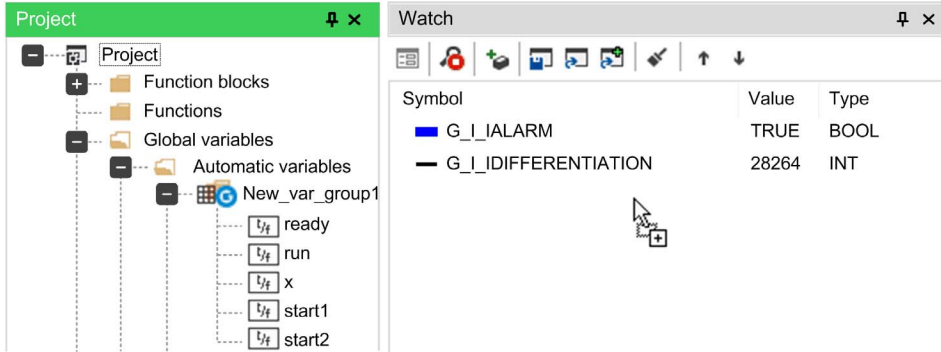
Adding a Variable from a Variables Editor

To add a variable to the **Watch** window, select the corresponding record in the variables editor and drag it in the **Watch** window:



Adding a Variable from the Project Tree

To add a variable to the **Watch** window, select it in the project tree and drag it in the **Watch** window:



Adding a Variable from the Watch Window Toolbar

To add a variable to the **Watch** window, click  **Insert new item** in the **Watch** window toolbar.

You shall type (or select by browsing the project symbols) the name of the variable and its location (where it has been declared).

Add item to watch window

Symbol name

Browse

Address

Location

Browse

OK

Cancel

Removing a Variable

Description

If you want to remove a variable from the **Watch** window, select it by clicking its name once, then press the **Delete** key.

Refreshment of Values

Normal Operation

The watch window manager reads periodically from memory the value of the variables.

Watch ⌵ ✕			
📄 🔍 ➕ 📁 🔄 🔧 ⬆ ⬇			
Symbol	Value	Type	Location
▲ HMISINVAL	0.982414	REAL	@FAST:LOOPS

However, this action is carried out asynchronously. It may happen that a higher-priority task modifies the value of some of the variables while they are being read. Thus, at the end of a refreshment process, the values displayed in the window may refer to different execution states of the PLC code.

Target Disconnected

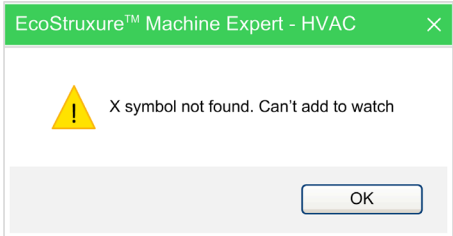
If the target device is disconnected, the **Value** column contains three dots.

Watch ⌵ ✕			
📄 🔍 ➕ 📁 🔄 🔧 ⬆ ⬇			
Symbol	Value	Type	Location
HMISINVAL	...	REAL	@FAST:LOOPS

Object Not Found

If the PLC code changes and **PROGRAMMING** cannot retrieve the memory location of an object in the **Watch** window, then the **Value** column contains three dots.

If you try to add to the **Watch** window a symbol which has not been allocated, **PROGRAMMING** gives the following error message:



Changing the Format of Data

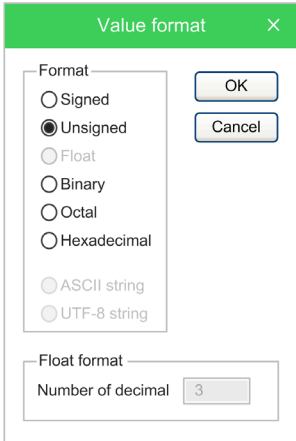
Description

When you add a variable to the **Watch** window, **PROGRAMMING** automatically recognizes its type (unsigned integer, signed integer, floating point, hexadecimal), and displays its value consistently. Also, if the variable is floating point, **PROGRAMMING** assigns it a default number of decimal figures.

However, you may need the variable to be displayed in a different format.

To impose another format than the one assigned by **PROGRAMMING**, select the variable and click the  **Value format** button in the toolbar.

Choose the format and confirm your choice.



Working with Watch Lists

Description

You can store to file the set of all the items in the **Watch** window in order to easily restore the status of this debugging tools in a successive working session.

Follow this procedure to save a watch list:

Step	Action
1	Click the  Save watch list button in the Watch window toolbar.
2	Enter the file name and choose its destination in the file system.

You can load a watch list from file, removing the opened one, following this procedure:

Step	Action
1	Click the  Load (no appends) watch list button in the Watch window toolbar.
2	Browse the file system and select the watch list file. The set of symbols in the watch list is added to the Watch window.

You can load a watch list from file, appending to the opened one, following this procedure:

Step	Action
1	Click the  Load watch list button in the Watch window toolbar.
2	Browse the file system and select the watch list file. The set of symbols in the watch list is added to the Watch window.

You can clear the current opened watch list by clicking the  **Remove all items from the watch list** button.

You can modify the place of a selected item in the current open watch list by clicking the  **Move up item into the list** button or the  **Move down item into the list** button.

Autosave Watch List

Description

By selecting the associated option in the project options dialog (refer to Debug ([see page 170](#)) for more information), the watch list is automatically saved on the project closing.

The saved watch list is automatically loaded on the first connection to target when the project is reopened.

Section 13.3

Oscilloscope

What Is in This Section?

This section contains the following topics:

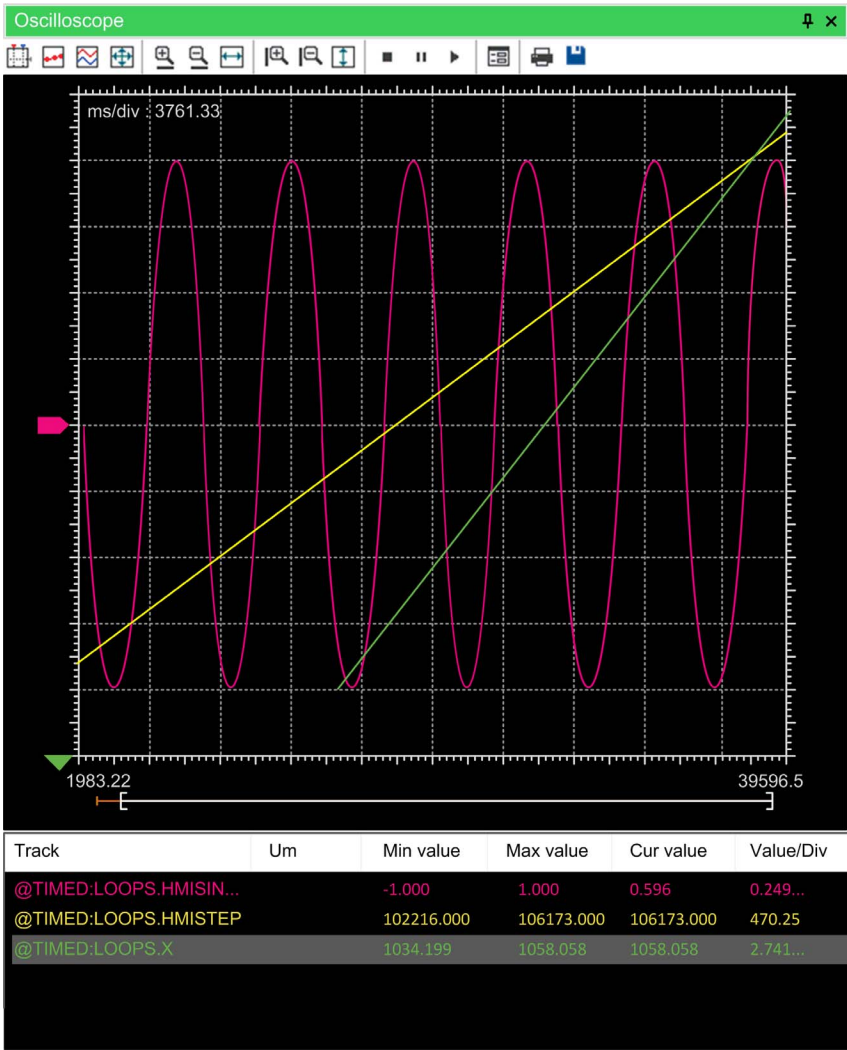
Topic	Page
Overview	304
Opening and Closing the Oscilloscope	305
Adding Items to the Oscilloscope	305
Removing a Variable	309
Variables Sampling	309
Controlling Data Acquisition and Display	310
Changing the Polling Rate	318
Saving and Printing the Graph	318

Overview

Description

The Oscilloscope allows you to plot the evolution of the values of a set of variables. Being an asynchronous tool, the Oscilloscope cannot establish synchronization of samples.

Oscilloscope window is an interface for accessing the debugging functions that the Oscilloscope makes available:



The **Oscilloscope** consists of three elements:

- The toolbar allows you to control the Oscilloscope. A detailed description of the function of each control is given later in this chapter.
- The Chart area includes several items:
 - Plot: area containing the curve of the variables.
 - Vertical cursors: cursors identifying two distinct vertical lines. The values of each variable at the intersection with these lines are reported in the corresponding columns.
 - Scroll bar: if the scale of the x-axis is too large to display all the samples in the Plot area, the scroll bar allows you to slide back and forth along the horizontal axis.
- The lower section of the Oscilloscope is a table consisting of a row for each variable.

Opening and Closing the Oscilloscope

Description

To open, close the Oscilloscope, click  **View → Tool windows → Oscilloscope**.

Closing the Oscilloscope means simply hiding it, not resetting it. If you open again the Oscilloscope after closing it, you will see that plotting of the curve of all the variables you added to it starts again.

Adding Items to the Oscilloscope

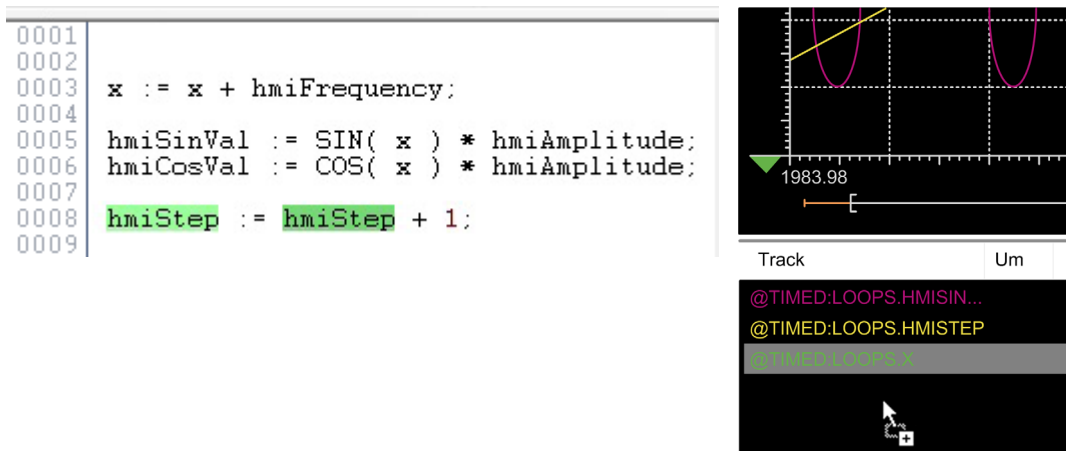
Description

In order to plot the evolution of the value of a variable, you need to add it to the Oscilloscope.

Unlike trigger windows and the **Graphic trigger** window, you can add to the Oscilloscope all the variables of the project, regardless of where they were declared.

Adding a Variable from a Textual Source Code Editor

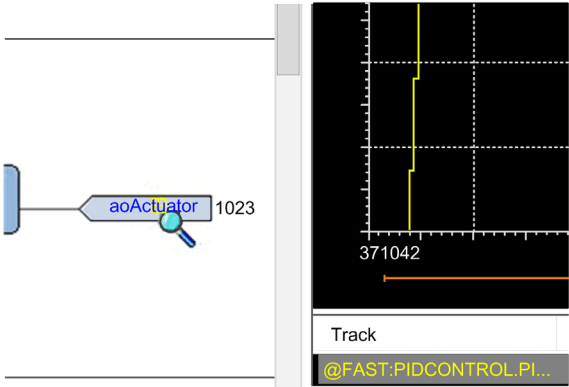
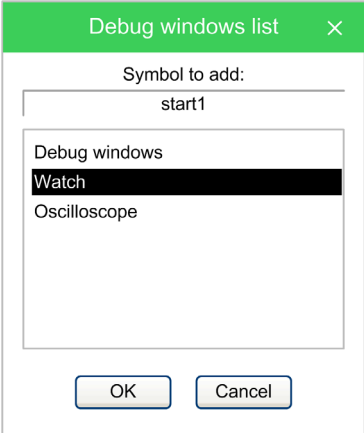
To add a variable to the Oscilloscope from a textual (that is, IL or ST) source code editor, select a variable by double-clicking it, and then drag it into the **Oscilloscope** window.



The same procedure applies to all the variables you wish to monitor.

Adding a Variable from a Graphical Source Code Editor

Follow this procedure to add a variable to the Oscilloscope from a graphical (that is, LD, FBD, or SFC) source code editor:

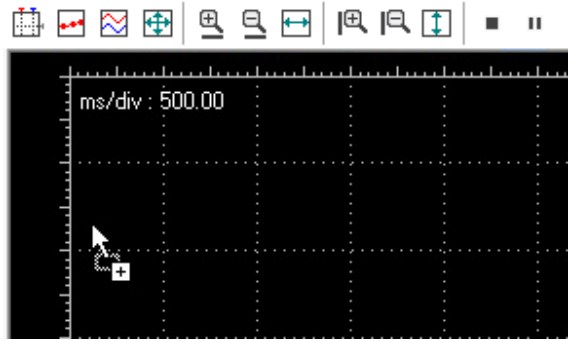
Step	Action
1	Click  Edit → Watch mode.
2	Click the block representing the variable you wish to trace in the Oscilloscope: 
3	A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked:  Select Oscilloscope, then click OK. The name of the variable is now displayed in the Track column.

The same procedure applies to all the variables you wish to monitor.
Once you have added to the Oscilloscope all the variables you want to observe, you should click  **Edit → Insert/Move mode**: the mouse cursor turns to its original shape.

Adding a Variable from a Variables Editor

In order to add a variable to the Oscilloscope, you can select the corresponding record in the variables editor and then either drag-and-drop it in the Oscilloscope:

Local variables							
	Class	Pin	Name	Type	Array	Init value	Attribute
1	VAR		absSpeed	REAL	NO		..
2	VAR		T	REAL	NO		..
3	VAR		remSpace	DINT	NO		..
4	VAR		T2	REAL	NO		..
5	VAR		sign	REAL	NO		..
6	VAR		prevSpeed	REAL	NO		..



or press the **F10** key and choose **Oscilloscope** from the list of debug windows which pops up.

Adding a Variable from the Project Tree

In order to add a variable to the Oscilloscope, you can select it in the project tree and then either drag-and-drop it in the Oscilloscope:



or press the **F10** key and choose **Oscilloscope** from the list of debug windows which pops up.

Removing a Variable

Description

If you want to remove a variable from the Oscilloscope, select it by clicking its name once, then press the **Delete** key.

Variables Sampling

Normal Operation

The Oscilloscope manager periodically reads from memory the value of the variables.

However, this action is carried out asynchronously. It may happen that a higher-priority task modifies the value of some of the variables while they are being read. At the end of a sampling process, data associated with the same value of the x-axis may refer to different execution states of the PLC code.

Target Disconnected

If the target device is disconnected, the curves of the dragged-in variables are frozen until communication is restored.

Controlling Data Acquisition and Display

Description

The Oscilloscope includes a toolbar with several commands, which can be used to control the acquisition process and the way data are displayed. This paragraph focuses on these commands.

The commands in the toolbar are disabled if no variable has been added to the Oscilloscope.

Starting and Stopping Data Acquisition

When you add a variable to the Oscilloscope, data acquisition begins immediately.

However, you can suspend the acquisition by clicking  **Pause acquisition**.

The curve freezes (while the process of data acquisition is still running in the background) until you click  **Restart acquisition**.

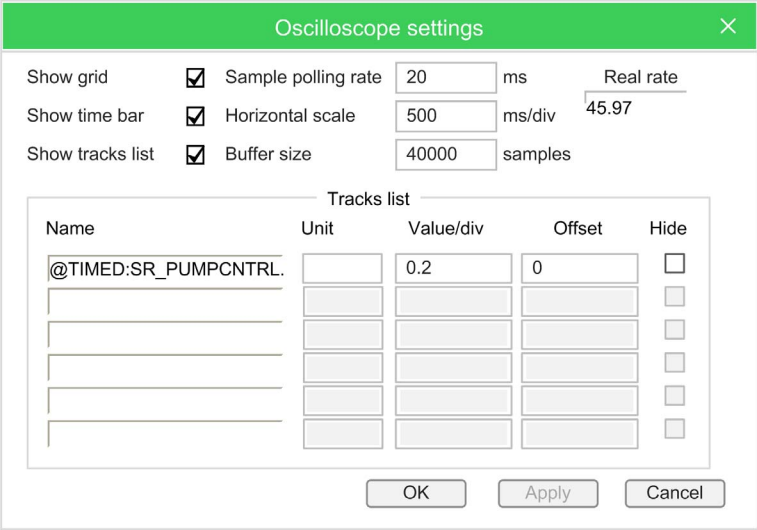
In order to stop the acquisition, you may click  **Stop acquisition**.

In this case, when you click  **Restart acquisition**, the plot is reset and restarts with the actual value of the variable.

Setting the Scale of the Axes

When you open the Oscilloscope, **PROGRAMMING** applies a default scale to the axes.

Follow this procedure to modify the scale value:

Step	Action
1	Open the Oscilloscope settings by clicking the  Graph properties button in the toolbar.
2	Set the scale of the horizontal axis, which is common to all the tracks: 

Step	Action																																			
3	For each variable, you may specify a distinct scale for the vertical axis: Oscilloscope settings Show grid ☒ Sample polling rate 20 ms Real rate 45.97 Show time bar ☒ Horizontal scale 500 ms/div Show tracks list ☒ Buffer size 40000 samples Tracks list <table><tr><th>Name</th><th>Unit</th><th>Value/div</th><th>Offset</th><th>Hide</th></tr><tr><td>@TIMED:SR_PUMPCNTRL.</td><td></td><td>100</td><td>0</td><td>☐</td></tr><tr><td></td><td></td><td></td><td></td><td>☐</td></tr><tr><td></td><td></td><td></td><td></td><td>☐</td></tr><tr><td></td><td></td><td></td><td></td><td>☐</td></tr><tr><td></td><td></td><td></td><td></td><td>☐</td></tr><tr><td></td><td></td><td></td><td></td><td>☐</td></tr></table> OK Apply Cancel	Name	Unit	Value/div	Offset	Hide	@TIMED:SR_PUMPCNTRL.		100	0	☐					☐					☐					☐					☐					☐
Name	Unit	Value/div	Offset	Hide																																
@TIMED:SR_PUMPCNTRL.		100	0	☐																																
				☐																																
				☐																																
				☐																																
				☐																																
				☐																																
4	Confirm your settings. The graph adapts to reflect the new scale.																																			

You can also zoom in and out with respect to both the horizontal and the vertical axes:



Finally, you may also adapt the scale of the horizontal axis, the vertical axis, or both to include all the samples, by clicking the corresponding item of the toolbar:



Vertical Split

When you are watching the evolution of two or more variables, you may want to split the respective tracks. For this purpose, click the  **Vertical split** item in the **Oscilloscope** toolbar:



Viewing Samples

If you click the  **Show samples** item in the **Oscilloscope** toolbar, the tool highlights the single values detected during data acquisition:

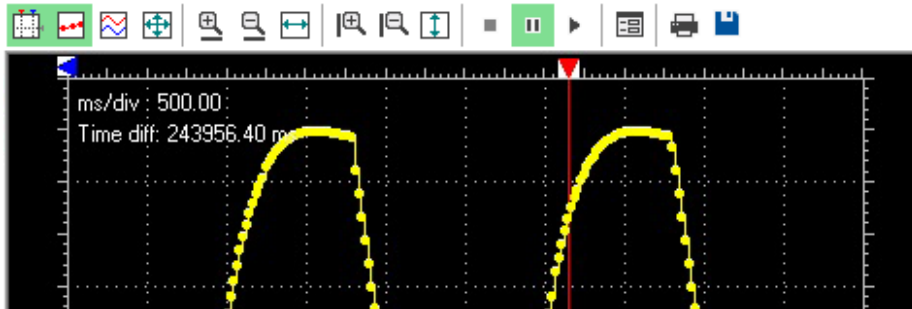


You can click the same item again in order to go back to the default view mode.

Taking Measures

The Oscilloscope includes two measure bars, which can be exploited to take some measures on the chart. In order to show and hide them, click the  **Show measure bars** item in the **Oscilloscope** toolbar.

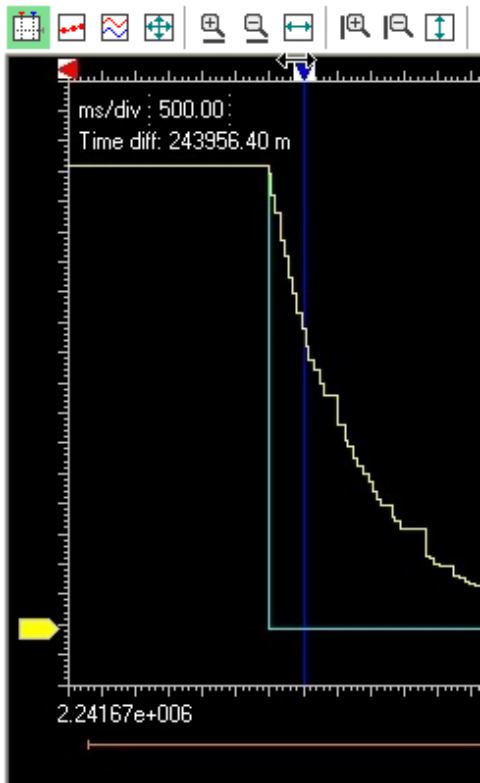
If you want to measure a time interval between two events, you have to move one bar to the point in the graph that corresponds to the first event and the other to the point that corresponds to the second one:



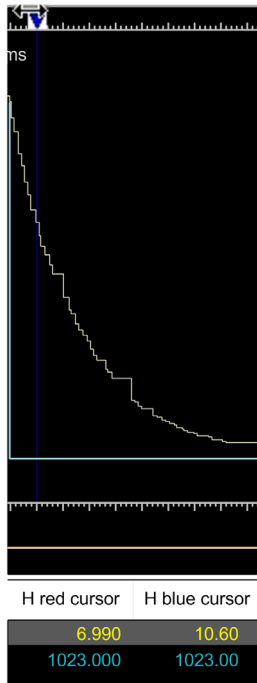
The time interval between the two bars is displayed in the top left corner of the chart:



You can use a measure bar also to read the value of all the variables in the Oscilloscope at a particular moment: move the bar to the point in the graph which corresponds to the instant you want to observe:



In the following table (below the graphic), you can now read the values of all the variables at that particular moment:



Oscilloscope Settings

You can further customize the appearance of the Oscilloscope by clicking the  **Graph properties** item in the toolbar.

In the window that pops up you can choose whether to display or not the **Background grid**, the **Time slide bar**, and the **Track list**:

Oscilloscope settings

Show grid

☐

Show time bar

☒

Show tracks list

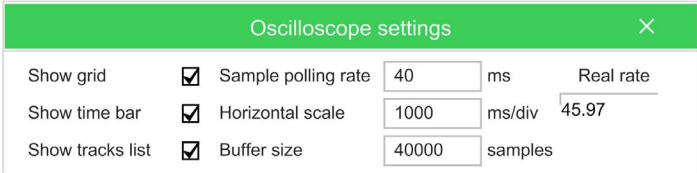
☒

Changing the Polling Rate

Description

PROGRAMMING periodically sends queries to the target device in order to read the data to be plotted in the Oscilloscope.

Follow this procedure to configure the polling rate:

Step	Action
1	Click the  Graph properties item in the toolbar.
2	In the window that pops up, edit the Sampling polling rate : 
3	Confirm your decision.

The rate depends on the performance of the target device (in particular, on the performance of its communication task). You can read the rate in the **Oscilloscope settings** window.

Saving and Printing the Graph

Description

PROGRAMMING allows you to persist the acquisition either by saving the data to a file or by printing a view of the data plotted in the Oscilloscope.

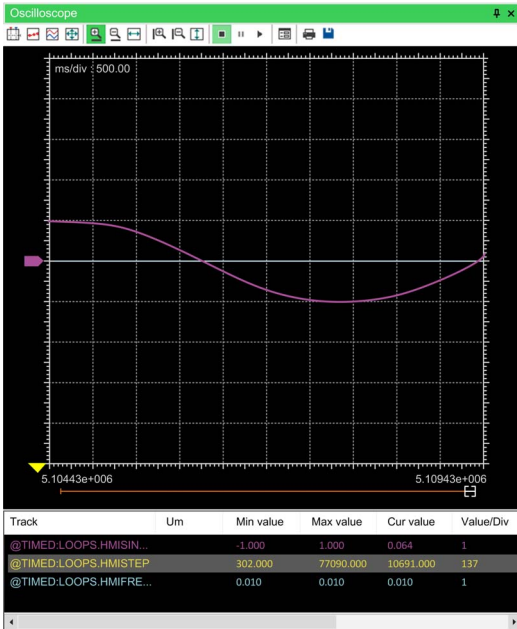
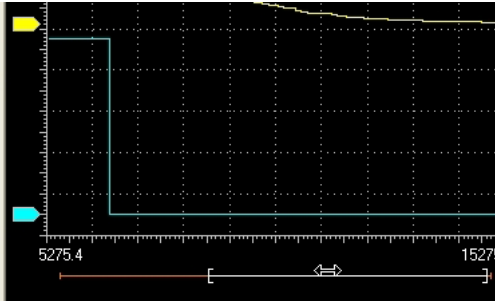
Saving Data to a File

You can save the samples acquired by the Oscilloscope to a file in order to further analyze the data with other tools:

Step	Action
1	Stop the acquisition  before saving data to a file.
2	Click the  Save tracks data into file in the Oscilloscope toolbar.
3	Choose between the available output file formats: • OSC is a simple plain-text file, containing time and value of each sample. • OSCX is an XML file that includes more complete information, which can be further analyzed with another tab.
4	Choose a file name and a destination directory, then confirm the operation.

Printing the Graph

Follow this procedure to print a view of the data plotted in the Oscilloscope:

Step	Action																								
1	Either suspend  or stop  the acquisition.																								
2	Move the time slide bar and adjust the zoom in order to include in the view the elements you want to print:  The screenshot shows the Oscilloscope window with a sine wave plot. Below the plot is a table with the following data: <table><tr><th>Track</th><th>Um</th><th>Min value</th><th>Max value</th><th>Cur value</th><th>Value/Div</th></tr><tr><td>@TIMED-LOOPS.HMSIN...</td><td></td><td>-1.000</td><td>1.000</td><td>0.064</td><td>1</td></tr><tr><td>@TIMED-LOOPS.HMISTEP</td><td></td><td>302.000</td><td>77090.000</td><td>10691.000</td><td>137</td></tr><tr><td>@TIMED-LOOPS.HMIFRE...</td><td></td><td>0.010</td><td>0.010</td><td>0.010</td><td>1</td></tr></table>  The screenshot shows a zoomed-in view of the sine wave plot, highlighting the time range from 5275.4 to 15275.4.	Track	Um	Min value	Max value	Cur value	Value/Div	@TIMED-LOOPS.HMSIN...		-1.000	1.000	0.064	1	@TIMED-LOOPS.HMISTEP		302.000	77090.000	10691.000	137	@TIMED-LOOPS.HMIFRE...		0.010	0.010	0.010	1
Track	Um	Min value	Max value	Cur value	Value/Div																				
@TIMED-LOOPS.HMSIN...		-1.000	1.000	0.064	1																				
@TIMED-LOOPS.HMISTEP		302.000	77090.000	10691.000	137																				
@TIMED-LOOPS.HMIFRE...		0.010	0.010	0.010	1																				
3	Click the  Print graph item.																								

Section 13.4

Edit and Debug Mode

Overview

Description

While both the **Watch** window and the Oscilloscope do not make use of the source code, all the other debuggers do: when debug mode is on, changes to the source code are inhibited and debug tools become active.

PROGRAMMING automatically enables debug mode when at least one of the following conditions are met:

- At least one breakpoint is correctly set.
- At least one trigger (graphic or textual) is correctly set.
- Live debug mode is on.

If or when all the conditions are not met, the debug mode is automatically disabled and **PROGRAMMING** enters in edit mode.

The status bar shows whether the debug mode is active or not:



You cannot enter the debug mode if the connection status differs from **Connected** and the application in the software and the controller are the same as indicated by **SOURCE OK**.

Section 13.5

Live Debug

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	322
SFC Simulation	323
LD Simulation	324
FBD Simulation	324
IL and ST Simulation	325

Overview

Description

PROGRAMMING can display meaningful animation of the current and changing state of execution over time of a Program Organization Unit (POU) coded in any IEC 61131-3 programming language.

To enable and disable the live debug mode, you may click  **Debug → Live debug mode**.

The controller allows you to force the state of selected outputs and variables to a defined value for the purposes of system testing, commissioning, and maintenance. You can force the value of an output and/or variable while your controller is connected to EcoStruxure Machine Expert - HVAC.

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 EcoStruxure Machine Expert - HVAC without removing the forcing.

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

SFC Simulation

Description

As explained in the relevant section of the language reference, an SFC POU is structured in a set of steps, each of which is either active or inactive at any given moment. Once started up, this SFC-specific debugging tool animates the SFC documents by highlighting the active steps.

Animation OFF	Animation ON	Animation ON in hold status
A portion of an SFC network is displayed, diagram animation being off.	The same portion of network is displayed when the live debug mode is active. The picture shows that steps S1 and S3 are currently active, whereas Init, S2, and S4 are inactive.	The same portion of network is displayed with steps S1 and S3 that are currently active but in hold status. This may occur in SFC blocks when they are children of a parent in inactive status.

The SFC animation manager tests periodically the state of all steps, you are not allowed to edit the sampling period. A step may remain active too briefly to be displayed. The fact that a step is never highlighted does not imply that its action is not executed. It may simply mean that the sampling rate is too slow to detect the execution.

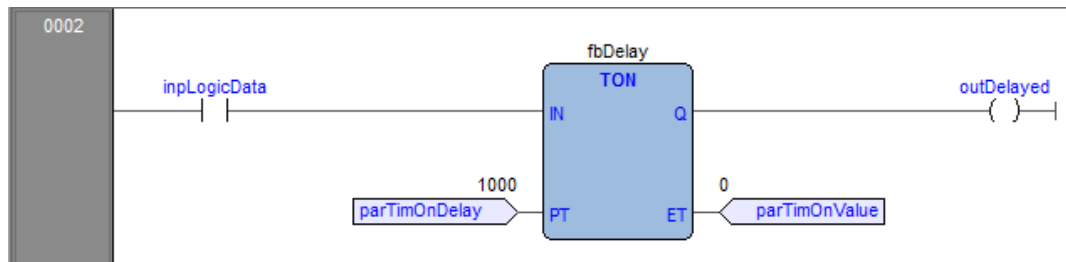
Debugging Actions and Conditions

As explained in the SFC language reference, a step can be assigned to an action, and a transition can be associated with a condition. Actions and conditions can be coded in any of the IEC 61131-3 languages. General-purpose debugging tools can be used within each action/condition, as if it was a stand-alone POU.

LD Simulation

Description

In live debug mode, Ladder Diagram schemes are animated by highlighting the contacts and coils whose value is true (in the example, i1 and i2):

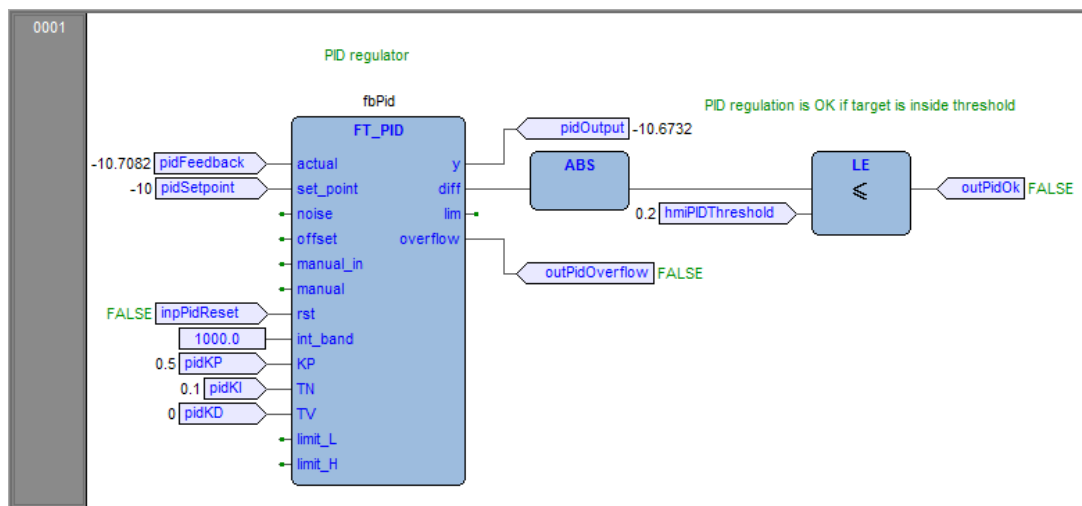


The LD animation manager tests periodically the state of all the elements. It may happen that an element remains true too briefly to be displayed. The fact that an element is never highlighted does not imply that its value never becomes true (the sampling rate may be too slow).

FBD Simulation

Description

In live debug mode, **PROGRAMMING** displays the values of all the visible variables directly in the graphical source code editor:



This works for both FBD and LD programming language.

The FBD animation manager tests periodically the state of all the elements. It may happen that an element remains true too briefly to be displayed. The fact that an element is never highlighted does not imply that its value never becomes true (the sampling rate may be too slow).

IL and ST Simulation

Description

The live debug mode also applies to textual source code editors (the ones for IL and ST). You can watch the values of a variable by hovering with the mouse over it:

```
0001
0002
0003  (* Analog output 0 = analog inp 0 + analog inp 1 *)
0004
0005  aout0 := ainp0 + ainp1;
0006
0007  (* SFC state logic *)
0008
0009  fbStati( enab := inp10, run := inp11, stop := inp12 );
0010
0011  cnt := cnt + 1;
0012
0013      -29133
0014
```

Section 13.6

Triggers

What Is in This Section?

This section contains the following topics:

Topic	Page
Trigger Window	326
Debugging with Trigger Windows	332

Trigger Window

Description

The **Trigger window** tool allows you to select a set of variables and to have them updated synchronously in a special pop-up window.

Pre-Conditions to Open a Trigger Window

Memory availability

A trigger window takes a segment in the application code sector, having a well-defined length. Obviously, in order to start up a trigger window, it is necessary that a sufficient amount of application memory is available, otherwise an error message appears.

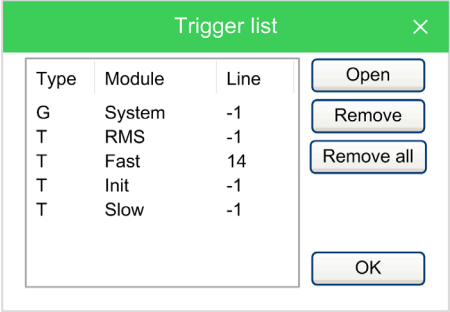
Incompatibility with graphic trigger windows

A graphic trigger window takes the whole free space of the application code sector. Once such a debugging tool has been started, it is not possible to add any trigger window, and an error message appears if you attempt to start a new window. Once the graphic trigger window is closed, trigger windows are enabled again.

All the trigger windows existing before the starting of a graphic trigger window keep working normally. You are not allowed to add new ones.

Trigger Window Toolbar

Trigger window icons are part of the **Debug** toolbar and are enabled only if **PROGRAMMING** is in debug mode.

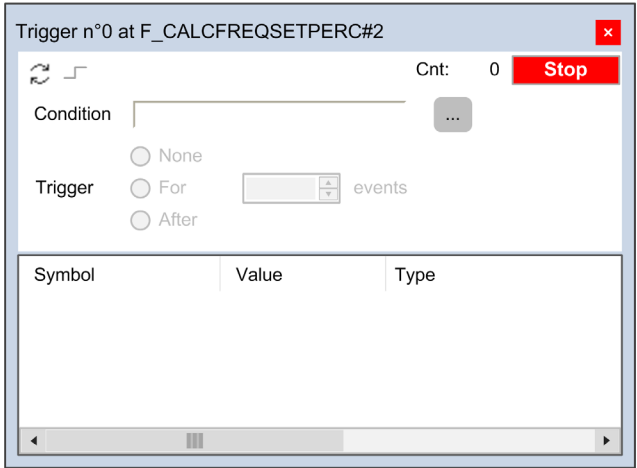
Icon	Command	Description																		
	Add/remove text trigger	To start a trigger window, select the point of the PLC code where to insert the relative trigger and then click this button. The same procedure applies to trigger window removal: in order to definitely close a debug window, click once the instruction/block where the trigger was inserted, then click this button again. Shortcuts key: F9 .																		
	Add/remove graphic trigger	This button operates exactly as the  Add/remove text trigger , except for that it opens a graphic trigger window. It can be used likewise also to remove a graphic trigger window. Shortcuts key: Shift + F9 .																		
	Remove all triggers	Clicking this button causes all the existing trigger windows and the graphic trigger window to be removed simultaneously. Shortcuts key: Ctrl+Shift+F9 .																		
	Trigger list	This button opens a dialog listing all the existing trigger windows:  <table border="1"> <thead> <tr> <th>Type</th><th>Module</th><th>Line</th></tr> </thead> <tbody> <tr> <td>G</td><td>System</td><td>-1</td></tr> <tr> <td>T</td><td>RMS</td><td>-1</td></tr> <tr> <td>T</td><td>Fast</td><td>14</td></tr> <tr> <td>T</td><td>Init</td><td>-1</td></tr> <tr> <td>T</td><td>Slow</td><td>-1</td></tr> </tbody> </table> Shortcut key: Ctrl+I.	Type	Module	Line	G	System	-1	T	RMS	-1	T	Fast	14	T	Init	-1	T	Slow	-1
Type	Module	Line																		
G	System	-1																		
T	RMS	-1																		
T	Fast	14																		
T	Init	-1																		
T	Slow	-1																		

Each record refers to a trigger window, either graphic or textual. The following table explains the meaning of each field.

Field	Description
Type	T : trigger window. G : graphic trigger window.
Module	Name of the program, function, or function block where the trigger is placed. If the module is a function block, this field contains its name, not the name of its instance where you put the trigger.
Line	For the textual languages (IL, ST) indicates the line in which the trigger is placed. For the other languages, the value is always -1.

Trigger Window Interface

Setting a trigger causes a pop-up window to appear, which is called **Interface** window: this is the interface to access the debugging functions that the trigger window makes available. It consists of three elements, as presented below:



Caption bar

The **Caption** bar of the pop-up window shows information on the location of the trigger which causes the refresh of the **Variables** window, when reached by the processor.

The text in the **Caption** bar has the following format:

Trigger n° X at ModuleName#Location

where

X	Trigger identifier.
ModuleName	Name of the program, function, or function block where the trigger was placed.
Location	Exact location of the trigger, within module ModuleName . If ModuleName is in IL, Location has the following format: N1 Otherwise, if ModuleName is in FBD, it becomes: N2\$BT:PID where: N1 = instruction line number N2 = network number BT = block type (operand, function, function block, and so on) PID = block identifier

Controls section

This dialog box allows you to control the refresh of the trigger window to get more information on the code under scope. A detailed description of the function of each control is given in the **Trigger window** controls section. Refer to Using Controls description ([see page 343](#)).

All controls are not accessible until at least one variable is dragged into the debug window.

The Variables section

This lower section of the **Debug** window is a table consisting of a row for each variable that you dragged in. Each row has several fields: the name of the variable, its value, its type, its location (@task:ModuleName), and its description read from memory during the last refresh:

Trigger n°0 at F_CALCFREQSETPERC#2

Cnt: 0

Triggered

Condition

None

For

After

events

Symbol	Value	Type
A	10	UINT
B	TRUE	BOOL

Trigger Window: Drag and Drop Information

To watch a variable, you need to copy it to the lower section of the **Debug** window.

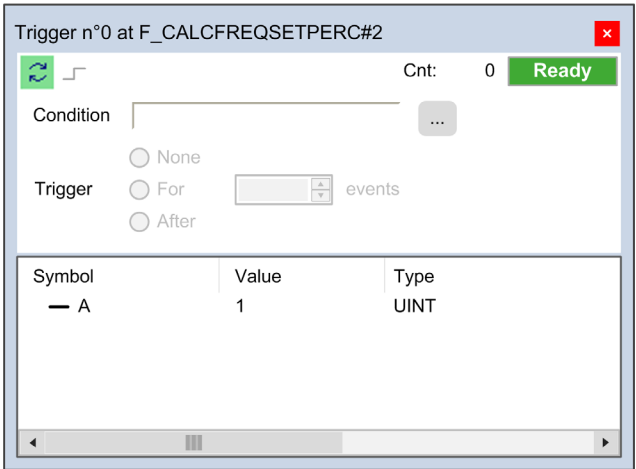
This section is a table consisting of a row for each variable you dragged in. You can drag into the trigger window only variables local to the module where you placed the relative trigger, or global variables, or parameters. You cannot drag variables declared in another program, or function, or function block.

Refresh of the Values

Let consider the following example:

0001	LD 1
0002	ST a
0003	
0004	LD 2
0005	ST a
0006	
0007	LD 3
0008	ST a
0009	

The value of variables is refreshed every time the window manager is triggered that is every time the processor executes the instruction marked by the green arrowhead. However, you can set controls in order to have variables refreshed only when triggers satisfy the more limiting conditions you define.



The value of the variables in column **Symbol** is read from memory just before the marked instruction (in this case: the instruction at line 5) and immediately after the previous instruction (the one at line 4) has been performed.

Thus, in the previous example the second ST statement has not been executed yet when the new value of **a** is read from memory and displayed in the trigger window. Thus the result of the second ST **a** is 1.

Trigger Window Controls

Trigger window controls allow you to supervise the working of this debugging tool.

Trigger window controls act in a well-defined way on the behavior of the window, regardless for the type of the module (either IL or FBD) where the related trigger has been inserted.

All controls are not accessible until at least one variable is dragged into the **Variables** window.

Window controls are made accessible to you through the gray top half of the debug window:

Button	Command	Description
	Start/Stop	This control is used to start a triggering session. While triggering, you can click this button to stop the session. Otherwise, the session automatically stops when conditions are reached. You can click this button to resume the triggering session.
	Set step Trigger	This control is used to execute a single step trigger. It is enabled only when there is no active triggering session and None is selected. The defined condition specified then is active. After the single step trigger is done, triggering session automatically stops.

Trigger counter

A read-only control **Cnt** counts how many times the debug window manager has been triggered since the window was installed.

The window manager automatically resets this counter every time a new triggering session is started.

Trigger state

This read-only control displays the state of the **Debug** window. It can assume the following values.

	The trigger has not occurred during the task execution.
	The trigger has occurred during the task execution.
	System is not triggering. It has been stopped by you or a halt condition has been reached.
	Communication with target interrupted, the state of the trigger window cannot be determined.

User-defined condition

If you define a condition by using this control, the values in the **Debug** window are refreshed every time the window manager is triggered and the user-defined condition is true.

After you have entered a condition, the control displays its simplified expression:

Condition 

Counters

These controls allow you to define conditions on the trigger counter:

☒ None
Trigger ☐ For  events
☐ After

The trigger window can be in one of the following three states.

- **None**: no counter has been started up, thus no condition has been specified upon the trigger.
- **For**: assuming that you gave the counter limit the value *N*, the window manager adds **1** to the current value of the counter and refreshes the value of its variables, each time the debug window is triggered. However, when the counter equals *N*, the window stops refreshing the values, and it changes to the **Stop** state.
- **After**: assuming that you gave the counter limit the value *N*, the window manager resets the counter and adds **1** to its current value each time it is triggered. The window remains in the **Ready** state and does not update the value of its variables until the counter reaches *N*.

Debugging with Trigger Windows

Introduction

The trigger window tool allows you to select a set of variables and to have their values displayed and updated synchronously in a pop-up window. Unlike the **Watch** window, trigger windows refresh simultaneously all the variables they contain, every time they are triggered.

Opening a Trigger Window from an IL Module

For this example, assume that you have an IL module containing the following instructions:

```

0001
0002 LD a
0003 ADD b
0004 ST a
0005
0006 LD c
0007 ADD d
0008 ST c
0009
0010 LD k
0011 ADD 1
0012 ST k
0013

```

You want to know the value of `b`, `d`, and `k`, just before the **ST k** instruction is executed.

To do so, move the cursor to line 12:

```

0009
0010 LD k
0011 ADD 1
0012 ST k
0013

```

Then you can click  **Debug → Add/remove text trigger.**

In both cases, a green arrowhead appears next to the line number, and the related trigger window pops up.

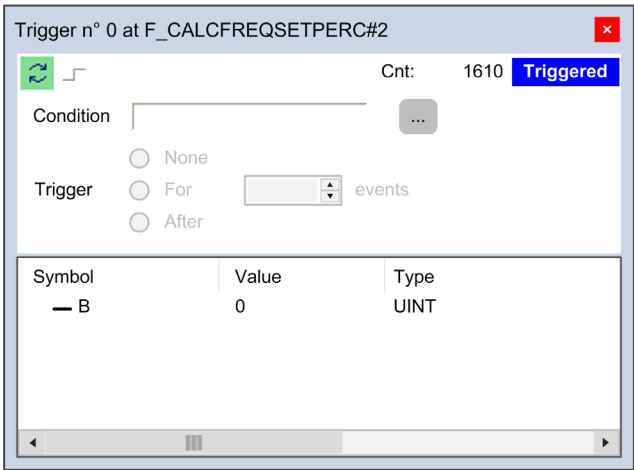
Not all the IL instructions support triggers. For example, it is not possible to place a trigger at the beginning of a line containing a `JMP` statement.

Adding a Variable to a Trigger Window from an IL Module

In order to watch the value of a variable, you need to add it to the trigger window.
Select a variable by double-clicking it, and then drag it into the **Variables** window that is the lower white box in the pop-up window:

```
0001
0002 LD a
0003 ADD b
0004 ST a
0005
0006 LD c
0007 ADD d
0008 ST c
0009
0010 LD k
0011 ADD 1
0012 ST k
0013
```

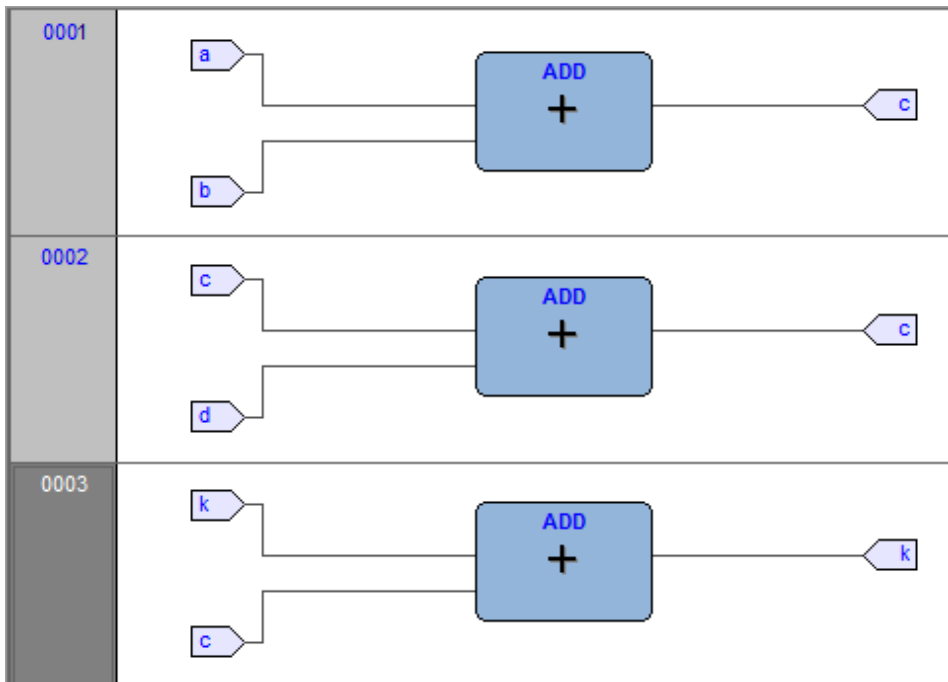
The name of the variable is now displayed in the **Symbol** column:



The same procedure applies to all the variables you wish to monitor.

Opening a Trigger Window from an FBD Module

For this example, assume that you have an FBD module containing the following instructions:

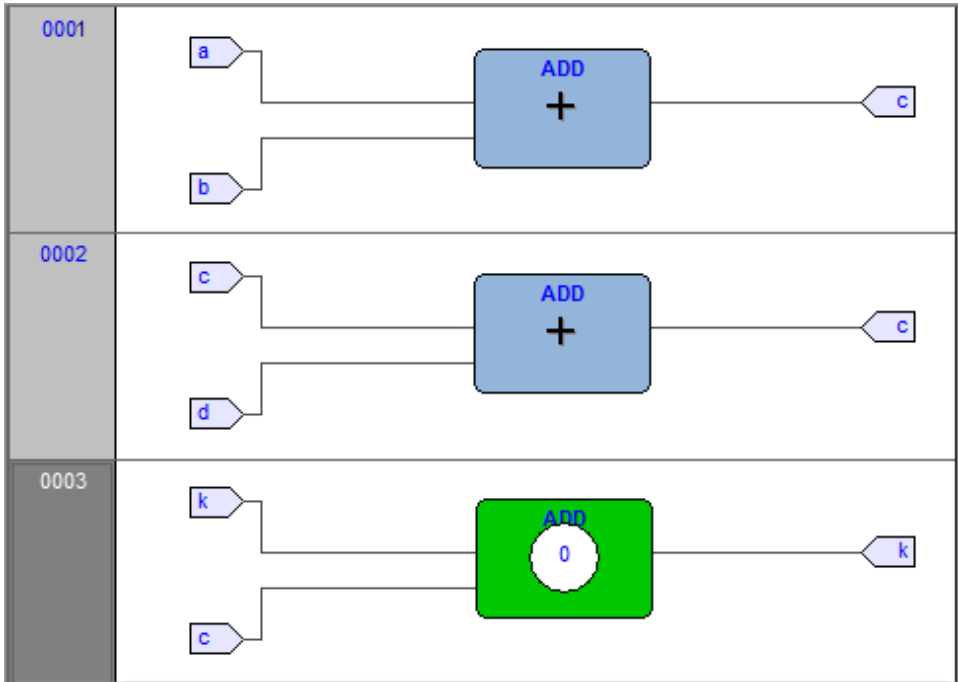


You want to know the values of C, D, and K, just before the `ST k` instruction is executed.

Provided that you can never place a trigger in a block representing a variable such as  you must select the first available block preceding the selected variable. In the example of the previous figure, you must move the cursor to network 3, and click the `ADD` block.

You can click  **Debug → Add/remove text trigger.**

In both cases, the color of the selected block turns to green, a white circle with a number inside appears in the middle of the block, and the related trigger window pops up:



When preprocessing FBD source code, the compiler translates it into IL instructions. The **ADD** instruction in network 3 is expanded to:

```
LD k
ADD 1
ST k
```

When you add a trigger to an FBD block, you place the trigger before the first statement of its IL equivalent code.

Adding a Variable to a Trigger Window from an FBD Module

To watch the value of a variable, you need to add it to the trigger window. For example, you want to monitor the value of variable `k` of the FBD code:

Click  **Edit → Watch mode.**

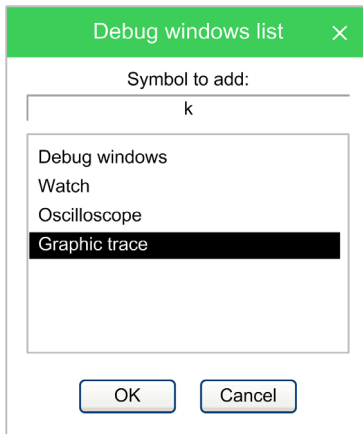
The cursor will become as follows: .

Now you can click the block representing the variable you wish to be displayed in the trigger window.

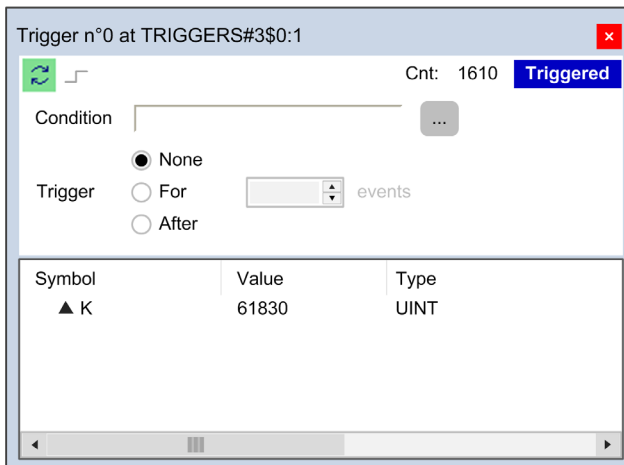
In the example, click the button block:



A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked:



In order to display the variable `k` in the trigger window, select its reference in the **Debug windows** column, then click **OK**. The name of the variable is now displayed in the **Symbol** column:

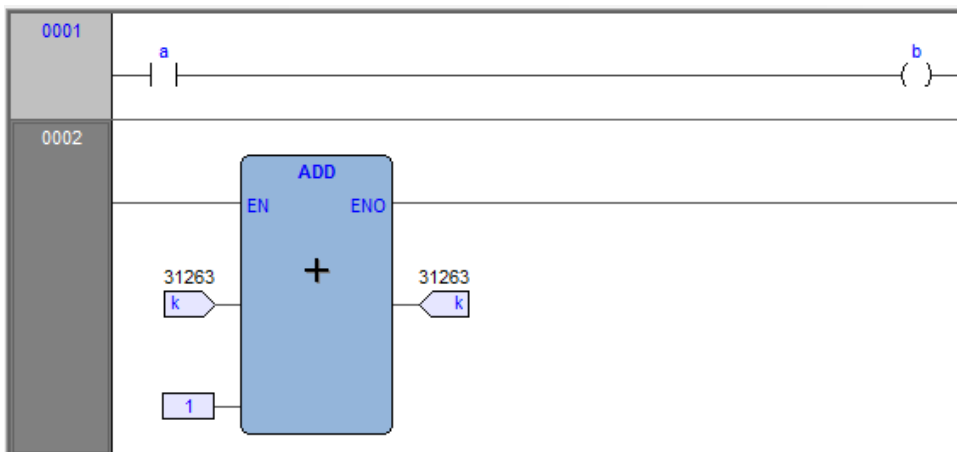


The same procedure applies to all the variables you wish to monitor:

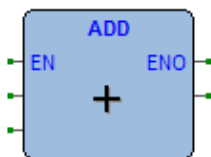
Once you have added to the **Graphic watch** window all the variables you want to observe, you can click  **Edit** → **Insert/Move mode**, to let the cursor take back its original shape.

Opening a Trigger Window from an LD Module

For this example, assume that you have an LD module containing the following instructions:



You can place a trigger on a block such as follows:

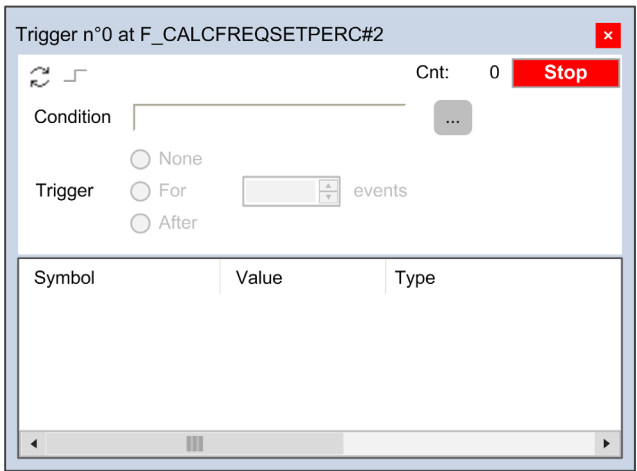
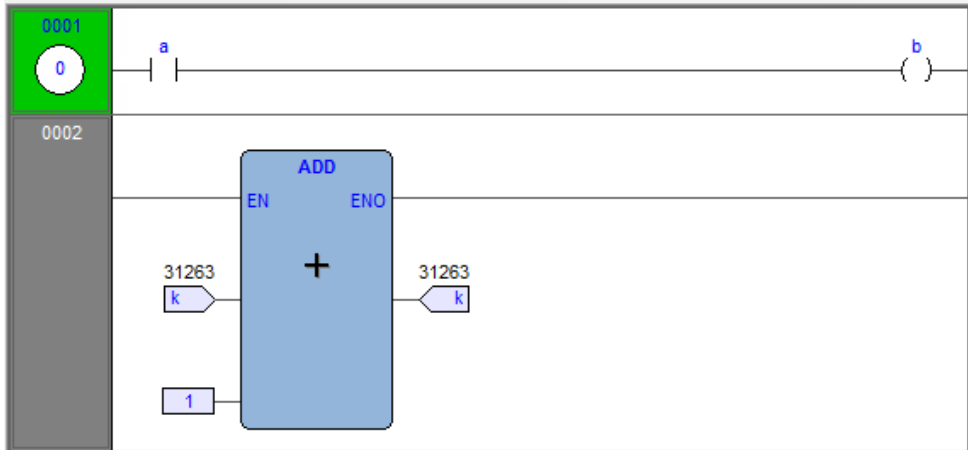


In this case, the same rules apply as to insert a trigger in an FBD module on a contact  or a coil .

In this case, follow the SE instructions. Let also assume that you want to know the value of some variables every time the processor reaches network number 1.

First you must click one of the items making up network number 1. Now you can click  **Debug** → **Add/remove text trigger**.

In both cases, the gray area containing the network number turns to green, and a white circle with the number of the trigger inside appears in the middle of the area while the related trigger window pops up:



Unlike the other languages supported by **PROGRAMMING**, LD does not allow you to insert a trigger into a single contact or coil, as it lets you select only an entire network. Thus the variables in the trigger window will be refreshed every time the processor reaches the beginning of the selected network.

Adding a Variable to a Trigger Window from an LD Module

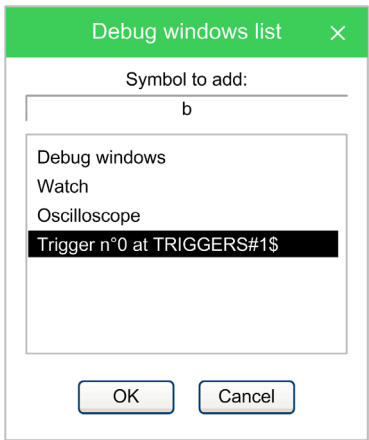
To watch the value of a variable, you need to add it to the trigger window. For example, you want to monitor the value of variable `b` in the LD code:

Click  **Edit → Watch mode.**

The cursor becomes as follows: 

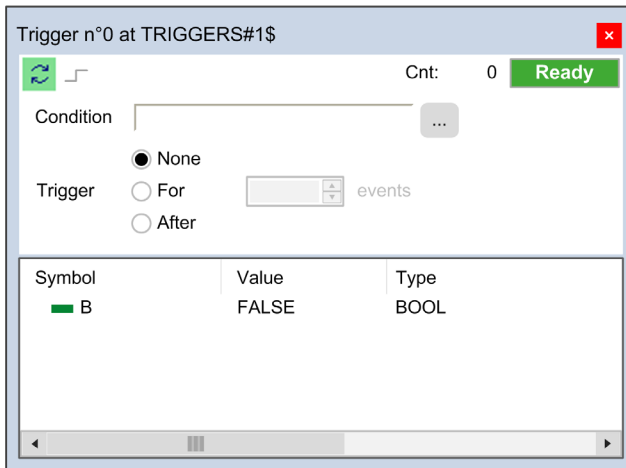
Now you can click the item representing the variable you wish to be displayed in the trigger window.

A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked.



To display variable `B` in the trigger window, select its reference in the **Debug window** column, then click **OK**.

The name of the variable is now displayed in the **Symbol** column:



The same procedure applies to all the variables you wish to monitor.

Opening a Trigger Window from an ST Module

For this example, assume that you have an ST module containing the following instructions:

```

0001
0002  a := b * b;
0003  c := c + SHR( a, 16#04 );
0004
0005  d := e * e;
0006  f := f + SHR( d, 16#04 );
0007

```

Let us also assume that you want to know the value of *e*, *d*, and *f*, just before the instruction

```
f := f + SHR( d, 16#04 )
```

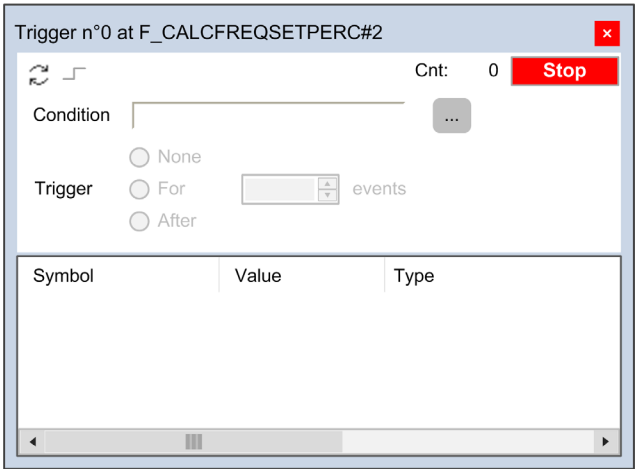
is executed. To do so, move the cursor to line 6.

Then you can click  **Debug → Add/remove text trigger**.

In both cases, a green arrowhead appears next to the line number,

```
0001
0002  a := b * b;
0003  c := c + SHR( a, 16#04 );
0004
0005  d := e * e;
0006  f := f + SHR( d, 16#04 );
0007
```

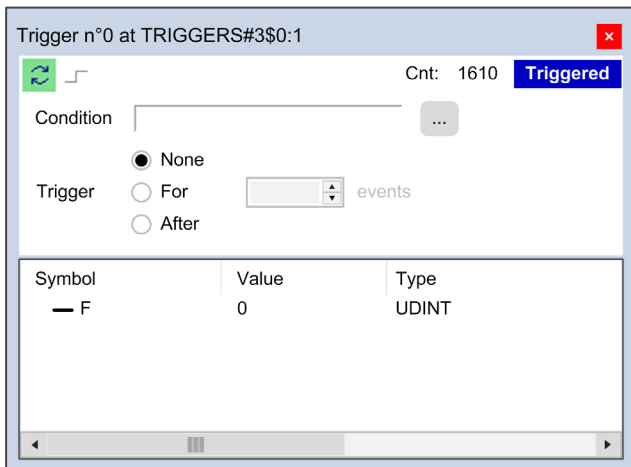
and the related trigger window pops up:



Not all the ST instructions support triggers. For example, it is not possible to place a trigger on a line containing a terminator such as `END_IF`, `END_FOR`, `END_WHILE`, and so on.

Adding a Variable to a Trigger Window from an ST Module

In order to watch the value of a variable, you need to add it to the trigger window. Select a variable, by double-clicking it, and then drag it into the **Variables** window that is the lower white box in the pop-up window. The variable name now appears in the **Symbol** column:



The same procedure applies to all the variables you wish to monitor.

Removing a Variable from the Trigger Window

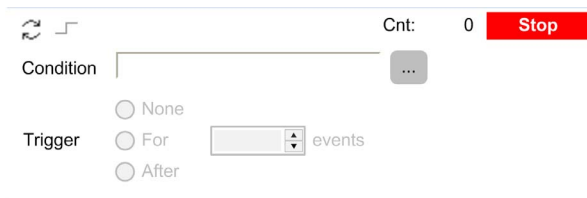
If you want to remove a variable from the trigger window, select it by clicking its name once, then press the **Delete** key.

Using Controls

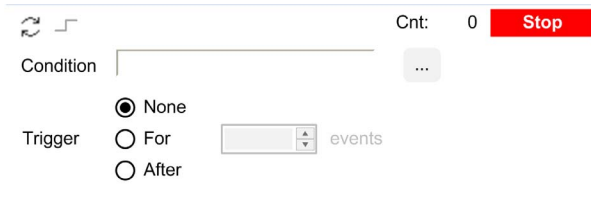
Trigger windows controls allow you to supervise the working of this debugging tool. The main purpose of trigger window controls is to let you define more limiting conditions so that variables in the **Variables** window are refreshed when the processor reaches the trigger location and these conditions are satisfied. If you do not use controls, variables are refreshed every time the processor reaches the relative trigger.

Enabling controls

When you set a trigger, all the elements in the **Control** window are disabled:



You cannot access any of the controls until at least one variable is dragged into the **Debug** window. When you do triggering automatically starts and the **Controls** window changes as follows:



Triggering can be started/stopped with the relevant button: 

Fixing the number of refresh

If you want the values to be refreshed the first time the window is triggered, select **None**, and press the  single step button, otherwise select **For** and set the events to **1**.

If you want the values to be refreshed the first **X** times the window is triggered, select **For** and set the events to **X**.

If you want the values to be refreshed after **Y** times the window is triggered, select **After** and set the events to **Y**.

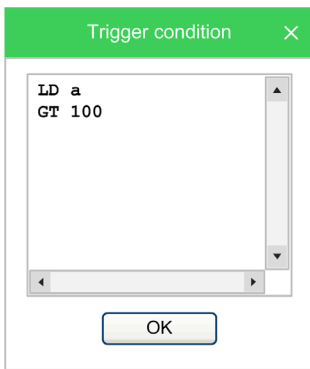
Trigger and condition settings become the settings when the triggering is (re)started.

Defining a condition

This control enables you to set a condition on the occurrences of a trigger. By default, this condition is set to **TRUE**, and the values in the debug window are refreshed every time the window manager is triggered.

If you want to put a restriction on the refreshment mechanism, you can specify a condition by clicking the button 

Then a text window appears where you can write the IL code that sets the condition:



Once you have finished writing the condition code, click the **OK** button to install it, or press the **Esc** key to cancel. If you choose to install it, the values in the debug window are refreshed every time the window manager is triggered and the user-defined condition is true.

A simplified expression of the condition now appears in the control:

Condition A GT 100 

To modify it, click the button  again.

The Trigger condition window appears, containing the IL code you originally wrote, which you can now edit.

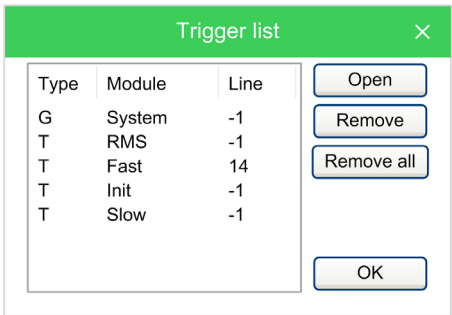
To completely remove a condition, delete all of the IL code in the window, then click **OK**.

The result of the condition code must be of type boolean (**TRUE** or **FALSE**), otherwise a compiler error occurs.

Only global variables and dragged-in variables can be used in the condition code. Namely, all variables local to the POU where the trigger was originally inserted are not valid if they have not been dragged into the debug window. No new variables can be declared in the condition window.

Closing a Trigger Window and Removing a Trigger

There are a number of actions you can take when you finish a debug session with a trigger window:



Closing the trigger window:

If you have finished watching a set of variables by using a trigger window, you may want to close the **Debug** window, without removing the trigger. If you click the button in the top right-hand corner, you hide the interface window while the window manager and the relative trigger keep working.

To resume debugging with a trigger window that you previously hid, you need to open the **Trigger list** window, select the record referred to that trigger window, and click the **Open** button.

The interface window appears with value of variables and trigger counter updated, as if it had not been closed.

Removing a trigger:

If you choose this option, you completely remove the code both of the window manager and of its trigger. Open the **Trigger list** window, select the record referred to the trigger window you want to eliminate, and click the **Remove** button.

Alternatively, you can move the cursor to the line (if the module is in IL or ST), or click the block (if the module is in FBD or LD) where you placed the trigger. Now click the  **Add/remove text trigger** button in the **Debug** toolbar.

Removing all the triggers:

Alternatively, you can remove all the existing triggers at once, regardless for which records are selected, by clicking the  **Remove all triggers** button.

Section 13.7

Graphic Triggers

What Is in This Section?

This section contains the following topics:

Topic	Page
Graphic Trigger Window	347
Debugging with the Graphic Trigger Window	354

Graphic Trigger Window

Description

The graphic trigger window tool allows you to select a set of variables, to have them sampled synchronously, and to have their curve displayed in a special pop-up window.

Sampling of the dragged-in variables occurs every time the processor reaches the position (that is, the instruction - if IL, ST - or the block - if FBD, LD) where you placed the trigger.

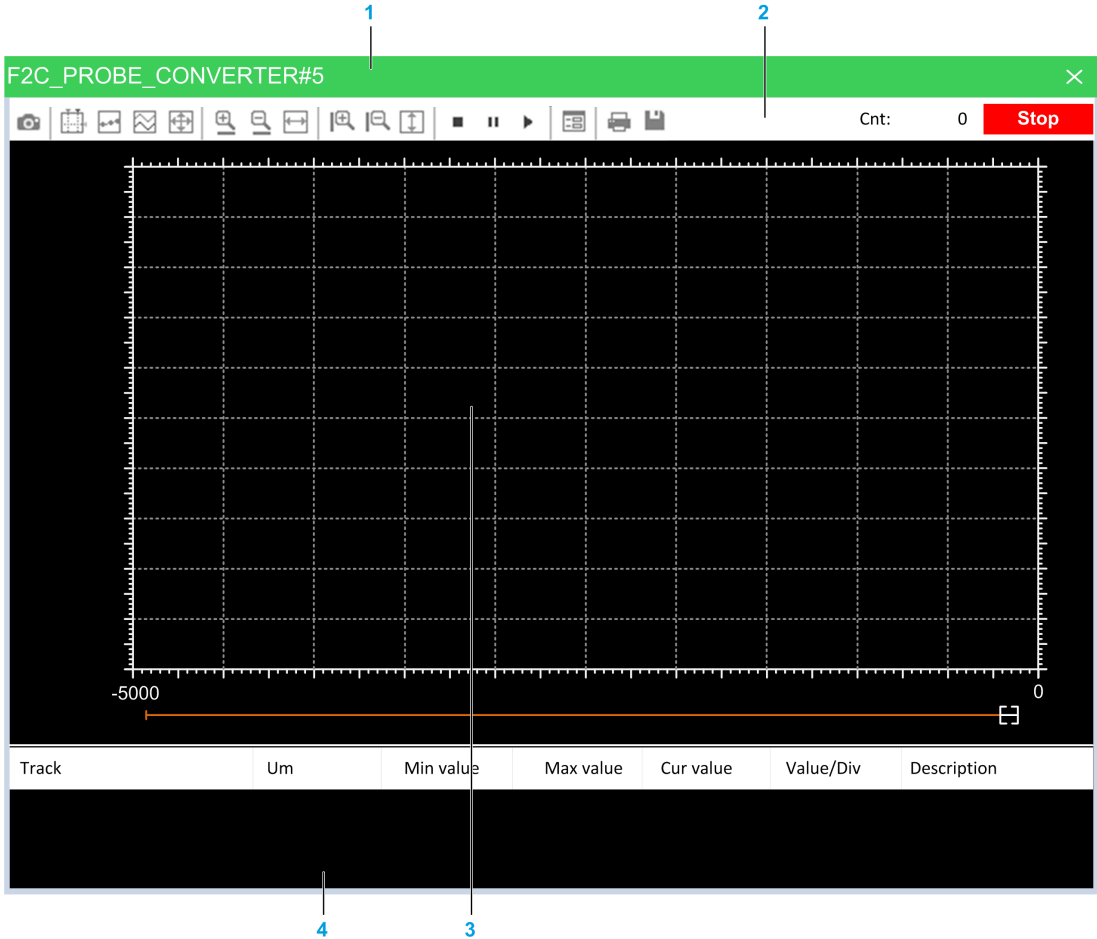
Pre-Conditions to Open a Graphic Trigger Window

Memory availability

If the available free memory space in the application code sector is insufficient, an error message to that effect is display. You must then remove some objects from memory to make the feature available.

Graphic Trigger Window Interface

Setting a graphic trigger causes a pop-up window to appear, which is called the **Interface** window. This is the main interface for accessing the debugging functions that the graphic trigger window makes available. It consists of several elements, as presented below:



- 1. Caption bar
- 2. Controls bar
- 3. Chart area
- 4. Variables window

The caption bar

The **Caption** bar at the top of the pop-up window shows information on the location of the trigger which causes the variables listed in the **Variables** window to be sampled.

The text in the caption has the following format:

`ModuleName#Location`

Where

ModuleName	Name of program, function, or function block where the trigger was placed.
Location	Exact location of the trigger within module <code>ModuleName</code>. If <code>ModuleName</code> is in IL, ST, <code>Location</code> has the format: <code>N1</code> Otherwise, if <code>ModuleName</code> is in FBD, LD, it becomes: <code>N2\$BT:BID</code> <code>N1</code> = instruction line number <code>N2</code> = network number <code>BT</code> = block type (operand, function, function block, and so on) <code>BID</code> = block identifier

The Controls bar

The Controls bar allows you to control the working of the graphic trigger window. A detailed description of the function of each control is given in the **Graphic trigger** window controls section. Refer to Graphic Trigger Window Controls description ([see page 351](#)).

The Chart area

The **Chart** area includes six items:

- Plot: area containing the plot of the curve of the dragged-in variables.
- Samples to acquire: number of samples to be collected by the graphic trigger window manager.
- Horizontal cursor: cursor identifying a horizontal line. The value of each variable at the intersection with this line is reported in the column **horz cursor**.
- Blue cursor: cursor identifying a vertical line. The value of each variable at the intersection with this line is reported in the column **left cursor**.
- Red cursor: cursor identifying a vertical line. The value of each variable at the intersection with this line is reported in the column **left cursor**.
- Scroll bar: if the scale of the x-axis is too large to display all the samples in the **Plot** area, the scroll bar allows you to slide back and forth along the horizontal axis.

The Variables window

This lower section of the **Debug** window is a table consisting of a row for each variable that you have dragged in.

Graphic Trigger Window: Variables Window

To watch a variable, you need to copy it to the lower section of the **Debug** window:

Track	Um	Min value	Max value	Cur value	Value/Div
PIDOUTPUT		-11.963	11.964	-11.952	2.99089
PIDFEEDBACK		-10.710	10.710	-7.685	2.67756

This lower section of the **Debug** window is a table consisting of a row for each variable that you dragged in. Each row has several fields, as presented in the following table:

Field	Description
Track	Name of the variable.
Um	Unit of measurement.
Min value	Minimum value in the record set.
Max value	Maximum value in the record set.
Cur value	Last sampled value of the variable.
Value/Div	How many engineering units are represented by a unit of the y-axis (that is, the space between two ticks on the vertical axis).
V blue curs	Value of the variable at the intersection with the line identified by the vertical blue cursor.
V red curs	Value of the variable at the intersection with the line identified by the vertical red cursor.
H blue curs	Value of the variable at the intersection with the line identified by the horizontal blue cursor.
H red curs	Value of the variable at the intersection with the line identified by the horizontal red cursor.

You can drag into the graphic trigger window only variables local to the module where you placed the relative trigger, or global variables, or parameters. You cannot drag variables declared in another program, function, or function block.

Sampling of Variables

The value of the variables is sampled every time the graphic trigger window manager is triggered, that is every time the processor executes the instruction marked by the green arrowhead. However, you can set controls in order to have variables sampled when triggers also satisfy further limiting conditions that you define.

The value of the variables in the column **Track** is read from memory just before the marked instruction and immediately after the previous instruction.

Graphic Trigger Window Controls

Controls allow you to specify in detail when **PROGRAMMING** samples the variables added to the **Variables** window.

Graphic trigger window controls act on the behavior of the window regardless for the type of the module (IL, ST, FBD, or LD) where the related trigger has been inserted.

Window controls are made accessible to you through the **Controls** bar of the debug window:

Button	Command	Description
	Start graphic trace	When you click this button, the acquisition starts. If acquisition is running and you click the button again, you stop the sample collection process, and you reset all the data you have acquired so far.
	Show measure bars	The two cursors (red cursor, blue cursor) may be seen and moved along their axis as long as this button is clicked. Click the button again if you want to hide all the cursors.
	Show samples	This control is used to put in evidence the exact point in which the variables are triggered at each sample.
	Vertical split	When clicked, this control splits the y-axis into as many segments as the dragged-in variables, so that the diagram of each variable is drawn in a separate band.
	Show all values	It is used to fill in the graph window all the values sampled for the selected variables in the current record set.
	Horizontal zoom + and Horizontal zoom -	Zooming in is an operation that makes the curves in the Chart area appear larger on the screen, so that greater detail may be viewed. Zooming out is an operation that makes the curves appear smaller on the screen so that it may be viewed in its entirety. Horizontal zoom acts only on the horizontal axis.
	Horizontal show all	This control is used to horizontally center record set samples. So first sample is placed on the left margin, and last is placed on the right margin of the graphic window.
	Vertical zoom + and Vertical zoom -	Zooming in is an operation that makes the curves in the Chart area appear larger on the screen so that greater detail may be viewed. Zooming out is an operation that makes the curves appear smaller on the screen so that it may be viewed in its entirety. Vertical Zoom acts only on the vertical axis.
	Vertical show all	This control is used to vertically center record set samples. So max value sample is placed near top margin and low value sample is placed on the bottom margin of the graphic window.
	Stop acquisition	This control is used to stop the acquisition.
	Pause acquisition	Click this button to suspend the acquisition.
	Re-start acquisition	Click this button to restart the acquisition.

Button	Command	Description
	Graph properties	Clicking this button causes a tabbed dialog box to appear, which allows you to set general user options affecting the action of the graphic trigger window. For more information, refer to Graphic Trigger Window Properties (see page 353).
	Print chart	Click this button to print both the Chart area and the Variables window.
	Save chart	Click this button to save the chart.

Trigger counter

This read-only control displays two numbers with the following format: `Cnt: X/Y`.

- `X` indicates how many times the debug window manager has been triggered since the graphic trigger was installed.
- `Y` represents the number of samples the graphic window has to collect before stopping data acquisition and drawing the curves.

Trigger state

This read-only control displays the state of the **Debug** window. It can assume the following values:

	No sample(s) taken, as the trigger has not occurred during the current task execution.
	Sample(s) collected, as the trigger has occurred during the current task execution.
	The trigger counter indicates that a number of samples has been collected satisfying the user request or memory constraints, thus the acquisition process is stopped.
	Communication with target interrupted; the state of the trigger window cannot be determined.

Graphic Trigger Window Properties

In order to open the properties window, you must click the **Graph properties** button in the **Controls** bar. The **Synchronous oscilloscope settings** dialog box appears.

General

Synchronous oscilloscope settings

Show grid

☒

Horizontal scale

500

Samples/div

Show time bar

☒

Buffer size

65535

Samples

(max. 65535)

Show tracks list

☒

Condition

...

Tracks list

Name	Unit	Value/div	Offset	Hide
F2C_PROBE_CONVERTER		1	0	☐
				☐
				☐
				☐
				☐
				☐
				☐
				☐

OK

Apply

Cancel

Control

Control	Description
Show grid	Select this control to display a grid in the Chart area background.
Show time bar	The scroll bar at the bottom of the Chart area is available as long as this box is selected.
Show tracks list	The Variables window is displayed as long as this box is selected, otherwise the Chart area extends to the bottom of the graphic trigger window.

Values

Control	Description
Horizontal scale Unit of the x-axis	Number of samples per unit of the x-axis. The unit x-axis is the space between two vertical lines of the background grid.
Buffer size	Number of samples to acquire. When you open the Synchronous oscilloscope settings window, after having previously dragged-in all the variables you want to watch, a default number appears in this field, representing the maximum number of samples you can collect for each variable.

User-defined condition

If you define a condition by using this control, the sampling process does not start until that condition is satisfied. Unlike trigger windows, once data acquisition begins, samples are taken every time the window manager is triggered, regardless whether the condition continues to be true.

After you enter a condition, the control displays its simplified expression:

Condition 

Tracks list

This section allows you to define some graphic properties of the plot of each variable. To select a variable, click its name in the **Name** column:

Control	Description
Unit	Unit of measurement, displayed in the table of the Variables window.
Value/div	Values per unit of the y-axis. The unit y-axis is the space between two horizontal lines of the background grid.
Offset	Set a value to apply an offset value on the graph.
Hide	Select this flag to hide the selected track on the graph.

Click **Apply** to make your changes effective, or click **OK** to apply your changes and to close the **Synchronous oscilloscope settings** window.

Debugging with the Graphic Trigger Window

Description

The graphic trigger window tool allows you to select a set of variables and to have them sampled synchronously and plotted in a special pop-up window.

Opening the Graphic Trigger Window from an IL Module

For this example, assume that you have an IL module containing the following instructions:

```
0001
0002 LD a
0003 ADD b
0004 ST a
0005
0006 LD c
0007 ADD d
0008 ST c
0009
0010 LD k
0011 ADD 1
0012 ST k
0013
```

Further, assume that you want to know the value of `b`, `d`, and `k`, just before the **ST k** instruction is executed. To do so, move the cursor to line 12:

```

0009
0010 LD k
0011 ADD 1
0012 ST k
0013

```

Then click  **Debug** → **Add/remove graphic trigger**.

A green arrowhead appears next to the line number, and the graphic trigger window pops up:

```

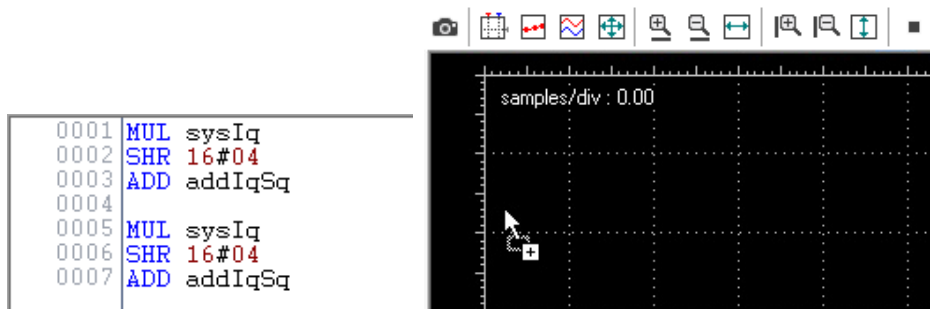
0001
0002 LD a
0003 ADD b
0004 ST a
0005
0006 LD c
0007 ADD d
0008 ST c
0009
0010 LD k
0011 ADD 1
0012 ST k
0013

```

Not all the IL instructions support triggers. For example, it is not possible to place a trigger at the beginning of a line containing a **JMP** statement.

Adding a Variable to the Graphic Trigger Window from an IL Module

In order to get the diagram of a variable plotted, you need to add it to the graphic trigger window. Select a variable, by double-clicking it, and then drag it into the **Variables** section. The variable now appears in the **Track** column:

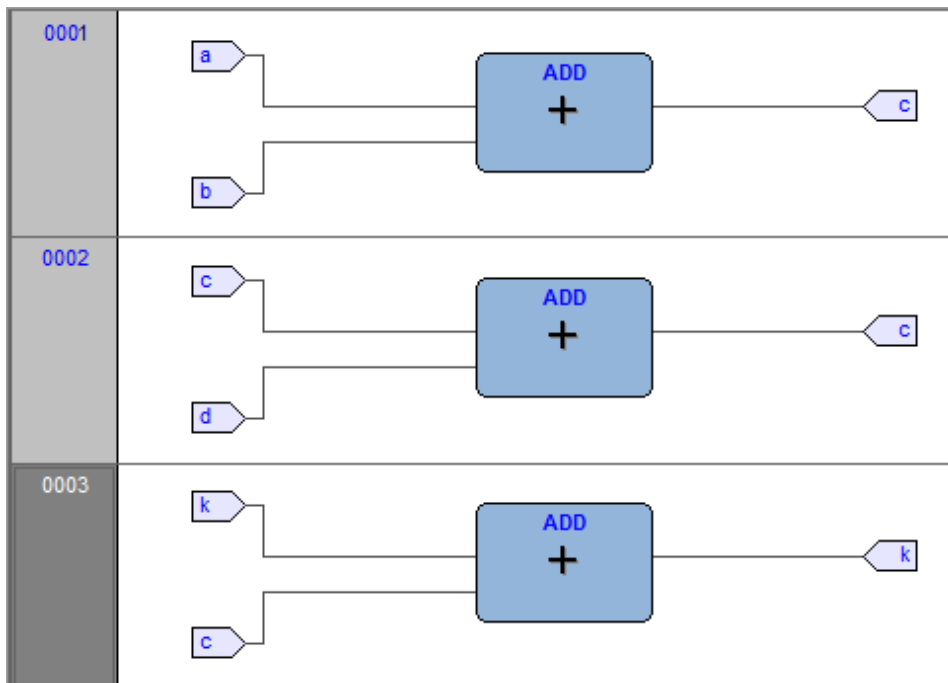


The same procedure applies to all the variables you wish to monitor.

Once the first variable is dropped into a graphic trace, the **Graphic properties** window is automatically displayed and allows you to setup sampling and visualization properties.

Opening the Graphic Trigger Window from an FBD Module

For this example, assume that you have an FBD module containing the following instructions:

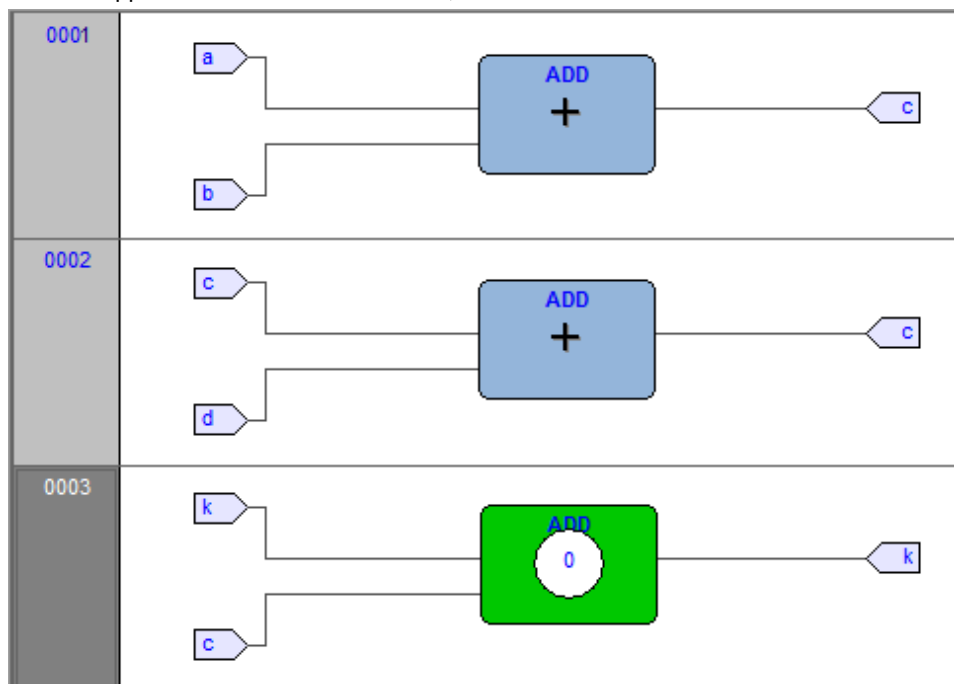


Further, assume that you want to know the values of *c*, *d*, and *k*, just before the *ST k* instruction is executed.

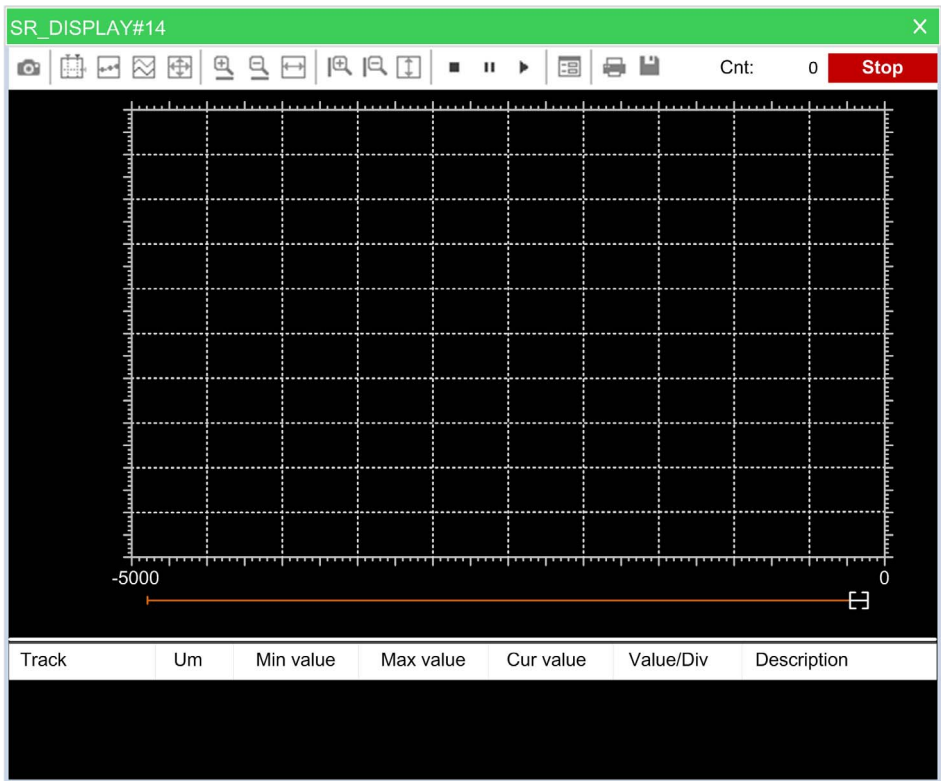
You can never place a trigger in a block representing a variable such as . You must select the first available block preceding the selected variable. In the example of the previous figure, you must move the cursor to network 3, and click the *ADD* block.

Now click  **Debug** → **Add/remove graphic trigger**.

This causes the color of the selected block to turn to green, a white circle with the trigger ID number inside to appear in the middle of the block,



and the related trigger window to pop up:



When preprocessing the FBD source code, the compiler translates it into IL instructions. The **ADD** instruction in network 3 is expanded to:

```
LD k
ADD c
ST k
```

When you add a trigger to an FBD block, you place the trigger before the first statement of its IL equivalent code.

Adding a Variable to the Graphic Trigger Window from an FBD Module

To watch a variable, you need to add it to the trigger window. As stated, assume that you want to see the plot of the variable **k** of the FBD code.

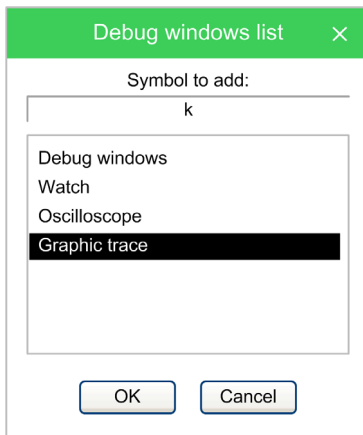
Click  **Edit → Watch mode.**

The cursor becomes as follows: 

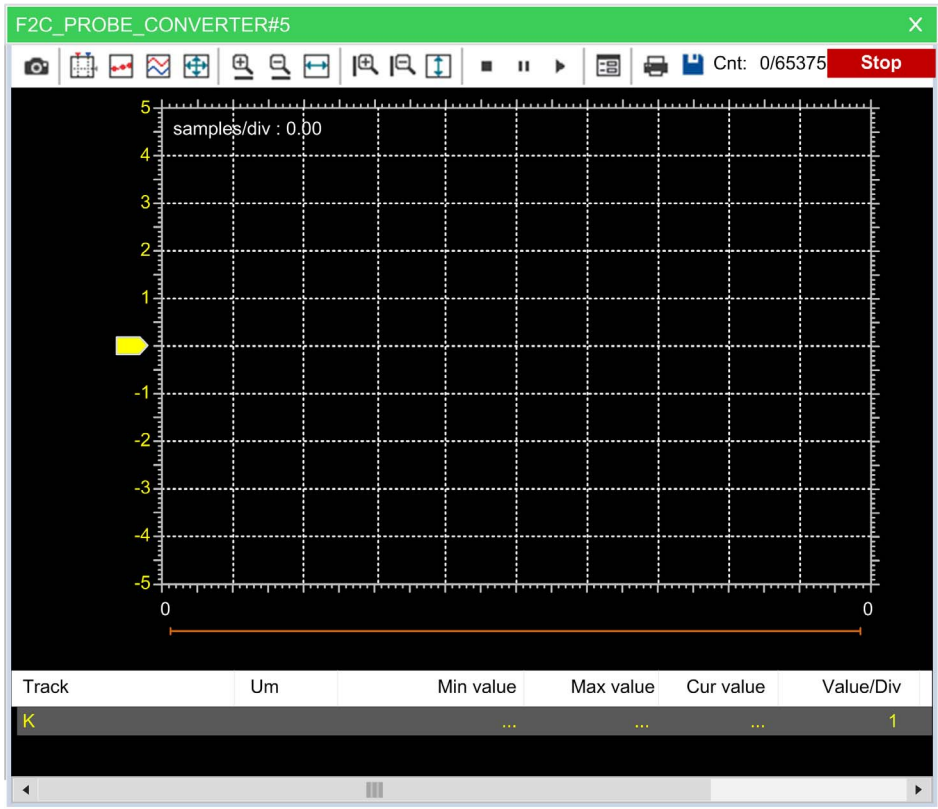
Now you can click the block representing the variable you wish to be displayed in the graphic trigger window.

In this example, click the button block: 

A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked:



In order to plot the curve of variable **k**, select **Graphic Trace** in the **Debug windows** column, then click **OK**. The name of the variable is now displayed in the **Track** column:



The same procedure applies to all the variables you wish to monitor.

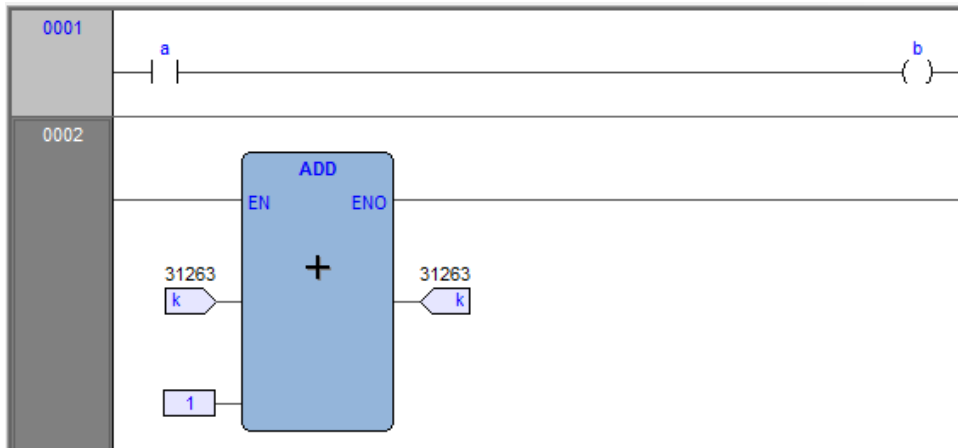
Once you have added to the **Graphic watch** window all the variables you want to monitor, you can

click  **Edit → Insert/Move mode** in order to restore the original cursor.

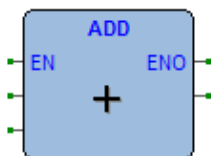
Once the first variable is dropped into a graphic trace, the **Graphic properties** window is automatically displayed and allows you to setup sampling and visualization properties.

Opening the Graphic Trigger Window from an LD Module

For this example, assume that you have an LD module containing the following instructions:



You can place a trigger on a block such as follows:

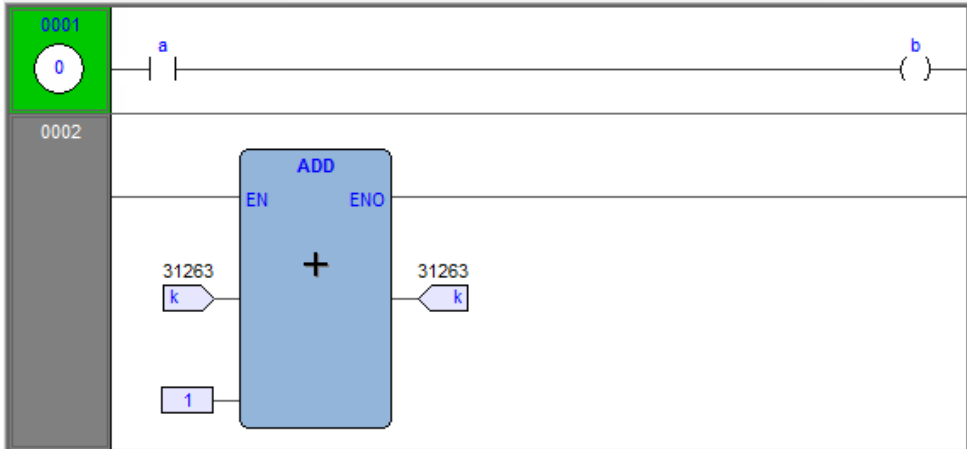


In this case, the same rules apply as to insert the graphic trigger in an FBD module.

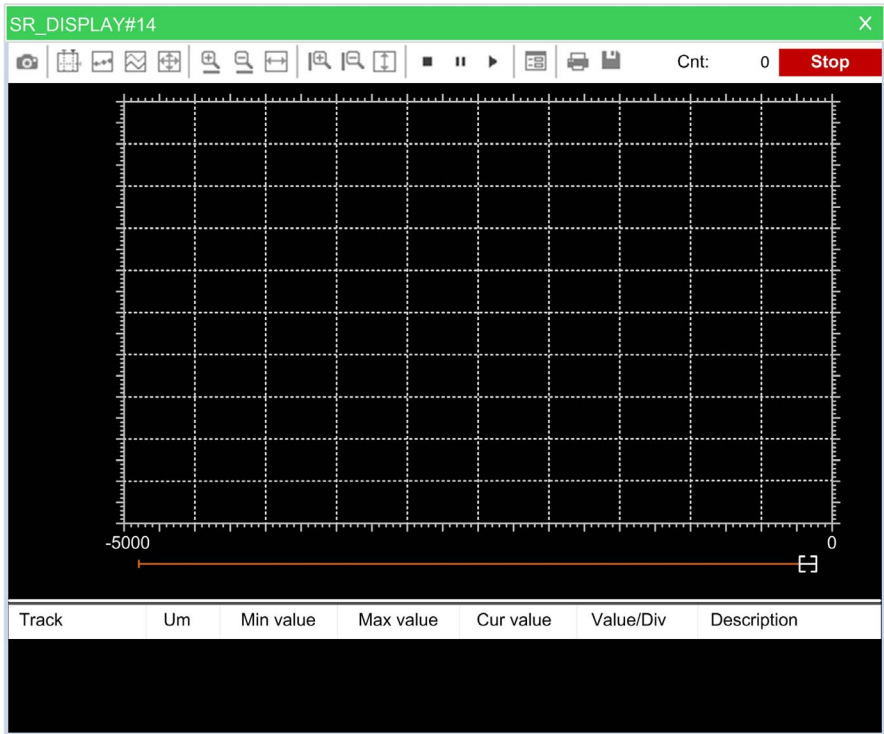
Further, assume that you want to know the value of some variables every time the processor reaches network number 1.

Click one of the items making up network number 1, then click  **Debug → Add/remove graphic trigger**

This causes the gray area containing the network number to turn to green, a white circle with a number inside to appear in the middle of the area,



and the graphic trigger window to pop up.



NOTE: Unlike the other languages supported by **PROGRAMMING**, LD does not allow you to insert a trigger before a single contact or coil, as it lets you select only an entire network. Thus the variables in the **Graphic trigger** window will be sampled every time the processor reaches the beginning of the selected network.

Adding a Variable to the Graphic Trigger Window from an LD Module

In order to watch the diagram of a variable, you must add it to the **Graphic trigger** window. In this example, assume that you want to see the plot of the variable **b** in the LD code.

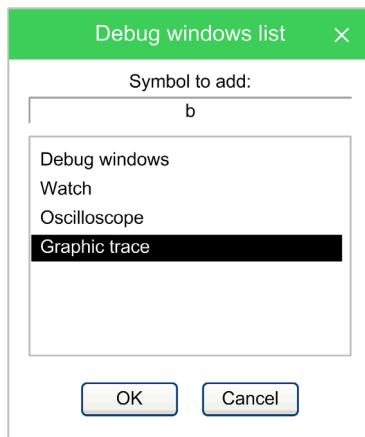
Click  **Edit → Watch mode.**

The cursor becomes as follows: .

Now you can click the item representing the variable you wish to be displayed in the **Graphic trigger** window.

A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked.

In order to plot the curve of variable **b**, select **Graphic trace** in the **Debug windows** column, then click **OK**. The name of the variable is now displayed in the **Track** column:



The same procedure applies to all the variables you wish to monitor.

Once you have added to the **Graphic watch** window all the variables you want to monitor, you can

click  **Edit → Insert/Move mode** to restore the original shape of the cursor.

Once the first variable is dropped into a graphic trace, the **Graphic properties** window is automatically displayed and allows you to setup sampling and visualization properties.

Opening the Graphic Trigger Window from an ST Module

For this example, assume that you have an ST module containing the following instructions:

```
0001
0002  a := b * b;
0003  c := c + SHR( a, 16#04 );
0004
0005  d := e * e;
0006  f := f + SHR( d, 16#04 );
0007
```

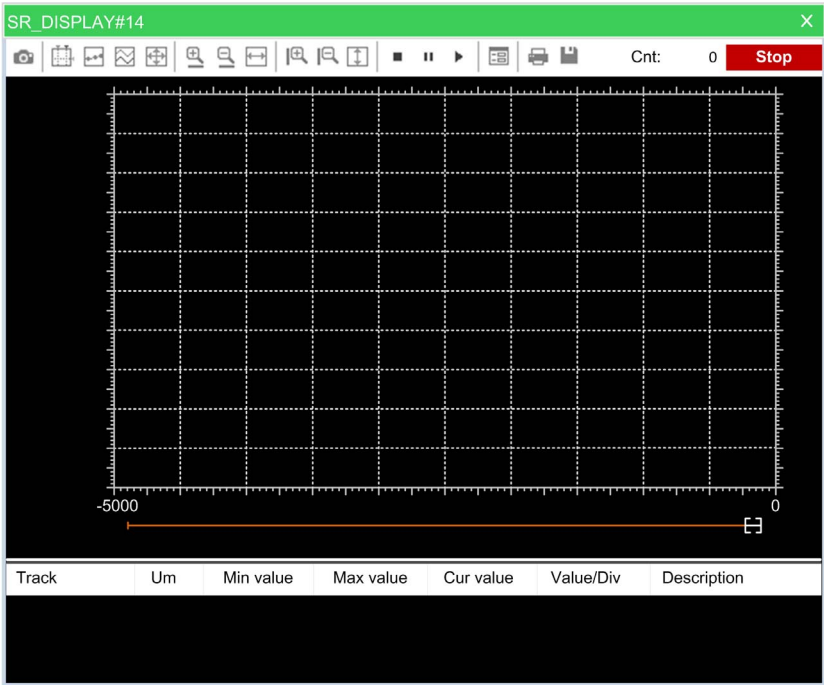
This example assumes that you want to know the value of `e`, `d`, and `f`, just before the instruction `f := f + SHR( d, 16#04 )` is executed. To do so, move the cursor to line 6.

Then click  **Debug** → **Add/remove graphic trigger**.

A green arrowhead appears next to the line number, and the **Graphic trigger** window pops up:

```
0001
0002  a := b * b;
0003  c := c + SHR( a, 16#04 );
0004
0005  d := e * e;
0006  f := f + SHR( d, 16#04 );
0007
```

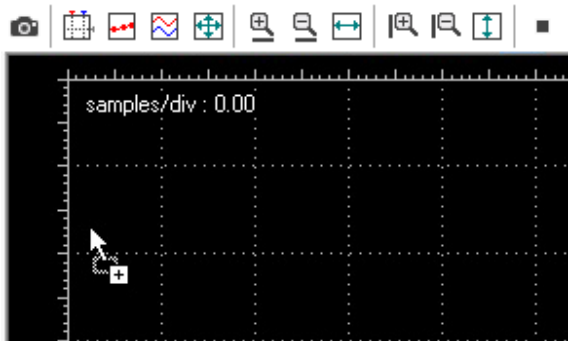




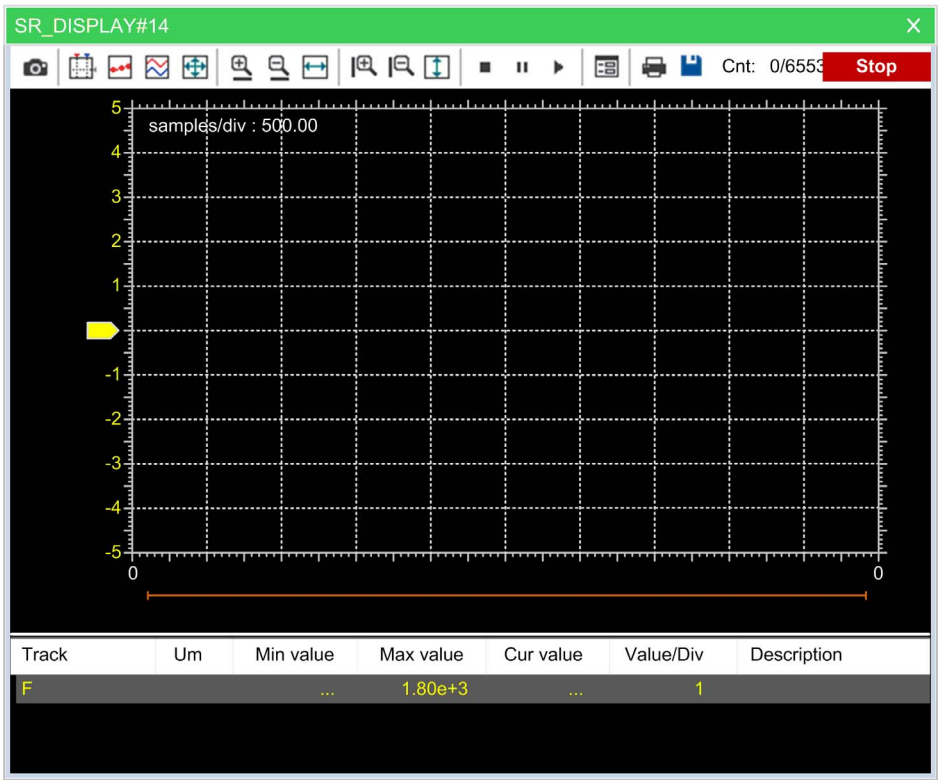
Not all the ST instructions support triggers. For example, it is not possible to place a trigger on a line containing a terminator such as `END_IF`, `END_FOR`, `END_WHILE`, and so on.

Adding a Variable to the Graphic Trigger Window from an ST Module

In order to get the diagram of a variable plotted, you need to add it to the **Graphic trigger** window. Select a variable, by double-clicking it, and then drag it into the **Variables** window that is the lower white box in the pop-up window:



The variable now appears in the **Track** column:



The same procedure applies to all the variables you wish to monitor.

Once the first variable is dropped into a graphic trace, the **Graphic properties** window is automatically displayed and allows you to setup sampling and visualization properties:

Synchronous oscilloscope settings [X]

Show grid ☒ Horizontal scale Samples/div

Show time bar ☒ Buffer size Samples (max. 65535)

Show tracks list ☒ Condition ...

Name	Unit	Value/div	Offset	Hide
F		1	0	☐
				☐
				☐
				☐
				☐
				☐
				☐
				☐
				☐
				☐

OK Apply Cancel

Removing a Variable from the Graphic Trigger Window

If you want to remove a variable from the Graphic trigger window, select it by clicking its name once, then press the **Delete** key.

Using Controls

Graphic trigger window controls allow you to supervise the working of this debugging tool and to get more information on the application.

Enabling controls

When you set a trigger, all the elements in the **Control** bar are enabled. You can start data acquisition by clicking the  **Start graphic trace** button.

If you defined a user condition, which is currently false, data acquisition does not start.

On the contrary, once the condition becomes true, data acquisition starts and continues until the  **Start graphic trace** button is released, regardless for the condition being or not still true.

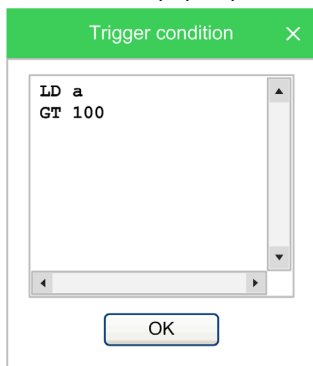
If you release the  **Start graphic trace** button before all the required samples have been acquired, the acquisition process stops and all the collected data get lost.

Defining a condition

This control enables you to set a condition on when to start acquisition. By default, this condition is set to true, and acquisition begins as soon as you click the **Enable/Disable acquisition** button. From that moment on, the value of the variables in the **Debug** window is sampled every time the trigger occurs.

In order to specify a condition, open the **Condition** tab of the **Options** dialog box, then click the relevant button: 

A text window pops up, where you can write the IL code that sets the condition:



Once you have finished writing the condition code, click **OK** to install it, or press the **Esc** key to

cancel. The collection of samples will not start until the  **Start graphic trace** button is clicked and the user-defined condition is true. A simplified expression of the condition now appears in the control:

Condition 

To modify it, click again the browser button: 

The text window appears, containing the text you originally wrote, which you can now edit.

To remove a user-defined condition, click again the button, delete the whole IL code in the text window, then click the **OK** button.

The result of the condition code must be of type boolean (**TRUE** or **FALSE**), otherwise a compiler error occurs.

Only global variables and dragged-in variables can be used in the condition code. Namely, all variables local to the module where the trigger was originally inserted are out of scope if they have not been dragged into the **Debug** window. Also, no new variables can be declared in the condition window.

Setting the scale of axes

- x-axis

When acquisition is completed, **PROGRAMMING** plots the curve of the dragged-in variables adjusting the x-axis so that all the data fit in the **Chart** window. If you want to apply a different scale, open the **General** tab of the **Graph properties** dialog box, type a number in the horizontal scale edit box, then confirm by clicking **Apply**.

- y-axis

You can change the scale of the plot of each variable through the **Tracks list** tab of the **Graph properties** dialog box. Otherwise, if you do not need to specify exactly a scale, you can use the **Zoom In** and **Zoom Out** controls.

Closing the Graphic Trigger Window and Removing the Trigger

At the end of a debug session with the graphic trigger window, you can choose between the following options:

- **Closing the graphic trigger window:**

If you have finished plotting the diagram of a set of variables by using the **Graphic trigger** window, you may want to close the **Debug** window without removing the trigger. If you click the button in the top right-hand corner, you hide the **Interface** window while the window manager and the relative trigger keep working.

To restore the **Graphic trigger** window that you previously hid:

- Open the **Trigger list** window;
- Select the record (having type **G**);
- Click the **Open** button.

The **Interface** window appears with the trigger counter properly updated, as if it had never been closed.

- **Removing the trigger:**

If you choose this option, you completely remove the code both of the window manager and of its trigger:

- Open the **Trigger list** window;
- Select the record (having type **G**);
- Click the **Remove** button.

Alternatively, you can move the cursor to the line (if the module is in IL), or click the block (if the module is in FBD) where you placed the trigger. Now click the **Graphic trace** button in the **Debug** toolbar.

- **Removing all the triggers:**

Alternatively, you can remove all the existing triggers at once, regardless for which records are selected, by clicking the  **Remove all triggers** button.

Chapter 14

Language Reference

Description

EcoStruxure Machine Expert - HVAC languages are IEC 61131-3 standard-compliant:

- Common elements
- Instruction list (IL)
- Function block diagram (FBD)
- Ladder diagram (LD)
- Structured text (ST)
- Sequential Function Chart (SFC)

Moreover, EcoStruxure Machine Expert - HVAC implements some extensions:

- Pointers
- Macros

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
14.1	Common Elements	372
14.2	Instruction List (IL)	404
14.3	Function Block Diagram (FBD)	409
14.4	Ladder Diagram (LD)	414
14.5	Structured Text (ST)	419
14.6	Sequential Function Chart (SFC)	430
14.7	EcoStruxure Machine Expert - HVAC Language Extensions	442

Section 14.1

Common Elements

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	372
Basic Elements	372
Elementary Data Types	373
Derived Data Types	373
Literals	376
Variables	378
Program Organization Units	382
Operator and Standard Blocks	385

Overview

Description

Common elements are textual and graphic elements shared by all the programmable controller programming languages specified by IEC 61131-3 standard.

NOTE: The definition and editing of most of the common elements (variables, structured elements, function blocks definitions, and so on) are managed by EcoStruxure Machine Expert - HVAC through specific editors, forms, and tables. EcoStruxure Machine Expert - HVAC does not allow you to edit directly the source code related to these common elements.

NOTE: The following information was derived directly from the copyrighted IEC standards.

Basic Elements

Character Set

Textual documents and textual elements of graphic languages are written by using the standard ASCII character set.

Comments

User comments are delimited at the beginning and end by the special character combinations " (*" and "*) ", respectively. Comments are allowed anywhere in the program, and they have no syntactic or semantic significance in any of the languages defined in this standard.

The use of nested comments, for example (* (* NESTED *) *), is treated as an error.

Elementary Data Types

Description

A number of elementary (pre-defined) data types is made available by EcoStruxure Machine Expert - HVAC;

Elementary data types, keyword for each data type, number of bits per data element, and range of values for each elementary data type are presented in the following table.

Keyword	Data type	Bits	Range
BOOL	Boolean	(1)	0...1
SINT	Short integer	8	-128...127
USINT	Unsigned short integer	8	0...255
INT	Integer	16	-32768...32767
UINT	Unsigned integer	16	0...65536
DINT	Double integer	32	$-2^{31} \dots 2^{31}-1$
UDINT	Unsigned long integer	32	$0 \dots 2^{32}$
BYTE	Bit string of length 8	8	-
WORD	Bit string of length 16	16	-
DWORD	Bit string of length 32	32	-
REAL	Real number	32	-3.40E+38...+3.40E+38
STRING	String of characters	-	-
(1) The implementation of the BOOL data type depends on the processor of the target device, for example, it is 1 bit long for devices that have a bit-addressable area.			

Derived Data Types

Description

Derived data types can be declared using the `TYPE . . . END_TYPE` construct. They can be used in variable declarations, in addition to the elementary data types.

Both single-element variables and elements of a multi-element variable, which are declared to be of derived data types, can be used anywhere where a variable of its parent type can be used.

Typedefs

The purpose of typedefs is to assign alternative names to existing types. There are not any differences between a typedef and its parent type, except the name.

Typedefs can be declared using the following syntax:

```
TYPE
    <enumerated data type name> : <parent type name>;
END_TYPE
```

For example, consider the following declaration, mapping the name `LONGWORD` to the IEC 61131-3 standard type `DWORD`:

```
TYPE
    LONGWORD : DWORD;
END_TYPE
```

Enumerated Data Types

An enumerated data type declaration specifies that the value of any data element of that type can only be one of the values given in the associated list of identifiers. The enumeration list defines an ordered set of enumerated values, starting with the first identifier of the list, and ending with the last.

Enumerated data types can be declared using the following syntax:

```
TYPE
    <enumerated data type name> : ( <enumeration list> );
END_TYPE
```

For example, consider the following declaration of two enumerated data types. When no explicit value is given to an identifier in the enumeration list, its value equals the value assigned to the previous identifier augmented by one.

```
TYPE
    enum1: (
        val1, (* the value of val1 is 0 *)
        val2, (* the value of val2 is 1 *)
        val3  (* the value of val3 is 2 *)
    );
    enum2: (
        k := -11,
        i := 0,
        j, (* the value of j is ( i + 1 ) = 1 *)
        l := 5
    );
END_TYPE
```

Different enumerated data types may use the same identifiers for enumerated values. To be uniquely identified when used in a particular context, enumerated literals may be qualified by a prefix consisting of their associated data type name and the `#` sign.

Subranges

A subrange declaration specifies that the value of any data element of that type is restricted between and including the specified upper and lower limits.

Subranges can be declared using the following syntax:

```
TYPE
    <subrange name> : <parent type name> ( <lower limit>..<>upper limit> );
END_TYPE
```

For example, consider the following declaration:

```
TYPE
    int_0_to_100 : INT (0..100);
END_TYPE
```

Structures

A **STRUCT** declaration specifies that data elements of that type shall contain subelements of specified types which can be accessed by the specified names.

Structures can be declared using the following syntax:

```
TYPE
    <structured type name> : STRUCT
        <declaration of structurestructure elements>
    END_STRUCT;
END_TYPE
```

For example, consider the following declaration:

```
TYPE
    structure1 : STRUCT
        elem1 : USINT;
        elem2 : USINT;
        elem3 : INT;
        elem3 : REAL;
    END_STRUCT;
END_TYPE
```

Literals

Numeric Literals

External representation of data in the various programmable controller programming languages consists of numeric literals.

There are two classes of numeric literals: integer literals and real literals. A numeric literal is defined as a decimal number or a based number.

Decimal literals are represented in conventional decimal notation. Real literals are distinguished by the presence of a decimal point. An exponent indicates the integer power of ten by which the preceding number needs to be multiplied to obtain the represented value. Decimal literals and their exponents can contain a preceding sign (+ or -).

Integer literals can also be represented in base 2, 8 or 16. The base is in decimal notation. For base 16, an extended set of digits consisting of letters A through F is used, with the conventional significance of decimal 10 through 15, respectively. Other than base 10 numbers do not contain any leading sign (+ or -).

Boolean data are represented by the keywords `FALSE` and `TRUE`.

Numerical literal features and examples are presented in the following table:

Feature description	Examples
Integer literals	-12 0 123 +986
Real literals	-12.0 0.0 0.4560
Real literals with exponents	-1.34E-12 or -1.34e-12 1.0E+6 or 1.0e+6 1.234E6 or 1.234e6
Base 2 literals	2#11111111 (256 decimal) 2#11100000 (240 decimal)
Base 8 literals	8#377 (256 decimal) 8#340 (240 decimal)
Base 16 literals	16#FF or 16#ff (256 decimal) 16#E0 or 16#e0 (240 decimal)
Boolean <code>FALSE</code> and <code>TRUE</code>	<code>FALSE</code> <code>TRUE</code>
For more details, refer to Two-character strings (see page 377).	

Character String Literals

A character string literal is a sequence of zero or more characters prefixed and terminated by the single quote character (').

Example	Explanation
' '	Empty string (length zero)
'A'	String of length one containing the single character A
' '	String of length one containing the space character
'\$ '	String of length one containing the single quote character ⁽¹⁾
'" '	String of length one containing the double quote character
'\$R\$L'	String of length two containing CR and LF characters ⁽¹⁾
'\$0A'	String of length one containing the LF character ⁽²⁾
(1) For more details, refer to Two-character strings (see page 377). (2) The three-character combination of the dollar sign (\$) followed by two hexadecimal digits shall be interpreted as the hexadecimal representation of the ASCII eight-bit character code.	

Two-character Combinations

Two-character combinations beginning with the dollar sign shall be interpreted as presented in the following table when they occur in character strings:

Combination	Interpretation when displayed
\$ \$	Dollar sign
\$ '	Single quote
\$L or \$l	Line feed
\$N or \$n	Newline
\$P or \$p	Form feed (page)
\$R or \$r	Carriage return
\$T or \$t	Tab

Variables

Foreword

Variables provide a means of identifying data objects whose contents may change, for example, data associated with the inputs, outputs, or memory of the programmable controller. A variable must be declared to be one of the elementary types. Variables can be represented symbolically, or alternatively in a manner which directly represents the association of the data element with physical or logical locations in the programmable controller's input, output, or memory structure.

Each program organization unit (POU) (program, function, or function block) contains at its beginning at least one declaration part. This declaration part consists of one or more structuring elements, which specify the types (and, if necessary, the physical or logical location) of the variables used in the organization unit. This declaration part has the textual form of one of the keywords `VAR`, `VAR_INPUT`, or `VAR_OUTPUT` as defined in the keywords section, followed in the case of `VAR` by zero or one occurrence of the qualifiers `RETAIN`, `NON_RETAIN` or the qualifier `CONSTANT`, and in the case of `VAR_INPUT` or `VAR_OUTPUT` by zero or one occurrence of the qualifier `RETAIN` or `NON_RETAIN`, followed by one or more declarations separated by semicolons and terminated by the keyword `END_VAR`. A declaration may also specify an initialization for the declared variable when a programmable controller supports the declaration by the user of initial values for variables.

Structuring Element

The declaration of a variable must be performed within the following program structuring element:

```
KEYWORD [RETAIN] [CONSTANT]
    Declaration 1
    Declaration 2
    ...
    Declaration N
END_VAR
```

Keywords and Scope

Keyword	Variable usage
<code>VAR</code>	Internal to organization unit.
<code>VAR_INPUT</code>	Externally supplied.
<code>VAR_OUTPUT</code>	Supplied by organization unit to external entities.
<code>VAR_IN_OUT</code>	Supplied by external entities, can be modified within organization unit.
<code>VAR_EXTERNAL</code>	Supplied by configuration via <code>VAR_GLOBAL</code> , can be modified within organization unit.
<code>VAR_GLOBAL</code>	Global variable declaration.

The scope (range of validity) of the declarations contained in structuring elements is local to the program organization unit (POU) in which the declaration part is contained. That is, the declared variables are accessible to other program organization units except by explicit argument passing via variables which have been declared as inputs or outputs of those units. The one exception to this rule is the case of variables which have been declared to be global.

Global variables are accessible to programs in any case, or via a `VAR_EXTERNAL` declaration to function blocks. The type of a variable declared in a `VAR_EXTERNAL` must agree with the type declared in the `VAR_GLOBAL` block.

To give access to these variables to all types of POU, without using any keyword, you must enable this option in the Code generation tab of the Project Options window ([see page 166](#)).

An error is detected if:

- Any POU attempts to modify the value of a variable that has been declared with the `CONSTANT` qualifier;
- A variable declared as `VAR_GLOBAL CONSTANT` in a configuration element or POU (the “containing element”) is used in a `VAR_EXTERNAL` declaration (without the `CONSTANT` qualifier) of any element contained within the containing element.

Qualifiers

Qualifier	Description
<code>CONST</code>	The attribute <code>CONST</code> indicates that the variables within the structuring elements are constants, that is, they have a constant value, which cannot be modified once the PLC project has been compiled.
<code>RETAIN</code>	The attribute <code>RETAIN</code> indicates that the variables within the structuring elements are retentive, that is, they keep their value even after the target device is reset or switched off.

Single-Element Variables and Arrays

A single-element variable represents a single data element of either one of the elementary types or one of the derived data types.

An array is a collection of data elements of the same data type; in order to access a single element of the array, a subscript (or index) enclosed in square brackets has to be used. Subscripts can be either integer literals or single-element variables.

To represent data matrices, arrays can be multi-dimensional; in this case, a composite subscript is required, one index per dimension, separated by commas. The maximum number of dimensions allowed in the definition of an array is three.

Declaration Syntax

Variables must be declared within structuring elements, using the following syntax:

```
VarName1 : Typename1 [ := InitialVal1 ];  
VarName2 AT Location2 : Typename2 [ := InitialVal2 ];  
VarName3 : ARRAY [ 0..N ] OF Typename3;
```

Where:

Keyword	Description
VarNameX	Variable identifier, consisting of a string of alphanumeric characters, of length 1 or more. It is used for symbolic representation of variables.
TypenameX	Data type of the variable, selected from elementary data types.
InitialValX	The value the variable assumes after reset of the target.
LocationX	Refer to Location description (see page 380).
N	Index of the last element, the array having length $N + 1$.

Location

Variables can be represented symbolically, that is, accessed through their identifier, or alternatively in a manner which directly represents the association of the data element with physical or logical locations in the input, output, or memory structure of the programmable controller.

Direct representation of a single-element variable is provided by a special symbol formed by the concatenation of the percent sign “%”, a location prefix and a size prefix, and one or two unsigned integers, separated by periods (.).

%location.size.index.index

1) location

The location prefix may be one of the following:

Location prefix	Description
I	Input location
Q	Output location
M	Memory location

2) size

The size prefix may be one of the following:

Size prefix	Description
X	Single bit size
B	Byte (8 bits) size

Size prefix	Description
W	Word (16 bits) size
D	Double word (32 bits) size

3) index.index

This sequence of unsigned integers, separated by dots, specifies the position of the variable in the area specified by the location prefix.

Example:

Direct representation	Description
%MW4.6	Word starting from the first byte of the seventh element of memory data block 4.
%IX0.4	First bit of the first byte of the fifth element of input set 0.

The absolute position depends on the size of the data block elements, not on the size prefix.

%MW4.6 and %MD4.6 begin from the same byte in memory, but the former points to an area which is 16 bits shorter than the latter.

For advanced users only: if the index consists of one integer only (no dots), then it loses any reference to data blocks, and it points directly to the byte in memory having the index value as its absolute address.

Direct representation	Description
%MW4.6	Word starting from the first byte of the seventh element of datablock 4 in memory.
%MW4	Word starting from byte 4 of memory.

Example:

```
VAR [RETAIN] [CONSTANT]
  XQuote : DINT;
  Enabling : BOOL := FALSE;
  TorqueCurrent AT %MW4.32 : INT;
  Counters : ARRAY [ 0 .. 9 ] OF UINT;
  Limits: ARRAY [0..3, 0..9]
END_VAR
```

- Variable `XQuote` is 32 bits long, and it is automatically allocated by the EcoStruxure Machine Expert - HVAC compiler.
- Variable `Enabling` is initialized to `FALSE` after target reset.
- Variable `TorqueCurrent` is allocated in the memory area of the target device, and it takes 16 bits starting from the first byte of the 33rd element of data block 4.
- Variable `Counters` is an array of 10 independent variables of type unsigned integer.

Declaring Variables in EcoStruxure Machine Expert - HVAC

Whatever the PLC language you are using, EcoStruxure Machine Expert - HVAC allows you to disregard the previous syntax, as it supplies the local variables editor, the global variables editor, and the parameters editor, which provide an interface to declare all kinds of variables.

Program Organization Units

Description

Program organization units (POU) are functions, function blocks, and programs. Program organization units can be delivered by the manufacturer, or programmed by you through the means defined in this part of the standard.

Program organization units are not recursive; that is, the invocation of a program organization unit cannot cause the invocation of the same program organization unit.

Functions

Introduction

For the purposes of programmable controller programming languages, a function is defined as a program organization unit (POU) which, when executed, yields exactly one data element, which is considered to be the function result.

Functions contain no internal state information, that is, invocation of a function with the same arguments (input variables `VAR_INPUT` and in-out variables `VAR_IN_OUT`) always yields the same values (output variables `VAR_OUTPUT`, in-out variables `VAR_IN_OUT`, and function result).

Declaration syntax

The declaration of a function must be performed as follows:

```
FUNCTION FunctionName : RetDataType
    VAR_INPUT
        declaration of input variables (see the relevant section)
    END_VAR
    VAR
        declaration of local variables (see the relevant section)
    END_VAR
    Function body
END_FUNCTION
```

Keyword	Description
FunctionName	Name of the function being declared.
RetDataType	Data type of the value to be returned by the function.
Function body	Specifies the operations to be performed upon the input variables in order to assign values dependent on the function's semantics to a variable with the same name as the function, which represents the function result. It can be written in any of the languages supported by EcoStruxure Machine Expert - HVAC.

Declaring functions

Whatever the PLC language you are using, EcoStruxure Machine Expert - HVAC allows you to disregard the previous syntax, as it supplies a friendly interface for using functions.

Function Blocks

Introduction

For the purposes of programmable controller programming languages, a function block is a program organization unit which, when executed, yields one or more values. Multiple, named instances (copies) of a function block can be created. Each instance has an associated identifier (the instance name), and a data structure containing its input, output, and internal variables. All the values of the output variables and the necessary internal variables of this data structure persist from one execution of the function block to the next. Invocation of a function block with the same arguments (input variables) does not always yield the same output values.

Only the input and output variables are accessible outside of an instance of a function block, that is, the function block's internal variables are hidden from the user of the function block.

In order to execute its operations, a function block needs to be started by another POU. Invocation depends on the specific language of the module calling the function block.

The scope of an instance of a function block is local to the program organization unit in which it is instantiated.

Declaration syntax

The declaration of a function must be performed as follows:

```
FUNCTION_BLOCK FunctionBlockName
  VAR_INPUT
    declaration of input variables (see the relevant section)
  END_VAR
  VAR_OUTPUT
    declaration of output variables
  END_VAR
  VAR_EXTERNAL
    declaration of external variables
  END_VAR
  VAR
    declaration of local variables
  END_VAR
  Function block body
END_FUNCTION_BLOCK
```

Keyword	Description
FunctionBlockName	Name of the function block being declared (note: name of the template, not of its instances).
VAR_EXTERNAL .. END_VAR	A function block can access global variables only if they are listed in a VAR_EXTERNAL structuring element. Variables passed to the FB via a VAR_EXTERNAL construct can be modified from within the FB.
Function block body	Specifies the operations to be performed upon the input variables in order to assign values to the output variables - dependent on the function block's semantics and on the value of the internal variables. It can be written in any of the languages supported by EcoStruxure Machine Expert - HVAC.

Declaring functions

Whatever the PLC language you are using, EcoStruxure Machine Expert - HVAC allows you to disregard the previous syntax, as it supplies a friendly interface for using function blocks.

Programs

Introduction

A program is defined in IEC 61131-1 as a “logical assembly of all the programming language elements and constructs necessary for the intended signal processing required for the control of a machine or process by a programmable controller system”.

Declaration syntax

The declaration of a program must be performed as follows:

```
PROGRAM < program name>
    Declaration of variables (see the relevant section)
    Program body
END_PROGRAM
```

Keyword	Description
Program Name	Name of the program being declared.
Program body	Specifies the operations to be performed to get the intended signal processing. It can be written in any of the languages supported by EcoStruxure Machine Expert - HVAC.

Writing programs

Whatever the PLC language you are using, EcoStruxure Machine Expert - HVAC allows you to disregard the previous syntax, as it supplies a friendly interface for writing programs.

Operator and Standard Blocks

Description

The availability of the following functions depends on the target device.

These functions are common to the whole set of programming languages and can be used in any programmable organization unit (POU). They are accessible from **Operators and blocks** window in **Operator and standard blocks** window tab.

Operators and standard blocks are sorted in groups:

- Arithmetic Functions and Operators (*see page 386*)
- Bistable Operators (*see page 391*)
- Bit Shift Functions (*see page 392*)
- Comparison Operators (*see page 393*)
- Conversion Functions (*see page 394*)
- Logic Functions (*see page 398*)
- Selection Functions (*see page 399*)
- String Functions (*see page 401*)

Arithmetic Functions and Operators

ABS	
Description	Absolute value. Computes the absolute value of input #0
Number of operands	1
Input data type	Any numerical type
Output data type	Same as input
Examples	<pre>OUT := ABS(-5); (* OUT = 5 *) OUT := ABS(-1.618); (* OUT = 1.618 *) OUT := ABS(3.141592); (* OUT = 3.141592 *)</pre>

ACOS	
Description	Arc cosine. Computes the principal arc cosine of input #0; result is expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := ACOS(1.0); (* OUT = 0.0 *) OUT := ACOS(-1.0); (* OUT = PI *)</pre>

ADD	
Description	Arithmetic addition. Computes the sum of the two inputs.
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Inputs
Examples	<pre>OUT := ADD(20, 40); (* OUT = 60 *)</pre>

ASIN	
Description	Arc sine. Computes the principal arc sine of input #0; result is expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := ASIN(0.0); (* OUT = 0.0 *) OUT := ASIN(1.0); (* OUT = PI / 2 *)</pre>

ATAN	
Description	Arc tangent. Computes the principal arc tangent of input #0; result is expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := ATAN(0.0); (* OUT = 0.0 *) OUT := ATAN(1.0); (* OUT = PI / 4 *)</pre>

ATAN2*	
Description	Arc tangent (with two parameters). Computes the principal arc tangent of Y/X; result is expressed in radians
Number of operands	2
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := ATAN2(0.0, 1.0); (* OUT = 0.0 *) OUT := ATAN2(1.0, 1.0); (* OUT = PI / 4 *) OUT := ATAN2(-1.0, -1.0); (* OUT = (-3/4) * PI *) OUT := ATAN2(1.0, 0.0); (* OUT = PI / 2 *)</pre>

CEIL*	
Description	Rounding up to integer. Returns the smallest integer that is greater than or equal to input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := CEIL(1.95); (* OUT = 2.0 *) OUT := CEIL(-1.27); (* OUT = -1.0 *)</pre>

COS	
Description	Cosine. Computes the cosine function of input #0 expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := COS(0.0); (* OUT = 1.0 *) OUT := COS(-3.141592); (* OUT ~ -1.0 *)</pre>

COSH*	
Description	Hyperbolic cosine. Computes the hyperbolic cosine function of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	OUT := COSH(0.0); (* OUT = 1.0 *)

DIV	
Description	Arithmetic division. Divides input #0 by input #1
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Inputs
Examples	OUT := DIV(20, 2); (* OUT = 10 *)

EXP	
Description	Natural exponential. Computes the exponential function of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	OUT := EXP(1.0); (* OUT ~ 2.718281 *)

FLOOR*	
Description	Rounding down to integer. Returns the largest integer that is less than or equal to input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	OUT := FLOOR(1.95); (* OUT = 1.0 *) OUT := FLOOR(-1.27); (* OUT = -2.0 *)

LN	
Description	Natural logarithm. Computes the logarithm with base e of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise

LN	
Output data type	LREAL where available, REAL otherwise
Examples	OUT := LN(2.718281); (* OUT = 1.0 *)

LOG	
Description	Common logarithm. Computes the logarithm with base 10 of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	OUT := LOG(100.0); (* OUT = 2.0 *)

MOD	
Description	Module. Computes input #0 module input #1
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Inputs
Examples	OUT := MOD(10, 3); (* OUT = 1 *)

MUL	
Description	Arithmetic multiplication. Multiplies the two inputs.
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Inputs
Examples	OUT := MUL(10, 10); (* OUT = 100 *)

POW	
Description	Exponentiation. Raises Base to the power Expo
Number of operands	2
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	OUT := POW(2.0, 3.0); (* OUT = 8.0 *) OUT := POW(-1.0, 5.0); (* OUT = -1.0 *)

SIN	
Description	Sine. Computes the sine function of input #0 expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := SIN(0.0); (* OUT = 0.0 *) OUT := SIN(2.5 * 3.141592); (* OUT ~ 1.0 *)</pre>

SINH*	
Description	Hyperbolic sine. Computes the hyperbolic sine function of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := SINH(0.0); (* OUT = 0.0 *)</pre>

SQRT	
Description	Square root. Computes the square root of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := SQRT(4.0); (* OUT = 2.0 *)</pre>

SUB	
Description	Arithmetic subtraction. Subtracts input #1 from input #0
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Inputs
Examples	<pre>OUT := SUB(10, 3); (* OUT = 7 *)</pre>

TAN	
Description	Tangent. Computes the tangent function of input #0 expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise

TAN	
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := TAN(0.0); (* OUT = 0.0 *) OUT := TAN(3.141592 / 4.0); (* OUT ~ 1.0 *)</pre>

TANH*	
Description	Hyperbolic tangent. Computes the hyperbolic tangent function of input #0
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	<pre>OUT := TANH(0.0); (* OUT = 0.0 *)</pre>

* function provided as extension to the IEC 61131-3 standard.

Bistable Operators

R	
Description	Boolean reset.
Number of operands	1
Input data type	BOOL
Output data type	BOOL
Examples	<pre>LD x R y ST z</pre>

S	
Description	Boolean set.
Number of operands	1
Input data type	BOOL
Output data type	BOOL
Examples	<pre>LD x S y ST z</pre>

Bit Shift Functions

ROL	
Description	Input #0 left-shifted of Input #1 bits, circular.
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Input #0
Examples	<pre>OUT := ROL(IN := 16#1000CAFE, 4); (* OUT = 16#000CAFE1 *)</pre>

ROR	
Description	Input #0 right-shifted of Input #1 bits, circular.
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Input #0
Examples	<pre>OUT := ROR(IN := 16#1000CAFE, 16); (* OUT = 16#CAFE1000 *)</pre>

SHL	
Description	Input#0 left-shifted of Input #1 bits, zero filled on the right.
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Input #0
Examples	<pre>OUT := SHL(IN := 16#1000CAFE, 16); (* OUT = 16#CAFE0000 *)</pre>

SHR	
Description	Input #0 right-shifted of Input #1 bits, zero filled on the left.
Number of operands	2
Input data type	Any numerical type
Output data type	Same as Input #0
Examples	<pre>OUT := SHR(IN := 16#1000CAFE, 24); (* OUT = 16#00000010 *)</pre>

Comparison Operators

Comparison operators can be also used to compare strings if this feature is supported by the target device.

EQ	
Description	Equal to. Returns TRUE if Input #0 = Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any
Output data type	BOOL
Examples	<pre>OUT := EQ(TRUE, FALSE); (* OUT = FALSE *) OUT := EQ('AZ', 'ABC'); (* OUT = FALSE *)</pre>

GE	
Description	Greater than or equal to. Returns TRUE if Input #0 >= Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any but BOOL
Output data type	BOOL
Examples	<pre>OUT := GE(20, 20); (* OUT = TRUE *) OUT := GE('AZ', 'ABC'); (* OUT = FALSE *)</pre>

GT	
Description	Greater than. Returns TRUE if Input #0 > Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any but BOOL
Output data type	BOOL
Examples	<pre>OUT := GT(0, 20); (* OUT = FALSE *) OUT := GT('AZ', 'ABC'); (* OUT = TRUE *)</pre>

LE	
Description	Less than or equal to. Returns TRUE if Input #0 <= Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any but BOOL
Output data type	BOOL
Examples	<pre>OUT := LE(20, 20); (* OUT = TRUE *) OUT := LE('AZ', 'ABC'); (* OUT = FALSE *)</pre>

LT	
Description	Less than. Returns TRUE if Input #0 < Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any but BOOL
Output data type	BOOL
Examples	<pre>OUT := LT(0, 20); (* OUT = TRUE *) OUT := LT('AZ', 'ABC'); (* OUT = FALSE *)</pre>

NE	
Description	Not equal to. Returns TRUE if Input #0!= Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any
Output data type	BOOL
Examples	<pre>OUT := NE(TRUE, FALSE); (* OUT = TRUE *) OUT := NE('AZ', 'ABC'); (* OUT = TRUE *)</pre>

Conversion Functions

According to the IEC 61131-3 standard, type conversion functions shall have the form `*_TO_*`, where “*” is the type of the input variable, and “**” the type of the output variable (for example, `INT_TO_REAL`). EcoStruxure Machine Expert - HVAC provides a more convenient set of overloaded type conversion functions, relieving you to specify the input variable type.

TO_BOOL	
Description	Conversion to BOOL (boolean)
Number of operands	1
Input data type	Any numerical type
Output data type	BOOL
Examples	<pre>out := TO_BOOL(0); (* out = FALSE *) out := TO_BOOL(1); (* out = TRUE *) out := TO_BOOL(1000); (* out = TRUE *)</pre>

TO_BYTE	
Description	Conversion to BYTE (8-bit string)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	BYTE
Examples	<pre>out := TO_BYTE(-1); (* out = 16#FF *) out := TO_BYTE(16#100); (* out = 16#00 *)</pre>

TO_DINT	
Description	Conversion to DINT (32-bit signed integer)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	DINT
Examples	<pre>out := TO_DINT(10.0); (* out = 10 *) out := TO_DINT(16#FFFFFFFF); (* out = -1 *)</pre>

TO_DWORD	
Description	Conversion to DWORD (32-bit string)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	DWORD
Examples	<pre>out := TO_DWORD(10.0); (* out = 16#0000000A *) out := TO_DWORD(-1); (* out = 16#FFFFFFFF *)</pre>

TO_INT	
Description	Conversion to INT (16-bit signed integer)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	INT
Examples	<pre>out := TO_INT(-1000.0); (* out = -1000 *) out := TO_INT(16#8000); (* out = -32768 *)</pre>

TO_LREAL	
Description	Conversion to LREAL (64-bit floating point)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	LREAL
Examples	<pre>out := TO_LREAL(-1000); (* out = -1000.0 *) out := TO_LREAL(16#8000); (* out = -32768.0 *)</pre>

TO_REAL	
Description	Conversion to REAL (32-bit floating point)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	REAL
Examples	<pre>out := TO_REAL(-1000); (* out = -1000.0 *) out := TO_REAL(16#8000); (* out = -32768.0 *)</pre>

TO_SINT	
Description	Conversion to SINT (8-bit signed integer)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	SINT
Examples	<pre>out := TO_SINT(-1); (* out = -1 *) out := TO_SINT(16#100); (* out = 0 *)</pre>

TO_STRING	
Description	Conversion to STRING
Number of operands	1
Input data type	Any numerical type
Output data type	STRING
Examples	<pre>str := TO_STRING(10.0); (* str = '10,0' *) str := TO_STRING(-1); (* str = '-1' *)</pre>

TO_STRINGFORMAT	
Description	Conversion to STRING, with format specifier
Number of operands	2
Input data type	Any numerical type, STRING
Output data type	STRING
Examples	<code>str := TO_STRINGFORMAT(10, '%04d'); (* str = '0010' *)</code>

TO_UDINT	
Description	Conversion to UDINT (32-bit unsigned integer)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	UDINT
Examples	<code>out := TO_UDINT(10.0); (* out = 10 *)</code> <code>out := TO_UDINT(16#FFFFFFFF); (* out = 4294967295 *)</code>

TO_UINT	
Description	Conversion to UINT (16-bit unsigned integer)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	UINT
Examples	<code>out := TO_UINT(1000.0); (* out = 1000 *)</code> <code>out := TO_UINT(16#8000); (* out = 32768 *)</code>

TO_USINT	
Description	Conversion to USINT (8-bit unsigned integer)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	USINT
Examples	<code>out := TO_USINT(-1); (* out = 255 *)</code> <code>out := TO_USINT(16#100); (* out = 0 *)</code>

TO_WORD	
Description	Conversion to WORD (16-bit string)
Number of operands	1

TO_WORD	
Input data type	Any numerical type or STRING
Output data type	WORD
Examples	<pre>out := TO_WORD(1000.0); (* out = 16#03E8 *) out := TO_WORD(-32768); (* out = 16#8000 *)</pre>

Logic Functions

AND	
Description	Logical AND if both Input #0 and Input #1 are BOOL, otherwise bitwise AND.
Number of operands	2
Input data type	Any but STRING
Output data type	Same as Inputs
Examples	<pre>OUT := TRUE AND FALSE; (* OUT = FALSE *) OUT := 16#1234 AND 16#5678; (* OUT = 16#1230 *)</pre>

NOT	
Description	Logical NOT if Input is BOOL, otherwise bitwise NOT.
Number of operands	1
Input data type	Any but STRING
Output data type	Same as Inputs
Examples	<pre>OUT := NOT FALSE; (* OUT = TRUE *) OUT := NOT 16#1234; (* OUT = 16#EDCB *)</pre>

OR	
Description	Logical OR if both Input #0 and Input #1 are BOOL, otherwise bitwise OR.
Number of operands	2
Input data type	Any but STRING
Output data type	Same as Inputs
Examples	<pre>OUT := TRUE OR FALSE; (* OUT = TRUE *) OUT := 16#1234 OR 16#5678; (* OUT = 16#567C *)</pre>

XOR	
Description	Logical XOR if both Input #0 and Input #1 are BOOL, otherwise bitwise XOR.
Number of operands	2
Input data type	Any but STRING

XOR	
Output data type	Same as Inputs
Examples	<pre>OUT := TRUE OR FALSE; (* OUT = TRUE *) OUT := 16#1234 OR 16#5678; (* OUT = 16#444C *)</pre>

Selection Functions

LIMIT	
Description	Limits Input #0 to be equal or more than Input#1, and equal or less than Input #2.
Number of operands	3
Input data type	Any numerical type
Output data type	Same as Inputs
Examples	<pre>OUT := LIMIT(IN := 4, MN := 0, MX := 5); (* OUT = 4 *) OUT := LIMIT(IN := 88, MN := 0, MX := 5); (* OUT = 5 *) OUT := LIMIT(IN := -1, MN := 0, MX := 5); (* OUT = 0 *)</pre>

MAX	
Description	Maximum value selection
Number of operands	2...30
Input data type	Any numerical type
Output data type	Same as max Input
Examples	<pre>OUT := MAX(-8, 120, -1000); (* OUT = 120 *)</pre>

MIN	
Description	Minimum value selection
Number of operands	2...30
Input data type	Any numerical type
Output data type	Same as min Input
Examples	<pre>OUT := MIN(-8, 120, -1000); (* OUT = -1000 *)</pre>

MUX	
Description	Multiplexer. Selects one of N inputs depending on input K
Number of operands	3...30
Input data type	Any numerical type

MUX	
Output data type	Same as selected Input
Examples	<code>OUT := MUX(0, A, B, C); (* OUT = A *)</code>

SEL	
Description	Binary selection
Number of operands	3
Input data type	BOOL, Any, Any
Output data type	Same as selected Input
Examples	<code>OUT := SEL(G := FALSE, IN0 := X, IN1 := 5); (* OUT = X *)</code>

Standard Operators

ADR	
Description	Address of
Number of operands	1
Input data type	Any type
Output data type	Pointer to type
Examples	<code>ptr_x := ADR(x)</code>

IMOVE	
Description	Query interface
Number of operands	1
Input data type	Any interface type
Output data type	Any interface type
Examples	<code>intf1 ?= obj1;</code>

JMP	
Description	Jump (conditioned/negated)
Number of operands	0
Input data type	Input
Output data type	Label name
Examples	<code>JMP mylabel;</code>

MOVE	
Description	Move
Number of operands	1
Input data type	Any type
Output data type	-
Examples	<code>MOVE x, y;</code>

REF	
Description	Reference to
Number of operands	0
Input data type	Any type
Output data type	Reference to type
Examples	<code>ref_x = REF(x);</code>

RET	
Description	Return (conditioned/negated)
Number of operands	0
Input data type	-
Output data type	-
Examples	<code>RET;</code>

SIZEOF	
Description	Size of
Number of operands	1
Input data type	Any type
Output data type	Numeric output
Examples	<code>mysize := SIZEOF(myvar);</code>

String Functions

CONCAT	
Description	Character string concatenation
Number of operands	2
Input data type	STRING

CONCAT	
Output data type	STRING
Examples	OUT := CONCAT('AB', 'CD'); (* OUT = 'ABCD' *)

DELETE	
Description	Delete L characters of IN, beginning at the P-th character position
Number of operands	3
Input data type	STRING, UINT, UINT
Output data type	STRING
Examples	OUT := DELETE(IN := 'ABXYC', L := 2, P := 3); (* OUT = 'ABC' *)

FIND	
Description	Find the character position of the beginning of the first occurrence of IN2 in IN1. If no occurrence of IN2 is found, then OUT := 0.
Number of operands	2
Input data type	STRING
Output data type	UINT
Examples	OUT := FIND(IN1 := 'ABCBC', IN2 := 'BC'); (* OUT = 2 *)

INSERT	
Description	Insert IN2 into IN1 after the P-th character position
Number of operands	3
Input data type	STRING, STRING, UINT
Output data type	STRING
Examples	OUT := INSERT(IN1 := 'ABC', IN2 := 'XY', P := 2); (* OUT = 'ABXYC' *)

LEFT	
Description	Leftmost L characters of IN
Number of operands	2
Input data type	STRING, UINT
Output data type	STRING
Examples	OUT := LEFT(IN := 'ASTR', L := 3); (* OUT = 'AST' *)

LEN	
Description	String length function
Number of operands	1
Input data type	STRING
Output data type	UINT
Examples	OUT := LEN(IN := 'ASTR'); (* OUT = 4 *)

MID	
Description	L characters of IN, beginning at the P-th
Number of operands	3
Input data type	STRING, UINT, UINT
Output data type	STRING
Examples	OUT := MID(IN := 'ASTR', L := 2, P := 2); (* OUT = 'ST' *)

REPLACE	
Description	Replace L characters of IN1 by IN2, starting at the P-th character position
Number of operands	4
Input data type	STRING, STRING, UINT, UINT
Output data type	STRING
Examples	OUT := REPLACE(IN1 := 'ABCDE', IN2 := 'X', L := 2, P := 3); (* OUT = 'ABXE' *)

RIGHT	
Description	Rightmost L characters of IN
Number of operands	2
Input data type	STRING, UINT
Output data type	STRING
Examples	OUT := RIGHT(IN := 'ASTR', L := 3); (* OUT = 'STR' *)

Section 14.2

Instruction List (IL)

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	404
Syntax and Semantics	404
Standard Operators	406
Calling Functions and Function Blocks	407

Overview

Description

This section defines the semantics of the IL (Instruction List) language.

Syntax and Semantics

Syntax of IL Instructions

IL code is composed of a sequence of instructions. Each instruction begins on a new line and contains an operator with optional modifiers, and, if necessary for the particular operation, one or more operands separated by commas. Operands can be any of the data representations for literals and for variables.

The instruction can be preceded by an identifying label followed by a colon (:). Empty lines can be inserted between instructions.

Example:

```
START:
  LD %IX1 (* Push button *)
  ANDN %MX5.4 (* Not inhibited *)
  ST %QX2 (* Fan out *)
```

The elements making up each instruction are classified as follows:

Label	Operator [+ modifier]	Operand	Comment
START:	LD	%IX1	(* Push button *)
	ANDN	%MX5.4	(* Not inhibited *)
	ST	%QX2	(* Fan out *)

Semantics of IL Instructions

- Accumulator

Accumulator is a register that contains the value of the current result.

- Operators

Unless otherwise specified, the semantics of the operators is:

```
accumulator := accumulator OP operand
```

That is, the value of the accumulator is replaced by the result yielded by operation OP applied to the current value of the accumulator itself, with respect to the operand.

For instance, the instruction “AND %IX1” is interpreted as:

```
accumulator := accumulator AND %IX1
```

The instruction “GT %IW10” will have the boolean result `TRUE` if the current value of the accumulator is greater than the value of input word 10, and the boolean result `FALSE` otherwise:

```
accumulator := accumulator GT %IW10
```

- Modifiers

The modifier “N” indicates bitwise negation of the operand.

The modifier “C” indicates that the associated instruction can be performed only if the value of the currently evaluated result is boolean 1 (or boolean 0 if the operator is combined with the “N” modifier).

The left parenthesis modifier “(” indicates that evaluation of the operator must be deferred until a right parenthesis operator “)” is encountered. The form of a parenthesized sequence of instructions is presented below, referred to the instruction:

```
accumulator := accumulator AND (%MX1.3 OR %MX1.4)
```

Standard Operators

Description

Standard operators with their allowed modifiers and operands are as listed below:

Operator	Modifiers	Supported operand types: Acc_type, Op_type	Semantics
LD	N	Any, Any	Sets the accumulator equal to operand.
ST	N	Any, Any	Stores the accumulator into operand location.
S		BOOL, BOOL	Sets operand to TRUE if accumulator is TRUE.
R		BOOL, BOOL	Sets operand to FALSE if accumulator is TRUE.
AND	N, (	Any but REAL, Any but REAL	Logical or bitwise AND
OR	N, (	Any but REAL, Any but REAL	Logical or bitwise OR
XOR	N, (	Any but REAL, Any but REAL	Logical or bitwise XOR
NOT		Any but REAL	Logical or bitwise NOT
ADD	(	Any but BOOL	Addition
SUB	(	Any but BOOL	Subtraction
MUL	(	Any but BOOL	Multiplication
DIV	(	Any but BOOL	Division
MOD	(	Any but BOOL	Modulo-division
GT	(	Any but BOOL	Comparison:
GE	(	Any but BOOL	Comparison: =
EQ	(	Any but BOOL	Comparison: =
NE	(	Any but BOOL	Comparison:
LE	(	Any but BOOL	Comparison:
LT	(	Any but BOOL	Comparison:
JMP	C, N	Label	Jumps to label
CAL	C, N	FB instance name	Calls function block
RET	C, N		Returns from called program, function, or function block.
)			Evaluates deferred operation.

Calling Functions and Function Blocks

Calling Functions

Functions (as defined in the relevant section) are invoked by placing the function name in the operator field. This invocation takes the following form:

```
LD 1
MUX 5, var0, -6.5, 3.14
ST vRES
```

The first argument is not contained in the input list, but the accumulator is used as the first argument of the function. Additional arguments (starting with the second), if required, are given in the operand field, separated by commas, in the order of their declaration. For example, operator `MUX` in the previous table takes 5 operands, the first of which is loaded into the accumulator, whereas the remaining 4 arguments are orderly reported after the function name.

The following rules apply to function invocation:

- Assignments to `VAR_INPUT` arguments may be empty, constants, or variables.
- Execution of a function ends upon reaching a `RET` instruction or the physical end of the function. When this happens, the output variable of the function is copied into the accumulator.

Calling Function Blocks

Function blocks (as defined in the relevant section) can be invoked conditionally and unconditionally via the `CAL` operator. This invocation takes the following form:

```
LD A
ADD 5
ST INST5.IN1
LD 3.141592
ST INST5.IN2
CAL INST5
LD INST5.OUT1
ST vRES
LD INST5.OUT2
ST vVALID
```

This method of invocation is equivalent to a `CAL` with an argument list, which contains only one variable with the name of the FB instance.

Input arguments are passed to / output arguments are read from the FB instance through `ST` / `LD` operations performed on operands taking the following form:

```
FBInstanceName.IO_var
```

where

Keyword	Description
FBInstanceName	Name of the instance to be invoked.
IO_var	Input or output variable to be written / read.

Section 14.3

Function Block Diagram (FBD)

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	409
Representation of Lines and Blocks	409
Direction of Flow in Networks	410
Evaluation of Networks	410
Execution Control Elements	412

Overview

Description

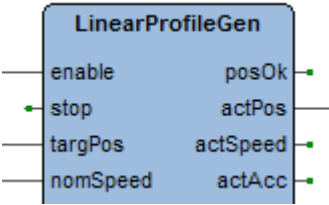
This section defines the semantics of the FBD (Function Block Diagram) language.

Representation of Lines and Blocks

Description

The graphic language elements are drawn using graphic or semi-graphic elements, as presented in the following table:

Feature	Example
Lines	
Line crossing with connection	

Feature	Example
Blocks with connecting lines and unconnected pins	 The diagram shows a blue rectangular block labeled "LinearProfileGen". It has six pins: "enable" (input, left), "stop" (input, left, with a green dot), "targPos" (input, left), "nomSpeed" (input, left), "posOk" (output, right, with a green dot), "actPos" (output, right), "actSpeed" (output, right, with a green dot), and "actAcc" (output, right, with a green dot). Lines connect the input pins to the block, and lines connect the output pins from the block.

No storage of data or association with data elements can be associated with the use of connectors; hence, to avoid ambiguity, connectors cannot be given any identifier.

Direction of Flow in Networks

Description

A network is defined as a maximal set of interconnected graphic elements. A network label delimited on the right by a colon (:) can be associated with each network or group of networks. The scope of a network and its label is local to the program organization unit (POU) where the network is located.

Graphic languages are used to represent the flow of a conceptual quantity through one or more networks representing a control plan. Namely, in the case of function block diagrams (FBD), the “Signal flow” is typically used, analogous to the flow of signals between elements of a signal processing system. Signal flow in the FBD language is from the output (right-hand) side of a function or function block to the input (left-hand) side of the function or function blocks so connected.

Evaluation of Networks

Order of Evaluation of Networks

The order in which networks and their elements are evaluated is not necessarily the same as the order in which they are labeled or displayed. When the body of a program organization unit (POU) consists of one or more networks, the results of network evaluation within the aforesaid body are functionally equivalent to the observance of the following rules:

- No element of a network is evaluated until the states of all of its inputs have been evaluated.
- The evaluation of a network element is not complete until the states of all of its outputs have been evaluated.
- As stated when describing the FBD editor, a network number is automatically assigned to every network. Within a program organization unit (POU), networks are evaluated according to the sequence of their number: network N is evaluated before network $N+1$, unless otherwise specified by using the execution control elements.

Combination of Elements

Elements of the FBD language must be interconnected by signal flow lines.

Outputs of blocks shall not be connected together. In particular, the “wired-OR” construct of the LD language is not allowed, as an explicit boolean “OR” block is required.

Feedback

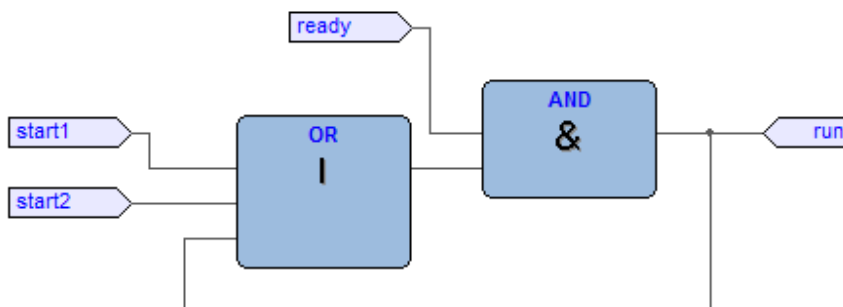
A feedback path is said to exist in a network when the output of a function or function block is used as the input to a function or function block which precedes it in the network; the associated variable is called a feedback variable.

Feedback paths can be utilized subject to the following rules:

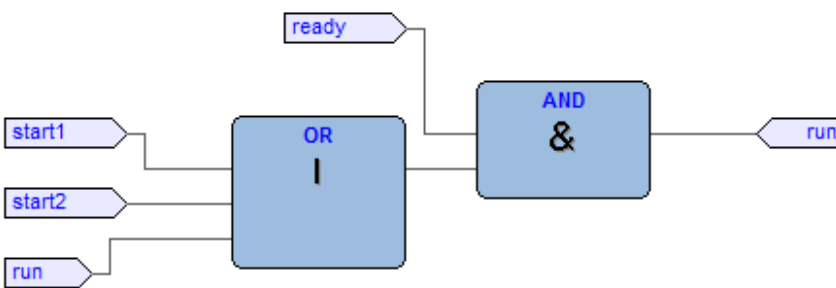
- Feedback variables must be initialized, and the initial value is used during the first evaluation of the network. Look at the global variables editor, the local variables editor, or the parameters editor to know how to initialize the respective item.
- Once the element with a feedback variable as output has been evaluated, the new value of the feedback variable is used until the next evaluation of the element.

For instance, the boolean variable `RUN` is the feedback variable in the following example:

Explicit loop:



Implicit loop:



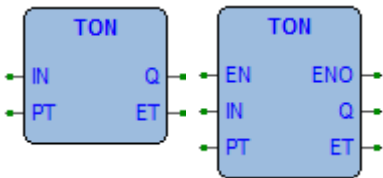
Execution Control Elements

EN/ENO Signals

Additional boolean **EN** (Enable) input and **ENO** (Enable Out) characterize EcoStruxure Machine Expert - HVAC blocks, according to the declarations:

EN	ENO
VAR_INPUT EN: BOOL := 1; END_VAR	VAR_OUTPUT ENO: BOOL; END_VAR

Refer to the Modifying properties of blocks section ([see page 235](#)) to know how to add these pins to a block:



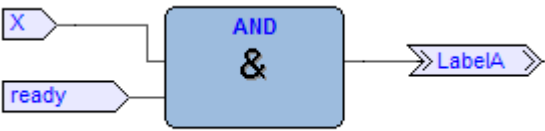
When these variables are used, the execution of the operations defined by the block are controlled according to the following rules:

- If the value of **EN** is **FALSE** when the block is invoked, the operations defined by the function body are not executed and the value of **ENO** is reset to **FALSE** by the programmable controller system.
- Otherwise, the value of **ENO** is set to **TRUE** by the programmable controller system, and the operations defined by the block body are executed.

Jumps

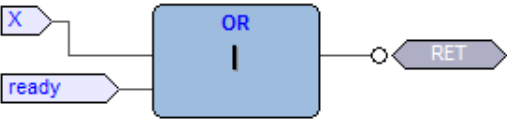
Jumps are represented by a boolean signal line terminated in a double arrowhead. The signal line for a jump condition originates at a boolean variable, or at a boolean output of a function or function block. A transfer of program control to the designated network label occurs when the boolean value of the signal line is **TRUE**; thus, the unconditional jump is a special case of the conditional jump.

The target of a jump is a network label within the program organization unit within which the jump occurs.

Symbol / Example	Explanation
	Unconditional Jump
	Conditional Jump
	Example: Jump Condition Network

Conditional Returns

- Conditional returns from functions and function blocks are implemented using a `RETURN` construction as presented in the following table. Program execution is transferred back to the invoking entity when the boolean input is `TRUE`, and continues in the normal fashion when the boolean input is `FALSE`.
- Unconditional returns are provided by the physical end of the function or function block.

Symbol / Example	Explanation
	Conditional Return
	Example: Return Condition Network

Section 14.4

Ladder Diagram (LD)

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	414
Power Rails	415
Link Elements and States	415
Contacts	416
Coils	417
Operators, Functions, and Function Blocks	418

Overview

Description

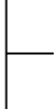
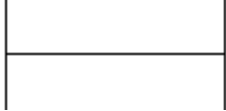
This section defines the semantics of the LD (Ladder Diagram) language.

Power Rails

Description

The LD network is delimited on the left side by a vertical line known as the left power rail, and on the right side by a vertical line known as the right power rail. The right power rail may be explicit in the EcoStruxure Machine Expert - HVAC implementation and it is always displayed.

The two power rails are always connected with a horizontal line named signal link. The LD elements must be placed and connected to the signal link.

Description	Symbol
Left power rail (with attached horizontal link)	
Right power rail (with attached horizontal link)	
Power rails connected by the signal link	

Link Elements and States

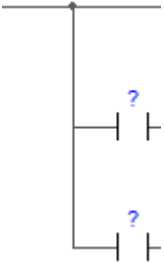
Description

Link elements may be horizontal or vertical. The state of the link elements shall be denoted “ON” or “OFF”, corresponding to the literal boolean values 1 or 0, respectively. The term link state shall be synonymous with the term power flow.

The following properties apply to the link elements:

- The state of the left rail shall be considered **ON** at all times. No state is defined for the right rail.
- A horizontal link element is indicated by a horizontal line. A horizontal link element transmits the state of the element on its immediate left to the element on its immediate right.
- The vertical link element consists of a vertical line intersecting with one or more horizontal link elements on each side. The state of the vertical link represents the inclusive **OR** of the **ON** states of the horizontal links on its left side, that is, the state of the vertical link is:
 - **OFF** if the states of all the attached horizontal links to its left are **OFF**.
 - **ON** if the state of one or more of the attached horizontal links to its left is **ON**.

- The state of the vertical link is copied to all of the attached horizontal links on its right.
- The state of the vertical link is not copied to any of the attached horizontal links on its left.

Description	Symbol
Vertical link with attached horizontal links	

Contacts

Description

A contact is an element which imparts a state to the horizontal link on its right side which is equal to the boolean AND of the state of the horizontal link at its left side with an appropriate function of an associated boolean input, output, or memory variable.

A contact does not modify the value of the associated boolean variable. Standard contact symbols are given in the following table:

Name	Description	Symbol
Normally open contact	The state of the left link is copied to the right link if the state of the associated boolean variable is ON. Otherwise, the state of the right link is OFF.	
Normally closed contact	The state of the left link is copied to the right link if the state of the associated boolean variable is OFF. Otherwise, the state of the right link is OFF.	
Positive transition-sensing contact	The state of the right link is ON from one evaluation of this element to the next when a transition of the associated variable from OFF to ON is sensed at the same time that the state of the left link is ON. The state of the right link shall be OFF at all other times.	
Negative transition-sensing contact	The state of the right link is ON from one evaluation of this element to the next when a transition of the associated variable from ON to OFF is sensed at the same time that the state of the left link is ON. The state of the right link shall be OFF at all other times.	

Coils

Description

A coil copies the state of the link on its left side to the link on its right side without modification, and stores an appropriate function of the state or transition of the left link into the associated boolean variable.

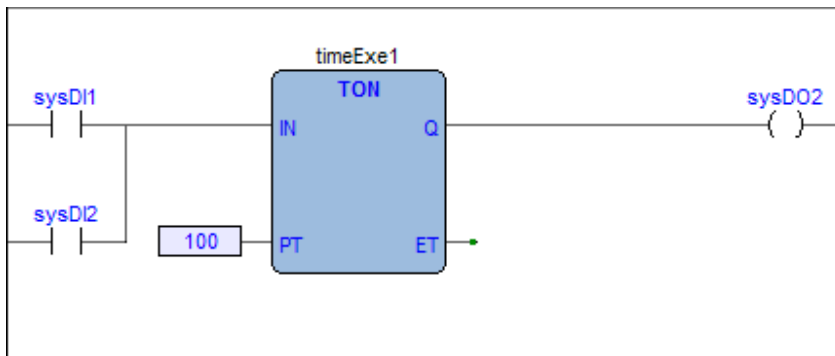
Standard coil symbols are presented in the following table:

Name	Description	Symbol
Coil	The state of the left link is copied to the associated boolean variable.	
Negated coil	The inverse of the state of the left link is copied to the associated boolean variable, that is, if the state of the left link is OFF, then the state of the associated variable is ON, and vice versa.	
SET (latch) coil	The associated boolean variable is set to the ON state when the left link is in the ON state, and remains set until reset by a RESET coil.	
RESET (unlatch) coil	The associated boolean variable is reset to the OFF state when the left link is in the ON state, and remains reset until set by a SET coil.	
Positive transition-sensing coil	The state of the associated boolean variable is ON from one evaluation of this element to the next when a transition of the left link from OFF to ON is sensed.	
Negative transition-sensing coil	The state of the associated boolean variable is ON from one evaluation of this element to the next when a transition of the left link from ON to OFF is sensed.	

Operators, Functions, and Function Blocks

Description

The representation of functions and function blocks in the LD language is similar to the one used for FBD. At least one boolean input and one boolean output shall be displayed on each block to allow for power flow through the block as presented in the following figure:



Section 14.5

Structured Text (ST)

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	419
Expressions	419
Statements in ST	421
Assignments	422
Function and Function Block Statements	423
Selection Statements	425
Iteration Statements	427

Overview

Description

This section defines the semantics of the ST (Structured Text) language.

Structured Text is a textual high-level programming language, similar to PASCAL or C. The program code is composed of expressions and instructions. In contrast to IL (Instruction List), you can use numerous constructions for programming loops, thus allowing the development of complex algorithms.

Expressions

Description

An expression is a construct which, when evaluated, yields a value corresponding to one of the data types listed in the elementary data types table (*see page 373*). EcoStruxure Machine Expert - HVAC does not set any constraint on the maximum length of expressions.

Expressions are composed of operators, operands, and/or assignments. An operand can be a constant, a variable, a function call, or another expression.

Operands

An operand can be a literal, a variable, a function invocation, or another expression.

Operators

Open the table of operators to see the list of all the operators supported by ST. The evaluation of an expression consists of applying the operators to the operands in a sequence defined by the operator precedence rules.

Operator Precedence Rules

Operators have different levels of precedence, as specified in the table of operators. The operator with highest precedence in an expression is applied first, followed by the operator of next lower precedence, and so on, until evaluation is complete. Operators of equal precedence are applied as written in the expression from left to right.

For example if A, B, C, and D are of type INT with values 1, 2, 3, and 4, respectively, then:

`A+B-C*ABS (D)`

yields -9, and:

`(A+B-C) *ABS (D)`

yields 0.

When an operator has two operands, the leftmost operand is evaluated first. For example, in the expression

`SIN (A) *COS (B)`

the expression `SIN (A)` is evaluated first, followed by `COS (B)`, followed by evaluation of the product.

Functions are invoked as elements of expressions consisting of the function name followed by a parenthesized list of arguments, as defined in the relevant section.

Operators of the ST Language

Operation	Symbol	Precedence
Parenthesization	(<expression>)	HIGHEST
Function evaluation	<fname>(<arglist>)	.
Negation Complement	- NOT	.
Exponentiation	**	.
Multiply Divide Modulo	* / MOD	.
Add Subtract	+ -	.
Comparison	<, >, <=, >=	.
Equality Inequality	= <>	.
Boolean AND	AND	LOWEST
Boolean Exclusive OR	XOR	
Boolean OR	OR	

Statements in ST

Description

All statements comply with the following rules:

- they are terminated by semicolons;
- unlike IL, a carriage return or new line character is treated the same as a space character;
- EcoStruxure Machine Expert - HVAC does not set any constraint on the maximum length of statements.

ST statements can be divided into classes, according to their semantics.

Assignments

Semantics

The assignment statement replaces the current value of a single or multi-element variable by the result of evaluating an expression.

The assignment statement is also used to assign the value to be returned by a function, by placing the function name to the left of an assignment operator in the body of the function declaration. The value returned by the function is the result of the most recent evaluation of such an assignment.

Syntax

An assignment statement consists of a variable reference on the left-hand side, followed by the assignment operator “:=”, followed by the expression to be evaluated. For instance, the statement

```
A := B ;
```

would be used to replace the single data value of variable A by the current value of variable B if both were of type `INT`.

Examples

Assignment:

```
a := b ;
```

Assignment:

```
pCV := pCV + 1 ;
```

Assignment with function invocation:

```
c := SIN( x ) ;
```

Assigning the output value to a function:

```
FUNCTION SIMPLE_FUN : REAL
  variables declaration
  ...
  function body
  ...
  SIMPLE_FUN := a * b - c ;
END_FUNCTION
```

Function and Function Block Statements

Semantics

- Functions are invoked as elements of expressions consisting of the function name followed by a parenthesized list of arguments. Each argument can be a literal, a variable, or an arbitrarily complex expression.
- Function blocks are invoked by a statement consisting of the name of the function block instance followed by a parenthesized list of arguments. Both invocation with formal argument list and with assignment of arguments are supported.
- RETURN: function and function block control statements consist of the mechanisms for invoking function blocks and for returning control to the invoking entity before the physical end of a function or function block. The RETURN statement provides early exit from a function or a function block (for example, as the result of the evaluation of an IF statement).

Syntax

Function:

```
dst_var := function_name( arg1, arg2 , ... , argN );
```

Function block with formal argument list:

```
instance_name(var_in1 := arg1 ,
              var_in2 := arg2 ,
              ... ,
              var_inN := argN );
```

Function block with assignment of arguments:

```
instance_name.var_in1 := arg1;
...
instance_name.var_inN := argN;
instance_name();
```

Function and function block control statement:

```
RETURN;
```

Examples

FB invocation with formal argument list:

```
CMD_TMR( IN := %IX5, PT:= 300 ) ;
```

FB invocation with assignment of arguments:

```
IN := %IX5 ;  
PT:= 300 ;  
CMD_TMR() ;
```

FB output usage:

```
a := CMD_TMR.Q;
```

Early exit from function or function block:

```
RETURN ;
```


Selection Statements

Semantics

Selection statements include the `IF` and `CASE` statements. A selection statement selects one (or a group) of its component statements for execution based on a specified condition.

- **IF:** the `IF` statement specifies that a group of statements is to be executed only if the associated boolean expression evaluates to the value `TRUE`. If the condition is false, then either no statement is to be executed, or the statement group following the `ELSE` keyword (or the `ELSIF` keyword if its associated boolean condition is true) is executed.
- **CASE:** the `CASE` statement consists of an expression which evaluates to a variable of type `DINT` (the “selector”), and a list of statement groups, each group being labeled by one or more integer or ranges of integer values, as applicable. It specifies that the first group of statements, one of whose ranges contains the computed value of the selector, is to be executed. If the value of the selector does not occur in a range of any case, the statement sequence following the keyword `ELSE` (if it occurs in the `CASE` statement) is executed. Otherwise, none of the statement sequences is executed.

EcoStruxure Machine Expert - HVAC does not set any constraint on the maximum allowed number of selections in `CASE` statements.

Syntax

Square brackets include optional code while braces include repeatable portions of code.

IF:

```
IF expression1 THEN
    stat_list
[ { ELSIF expression2 THEN
    stat_list } ]
ELSE
    stat_list
END_IF ;
```

CASE:

```
CASE expression1 OF
    intv [ {, intv } ] :
        stat_list
    { intv [ {, intv } ] :
        stat_list }
[ ELSE
    stat_list ]
END_CASE ;
```

`intv` being either a constant or an interval: `a` or `a..b`

Examples

IF statement:

```
IF d < 0.0 THEN
    nRoots := 0 ;
ELSIF d = 0.0 THEN
    nRoots := 1 ;
    x1 := -b / (2.0 * a) ;
ELSE
    nRoots := 2 ;
    x1 := (-b + SQRT(d)) / (2.0 * a) ;
    x2 := (-b - SQRT(d)) / (2.0 * a) ;
END_IF ;
```

CASE statement:

```
CASE tw OF
    1, 5:
        display := oven_temp ;
    2:
        display := motor_speed ;
    3:
        display := gross_tare;
    4, 6..10:
        display := status(tw - 4) ;
ELSE
    display := 0;
    tw_error := 1;
END_CASE ;
```

Iteration Statements

Semantics

Iteration statements specify that the group of associated statements are executed repeatedly. The `FOR` statement is used if the number of iterations can be determined in advance; otherwise, the `WHILE` or `REPEAT` constructs are used.

- **FOR:** the `FOR` statement indicates that a statement sequence is repeatedly executed, up to the `END_FOR` keyword, while a progression of values is assigned to the `FOR` loop control variable. The control variable, initial value, and final value are expressions of the same integer type (for example, `SINT`, `INT`, or `DINT`) and cannot be altered by any of the repeated statements. The `FOR` statement increments the control variable up or down from an initial value to a final value in increments determined by the value of an expression. The default increment value is 1. The test for the termination condition is made at the beginning of each iteration so that the statement sequence is not executed if the initial value exceeds the final value.
- **WHILE:** the `WHILE` statement causes the sequence of statements up to the `END_WHILE` keyword to be executed repeatedly until the associated boolean expression is false. If the expression is initially false, then the group of statements is not executed at all.
- **REPEAT:** the `REPEAT` statement causes the sequence of statements up to the `UNTIL` keyword to be executed repeatedly (and at least once) until the associated boolean condition is true.
- **EXIT:** the `EXIT` statement is used to terminate iterations before the termination condition is satisfied. When the `EXIT` statement is located within nested iterative constructs, `exit` is from the innermost loop in which the `EXIT` is located, that is, control passes to the next statement after the first loop terminator (`END_FOR`, `END_WHILE`, or `END_REPEAT`) following the `EXIT` statement.

NOTE: The `WHILE` and `REPEAT` statements cannot be used to achieve interprocess synchronization, for example as a “wait loop” with an externally determined termination condition. The SFC elements defined must be used for this purpose.

Syntax

Square brackets include optional code while braces include repeatable portions of code. If the terminating condition is not correct, it can cause an endless loop.

NOTICE**UNINTENDED EQUIPMENT OPERATION**

- Ensure that the variable used in FOR instructions is of a sufficient capacity (has a great upper limit) to account for the <END_VALUE> + 1.
- Ensure that the WHILE loop will be terminated within the instructions of the loop by creating a FALSE condition of the boolean expression.
- Ensure that the REPEAT loop will be terminated within the instructions of the loop by creating a TRUE condition of the boolean expression.

Failure to follow these instructions can result in equipment damage.

FOR:

```
FOR control_var := init_val TO end_val [ BY increm_val ] DO
    stat_list
END_FOR ;
```

WHILE:

```
WHILE expression DO
    stat_list
END_WHILE ;
```

REPEAT:

```
REPEAT
    stat_list
UNTIL expression
END_REPEAT ;
```

Examples

FOR statement:

```
j := 101 ;
FOR i := 1 TO 100 BY 2 DO
    IF arrvals[i] = 57 THEN
        j := i ;
        EXIT ;
    END_IF ;
END_FOR ;
```

WHILE statement:

```
j := 1 ;
WHILE j <=100 AND arrvals[i] <> 57 DO
    j := j + 2 ;
END_WHILE ;
```

REPEAT statement:

```
j := -1 ;
REPEAT
    j := j + 2 ;
    UNTIL j = 101 AND arrvals[i] = 57
END_REPEAT ;
```

Section 14.6

Sequential Function Chart (SFC)

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	430
Steps	431
Transitions	433
Rules of Evolution	434
SFC Control Flags	439
Check an SFC POU from Other Programs	440

Overview

Description

This section defines Sequential Function Chart (SFC) elements to structure the internal organization of a PLC program organization unit (POU), written in one of the languages defined in this standard, for performing sequential control functions. The definitions in this section are derived from IEC 848, with the necessary changes to convert the representations from a standard documentation to a set of execution control elements for a PLC program organization unit.

Since SFC elements require storage of state information, the only program organization units which can be structured using these elements are function blocks and programs.

If any part of a program organization unit is partitioned into SFC elements, the entire program organization unit is so partitioned. If no SFC partitioning is given for a program organization unit, the entire program organization unit is considered to be a single action which executes under the control of the invoking entity.

SFC elements

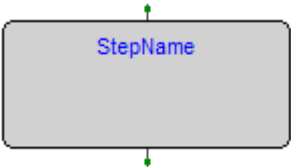
The SFC elements provide a means of partitioning a PLC program organization unit into a set of steps and transitions interconnected by directed links. Associated with each step is a set of actions, and with each transition is associated a transition condition.

Steps

Definition

A step represents a situation where the behavior of a program organization unit (POU) with respect to its inputs and outputs follows a set of rules defined by the associated actions of the step. A step is either active or inactive. At any given moment, the state of the program organization unit is defined by the set of active steps and the values of its internal and output variables.

A step is represented graphically by a block containing a step name in the form of an identifier. The directed links into the step can be represented graphically by a vertical line attached to the top of the step. The directed links out of the step can be represented by a vertical line attached to the bottom of the step.

Representation	Description
	Step (graphical representation with direct links)

EcoStruxure Machine Expert - HVAC does not set any constraint on the maximum number of steps per SFC within the bounds of the available memory.

Step flag

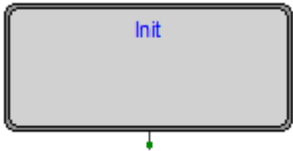
The step flag (active or inactive state of a step) can be represented by the logic value of a boolean variable `***_x`, where `***` is the step name. This boolean variable has the value `TRUE` when the corresponding step is active, and `FALSE` when it is inactive. The scope of step names and step flags is local to the program organization unit where the steps appear.

Representation	Description
<code>Step Name_x</code>	Step flag = <code>TRUE</code> when <code>Step Name_x</code> is active = <code>FALSE</code> otherwise

Initial Step

The initial state of the program organization unit is represented by the initial values of its internal and output variables, and by its set of initial steps, that is, the steps which are initially active. Each SFC network, or its textual equivalent, has exactly one initial step. An initial step can be drawn graphically with double lines for the borders, as presented below. For system initialization, the default initial state is FALSE for ordinary steps and TRUE for initial steps.

EcoStruxure Machine Expert - HVAC cannot compile an SFC network not containing exactly one initial step.

Representation	Description
	Initial step (graphical representation with direct links)

Actions

An action can be:

- A collection of instructions in the IL language;
- A collection of networks in the FBD language;
- A collection of rungs in the LD language;
- A collection of statements in the ST language;
- A sequential function chart (SFC) organized as defined in this section.

Zero or more actions can be associated with each step. Actions are declared via one of the textual structuring elements listed in the following table:

Structuring element	Description
STEP StepName : (* Step body *) END_STEP	Step (textual form)
INITIAL_STEP StepName : (* Step body *) END_STEP	Initial step (textual form)

Such a structuring element exists in the **lsc** file for every step having at least one associated action.

Action Qualifiers

The time when an action associated to a step is executed depends on its action qualifier.

EcoStruxure Machine Expert - HVAC implements the following action qualifiers:

Qualifier	Description	Meaning
N	Non-stored (null qualifier)	The action is executed as long as the step remains active.
P	Pulse	The action is executed only once per step activation, regardless of the number of cycles the step remains active.

If a step has zero associated actions, then it is considered as having a **WAIT** function, that is, waiting for a successor transition condition to become true.

Jumps

Direct links flow only downwards. To return to an upper step from a lower one, you cannot draw a logical wire from the latter to the former. A special type of block exists, called Jump, which lets you implement such a transition.

A Jump block is logically equivalent to a step, as they have to always be separated by a transition. The only effect of a Jump is to activate the step flag of the preceding step and to activate the flag of the step it points to.

Representation	Description
	Jump (logical link to the destination step)

Transitions

Definition

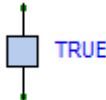
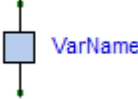
A transition represents the condition whereby control passes from one or more steps preceding the transition to one or more successor steps along the corresponding directed link. The transition is represented by a small gray square across the vertical directed link.

The direction of evolution following the directed links is from the bottom of the predecessor steps to the top of the successor steps.

Transition Condition

Each transition has an associated transition condition which is the result of the evaluation of a single boolean expression. A transition condition which is always true is represented by the keyword `TRUE`, whereas a transition condition always false is symbolized by the keyword `FALSE`.

A transition condition can be associated with a transition by one of the following means:

Representation	Description
	By placing the appropriate boolean constant { <code>TRUE</code> , <code>FALSE</code> } adjacent to the vertical directed link.
	By declaring a boolean variable, whose value determines whether the transition is cleared.
	By writing a piece of code, in any of the languages supported by EcoStruxure Machine Expert - HVAC, except for SFC. The result of the evaluation of such a code determines the transition condition.

The scope of a transition name is local to the program organization unit (POU) where the transition is located.

Rules of Evolution

Introduction

The initial situation of an SFC network is characterized by the initial step which is in the active state upon initialization of the program or function block containing the network.

Evolutions of the active states of steps take place along the directed links when caused by the clearing of one or more transitions.

A transition is enabled when all the preceding steps, connected to the corresponding transition symbol by directed links, are active. The clearing of a transition occurs when the transition is enabled and when the associated transition condition is true.

The clearing of a transition causes the deactivation (or “resetting”) of all the immediately preceding steps connected to the corresponding transition symbol by directed links, followed by the activation of all the immediately following steps.

The alternation Step/Transition and Transition/Step are always maintained in SFC element connections, that is:

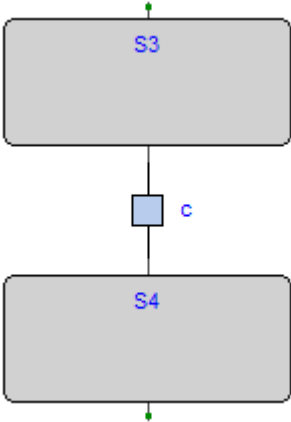
- Two steps are never directly linked; they are always separated by a transition;
- Two transitions are never directly linked; they are always separated by a step.

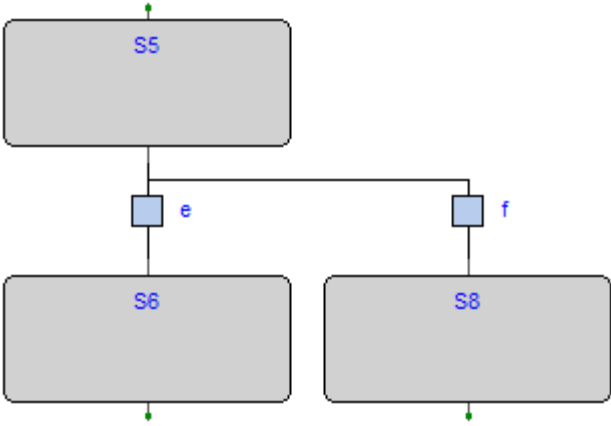
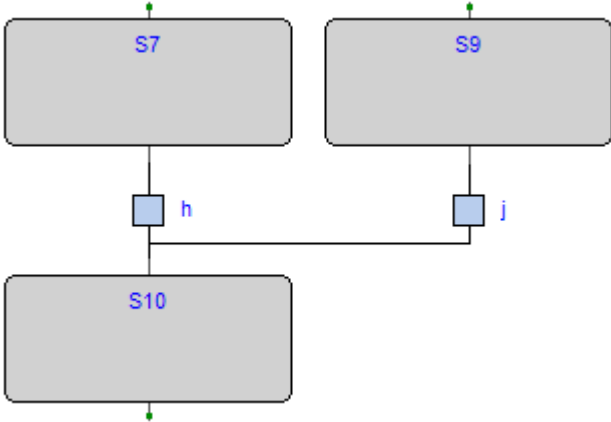
When the clearing of a transition leads to the activation of several steps at the same time, the sequences which these steps belong to are called simultaneous sequences. After their simultaneous activation, the evolution of each of these sequences becomes independent. In order to emphasize the special nature of such constructs, the divergence and convergence of simultaneous sequences is indicated by a double horizontal line.

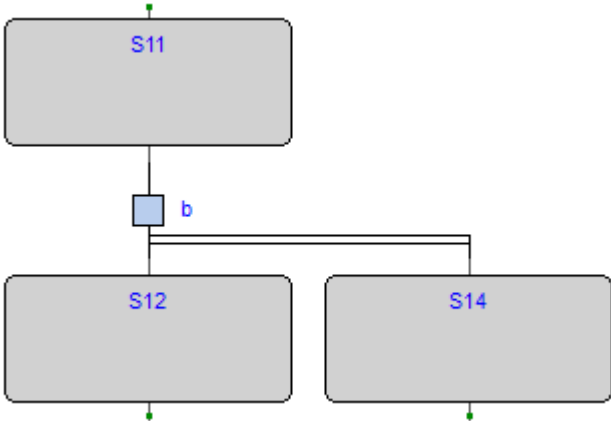
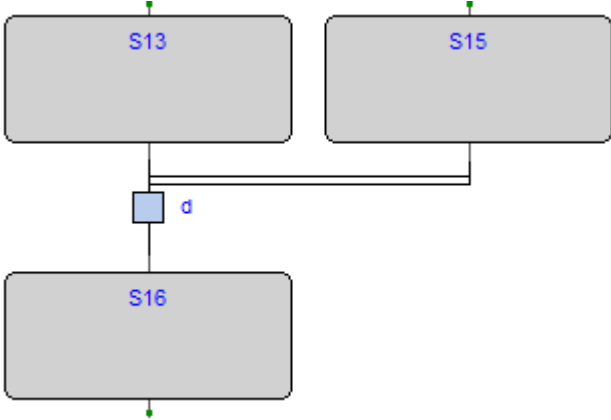
The clearing time of a transition may theoretically be considered as short as one may wish, but it can never be zero. In practice, the clearing time is imposed by the PLC implementation: several transitions which can be cleared simultaneously are cleared simultaneously, within the timing constraints of the particular PLC implementation and the priority constraints defined in the sequence evolution table. For the same reason, the duration of a step activity can never be considered to be zero. Testing of the successor transition conditions of an active step shall not be performed until the effects of the step activation have propagated throughout the program organization unit where the step is declared.

Sequence Evolution Table

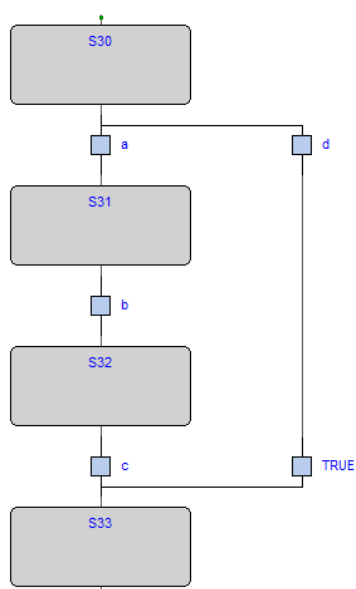
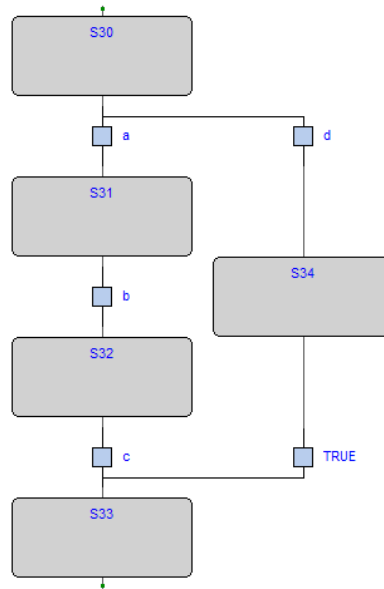
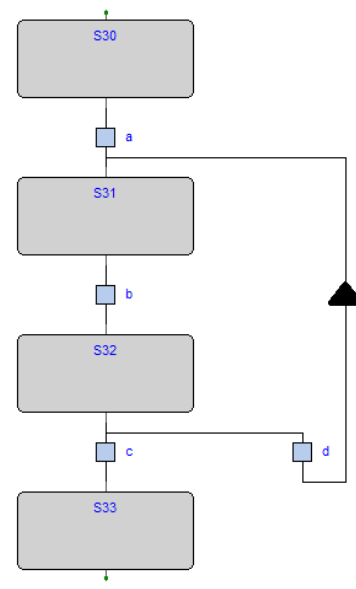
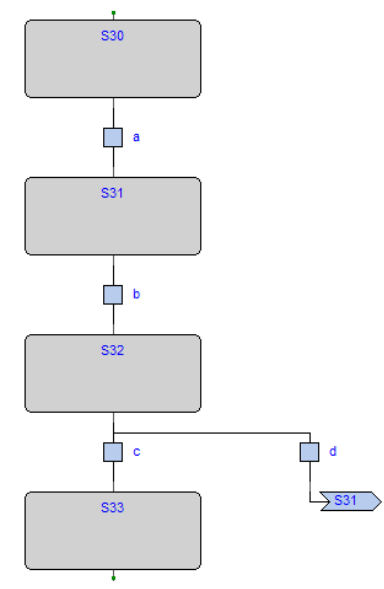
This table defines the syntax and semantics of the allowed combinations of steps and transitions.

Example	Rule
	Normal transition: An evolution from step S3 to step S4 takes place if and only if step S3 is in the active state and the transition condition c is TRUE.

Example	Rule
 The diagram shows a state S5 at the top. Below it are two states, S6 and S8. A transition line from S5 splits into two branches. The left branch passes through a small blue square labeled 'e' before reaching S6. The right branch passes through a small blue square labeled 'f' before reaching S8. Each state (S5, S6, S8) has a green arrow pointing into it from above and a green arrow pointing out of it from below.	Divergent transition: An evolution takes place from S5 to S6 if and only if S5 is active and the transition condition e is TRUE, or from S5 to S8 only if S5 is active and f is TRUE and e is FALSE.
 The diagram shows two states at the top, S7 and S9. Below them is a single state S10. A transition line from S7 passes through a small blue square labeled 'h' and then joins a transition line coming from S9. The transition from S9 passes through a small blue square labeled 'j' before joining the line from S7. The combined line then leads to S10. Each state (S7, S9, S10) has a green arrow pointing into it from above and a green arrow pointing out of it from below.	Convergent transition: An evolution takes place from S7 to S10 only if S7 is active and the transition condition h is TRUE, or from S9 to S10 only if S9 is active and j is TRUE.

Example	Rule
	Simultaneous divergent transition: An evolution takes place from S11 to S12, S14,... only if S11 is active and the transition condition b associated to the common transition is TRUE. After the simultaneous activation of S12, S14, and so on, the evolution of each sequence proceeds independently.
	Simultaneous convergent transition: An evolution takes place from S13, S15,... to S16 only if all steps above and connected to the double horizontal line are active and the transition condition d associated to the common transition is TRUE.

Examples

Invalid scheme	Equivalent allowed scheme	Note
		Expected behavior: an evolution takes place from S30 to S33 if a is FALSE and d is TRUE. The scheme in the leftmost column is invalid because conditions d and TRUE are directly linked.
		Expected behavior: an evolution takes place from S32 to S31 if c is FALSE and d is TRUE. The scheme in the leftmost column is invalid because direct links flow only downwards. Upward transitions can be performed via jump blocks.

SFC Control Flags

Description

EcoStruxure Machine Expert - HVAC provides some control flags for SFC program or function blocks.

To enable this feature, refer to paragraph Code generation (*see page 166*).

Those flags are:

- `<POU name>_HOLD_SFC (type BOOL) ;`
- `<POU name>_RESET_SFC (type BOOL) .`

Where `<POU name>` means the name of the SFC POU (program or function block).

For example, if the SFC POU is named `Main`, the control flags are named `Main_HOLD_SFC` and `Main_RESET_SFC`.

Another couple of actions is available for every SFC action, which also are contained in an SFC POU.

For example, if the program `Main` contains an SFC action named **Execute**, the control flags of this action are `Main_Execute_HOLD_SFC` and `Main_Execute_RESET_SFC`.

Hold Flag

Following the main characteristics of the `<POU name>_HOLD_SFC` flag:

- the default value is `FALSE`;
- When set to `TRUE`, the SFC block, which is referred to (the one with the same name as `<POU name>`), it is kept in the current status (hold) and no code is executed;
- When the flag is set back to `FALSE`, the SFC block execution is recovered from exactly the same point in which was set to hold, through `<POU name>_HOLD_SFC := TRUE`.

Reset Flag

Following the main characteristics of the `<POU name>_RESET_SFC` flag:

- The default value is `FALSE`;
- When set to `TRUE`, the SFC block, which is referred to (the one with the same name as `<POU name>`), it is brought back to the initial state, that is the execution state of the init action.
- This is an auto-reset flag, which means that if it is set to `TRUE` its own state becomes `FALSE` after its reset action has been executed. It is not necessary to bring the `<POU name>_RESET_SFC` value back to `FALSE`.

Flags Visibility

The `<POU name>_HOLD_SFC` and `<POU name>_RESET_SFC` flags are automatically generated from the code compiler and they belong to the local variables of the POU which are referred to.

EcoStruxure Machine Expert - HVAC does not show this flags in the variables list of the POU; they are hidden but in any case they can be used everywhere within the code.

Check an SFC POU from Other Programs

Description

To allow the managing of an SFC POU from other programs, EcoStruxure Machine Expert - HVAC provides the following functionalities:

- The compiler automatically generates the `<POU name>_RESET_SFC` and `<POU name>_HOLD_SFC` flags.
- If the SFC POU is a function block, the user has the possibility to declare, as `VAR_INPUT` and type `BOOL`, both flags having the name of the SFC POU control flags.
- If the SFC POU is a program, the user has the possibility to declare, as `VAR_GLOBAL` and type `BOOL`, both flags having the name of the SFC POU control flags.
- In both previous cases, EcoStruxure Machine Expert - HVAC compiler uses the variables declared among the `VAR_INPUT` or `VAR_GLOBAL` ones and not those automatically generated (therefore they are not generated).

Using these techniques, user then can manage the working state of the SFC POU from other POU using the `INPUT` variables of the SFC POU.

Example

```
FUNCTION_BLOCK test
  VAR_INPUT
    ...
    test_RESET_SFC : BOOL; (* Control flag explicitly declared *)
  END_VAR
  ...
END_FUNCTION_BLOCK

PROGRAM Main
  VAR
    ...
    block : test; (* SFC block instance *)
  END_VAR
  ...
  (* Reset SFC block state *)
  block.test_RESET_SFC := TRUE;
  ...
END_PROGRAM
```


SFC Macro Library

EcoStruxure Machine Expert - HVAC makes available a library called **SFCCControl.pll** to allow you to manage the SFC states through commands instead of variable settings.

This library is composed by macros usable only in ST language.

Usage Example of the Control Flags

Following are some example of control flags usage, assuming the SFC POU is named Main:

- Hold (freeze):
`Main_HOLD_SFC := TRUE;`
- Restart from hold state:
`Main_HOLD_SFC := FALSE;`
- Restart from initial state of an SFC block in hold state:
`Main_RESET_SFC := TRUE;`
`Main_HOLD_SFC := FALSE;`
- Reset to initial state and instant restart of SFC block:
`Main_RESET_SFC := TRUE; (* automatic reset from compiler *)`

Section 14.7

EcoStruxure Machine Expert - HVAC Language Extensions

What Is in This Section?

This section contains the following topics:

Topic	Page
Overview	442
Macros	442
Pointers	443
Waiting Statement	444

Overview

Description

EcoStruxure Machine Expert - HVAC features a few extensions to the IEC 61131-3 standard in order to further enrich the language and to adapt to different coding styles.

Macros

Description

EcoStruxure Machine Expert - HVAC implements macros in the same way a C programming language pre-processor does.

Macros can be defined using the following syntax:

```
MACRO <macro name>
  PAR_MACRO
    <parameter list>
  END_PAR
  <macro body>
END_MACRO
```

The parameter list may be empty, thus distinguishing between object-like macros, which do not take parameters, and function-like macros, which take parameters.

A concrete example of macro definition is the following, which takes two bytes and composes a 16-bit word:

```
MACRO MAKEWORD
  PAR_MACRO
    lobyte;
    hibyte;
  END_PAR
  { CODE:ST }
  lobyte + SHL( TO_UINT( hibyte ), 8 )
END_MACRO
```

Whenever the macro name appears in the source code, it is replaced (along with the current parameter list, in case of function-like macros) with the macro body. For example, given the definition of the macro `MAKEWORD` and the following Structured Text code fragment:

```
w := MAKEWORD( b1, b2 );
```

the macro pre-processor expands it to

```
w := b1 + SHL( TO_UINT( b2 ), 8 );
```

Pointers

Description

Pointers are special variables which act as a reference to another variable (the pointed variable). The value of a pointer is, in fact, the address of the pointed variable; in order to access the data stored at the address pointed to, pointers can be dereferenced.

Pointer declaration requires the same syntax used in variable declaration, where the type name is the type name of the pointed variable preceded by a `@` sign:

```
VAR
  <pointer name> : @<pointed variable type name>;
END_VAR
```

For example, the declaration of a pointer to a REAL variable shall be as follows:

```
VAR
  px : @REAL;
END_VAR
```

A pointer can be assigned with another pointer or with an address. A special operator, `ADR`, is available to retrieve the address of a variable.

```
px := py; (* px and py are pointers to REAL (that is, variables of type @REAL) *)
px := ADR( x ) (* x is a variable of type REAL *)
px := ?x (* ? is an alternative notation for ADR *)
```

The @ operator is used to dereference a pointer, hence to access the pointed variable.

```
px := ADR( x );
@px := 3.141592; (* the approximate value of pi is assigned to x *)
pn := ADR( n );
n := @pn + 1; (* n is incremented by 1 *)
```

Be aware that careless use of pointers is a potential source of serious programming errors that, in a runtime environment, can have an undesirable effect on the state of the controller and/or your machine or process.

CAUTION

INVALID POINTER

- Verify the validity of the pointers when using pointers on addresses before they are applied.
- Always initialize pointers with a valid memory address.

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

Waiting Statement

Description

EcoStruxure Machine Expert - HVAC implements a **WAITING** statement that can be used in ST code as following example:

```
...
WAITING <condition> DO
    <code to be executed waiting for condition becomes true>
END_WAITING;
...
```

Until the condition is not verified, the code is executed (not as in a loop cycle but returning to caller in every execution).

The **WAITING** statement can be used only if the associated project option is enabled. For more details, refer to Code Generation ([see page 166](#)).

Chapter 15

Errors Reference

Section 15.1

Compile Time Error Messages

Overview

Description

Error code	Short description	Explanation
A4097	Object not found	The object indicated (variable or function block) has not been defined in the application.
A4098	Unsupported data type	The size (in bits) requested by the indicated data type is not supported by the target system.
A4099	Auto vars space exhausted	The total allocation space requested by all local variables exceeds the space available on the target system.
A4100	Retentive vars space exhausted	The total allocation space requested by all local retentive variables exceeds the space available on the target system.
A4101	Bit vars space exhausted	The total allocation space requested by all local bit (boolean) variables exceeds the space available on the target system.
A4102	Invalid ++ in data block	The variable indicated is associated with an index that is not available in the relative data block.
A4103	Data block not found	The variable indicated is associated with a data block that does not exist (is not defined) in the target system.
A4104	Code space exhausted	The total size of code used for POU (programs, functions, and function blocks) exceed the space available on the target system.
A4105	Invalid bit offset	The variable indicated is associated with a bit index that is not available in the relative data block.
A4106	Image variable requested	Error code superseded.
A4107	Target function not found	The function indicated is not available on the target system.
A4108	Base object not found	The indicated instance refers to a function block definition non defined.
A4109	Invalid base object type	The indicated variable is associated with a data type (including function block definition) that is not defined.
A4110	Invalid data type	The data type used in the variable definition does not exist.
A4111	Invalid operand type	The operand type is not allowed for the current operator.

Error code	Short description	Explanation
A4112	Function block shares global data and is used by more tasks	The indicated function block is called by more than one task but uses global variables with process image. For this reason, the compiler is not able to refer to the proper image variable for each instance of the function block.
A4113	Temporary variables allocation error	Internal compiler error.
A4114	Embedded functions do not support arrays as input variables	-
A4115	Too many parameters input to embedded function	-
A4116	Incremental build failed, perform a full build command	-
A4117	Less than 10% of free data	-
A4118	Less than 10% of free retain data	-
A4119	Less than 10% of free bit data	-
A4120	Variable exceeds data block space	-
A4121	Element not found	-
A4123	Invalid access to private member	-
A4129	Not a structured type	-
A4130	Not a function block instance	-
A4131	Incompatible external declaration	-
A4133	Not a variable	-
A4134	Index exceeds array size	-
A4135	Invalid index data type	-
A4136	Missing index(es)	-
A4137	Function block instance required	-
A4138	Simple variable required	-
A4139	Too many indexes	-
A4140	Not a structure instance	-
A4141	Not an array	-
A4143	Not a pointer	-
A4144	Double pointer indirection not allowed	-
A4145	To be implemented	-
A4146	Bit datatype not allowed	-
A4147	Unable to calculate variable offset	-
A4148	Complex variables cannot have process image	-

Error code	Short description	Explanation
A4149	Cannot use directly represented variables with process image in function blocks (not implemented)	-
A4150	Function block instance not allowed	-
A4151	Structure not allowed	-
A4152	16-bit variables must be aligned to a 16-bit boundary	-
A4153	32-bit variables must be aligned to a 32-bit boundary	-
A4154	Temporary string variable allocation error. Instruction shall be split.	-
A4155	Ext/aux auto vars space exhausted	-
A4156	Ambiguous enum value, <enum># prefix required	-
B0001	Data block not found	The variable indicated is associated with a data block that does not exist (is not defined) in the target system.
B0002	Error on create file	The indicated file cannot be created due to a file system error or to a missing source file.
C0001	Parser not initialized	Internal compiler error.
C0002	Invalid token	Invalid word for the current language syntax
C0003	Invalid file specification	Internal compiler error.
C0004	Cannot open file	The indicated file cannot be opened due to a file system error or to a missing source file.
C0005	Parser table error	Internal compiler error.
C0006	Parser non specified	Internal compiler error.
C0007	Unexpected end of file	The indicated file is truncated or the syntax is incomplete.
C0009	Reserved keyword	The indicated word cannot be used for declaration purposes because is a keyword of the language.
C0010	Invalid element	The indicated word is not a valid one for the language syntax.
C0011	Aborted by user	-
C0032	Too many parameters in macro call	-
C0033	Invalid number of parameters in macro call	-
C0034	Too many macro calls nested	-
C4097	Invalid variable type	The data type indicated is not allowed.
C4098	Invalid location prefix	The address string of the indicated variable is not correct, '%' missing.

Error code	Short description	Explanation
C4099	Invalid location specification	The address string of the indicated variable is not correct, the data access type indication isn't 'I', 'Q' or 'M'.
C4100	Invalid location type	The address string of the indicated variable is not correct, the data type indication isn't 'X', 'B', 'W', 'D', 'R' or 'L'.
C4101	Invalid location index specification	The address string of the indicated variable is not correct, the index is not correct.
C4102	Duplicate variable name	The name of the indicated variable has already been used for some other project object.
C4103	Only 0 admitted here	The compiler uses only arrays zero-index based
C4104	Invalid array dimension	The dimension of the array is not indicated in the correct way (for example: contains invalid characters, negative numbers and so on).
C4105	Constant not initialized	Every constant need to have an initial value.
C4106	Invalid string size	-
C4107	Initialization exceeding string size	-
C4108	Invalid repetition in initialization	-
C4109	Invalid data type for initialization	-
C4353	Duplicate label	The indicated label has already been defined in the current POU (program, function, or function block).
C4354	Constant not admitted	The operation indicated does not allow to use constants (typically store or assign operations).
C4355	Address of explicit constant not defined	-
C4356	Maximum number of subscripts exceeded	-
C4358	Invalid array base	-
C4359	Invalid operand	-
C4609	Invalid binary constant	A constant value with 2# prefix must contain only binary digits (0 or 1).
C4610	Invalid octal constant	A constant value with 8# prefix must contain only octal digits (between 0 and 7).
C4611	Invalid hexadecimal constant	A constant value with 16# prefix must contain only hexadecimal digits (between 0 and 9 and between A and F).
C4612	Invalid decimal constant	A decimal constant must contain only digits between 0 and 9, a leading sign + or -, a decimal separator '.' Or a exponent indicator 'e' or 'E'.
C4613	Invalid time constant	A constant value with t# prefix must contain a time indication in decimal notation and a time unit between 'ms', 's' or 'm'.
C4614	Invalid constant string	-

Error code	Short description	Explanation
C4864	Duplicate function name	The indicated function name has already been used for another application object.
C4865	Invalid function type	The data type returned by the indicated function is not correct.
C5120	Duplicate program name	The indicated program name has already been used for another application object.
C5376	Duplicate function block name	The indicated function block name has already been used for another application object.
C5632	Invalid pragma	-
C5633	Invalid pragma value	-
C5889	Duplicate macro name	-
C5890	Duplicate macro parameter name	-
C6144	Invalid resource definition: two or more tasks have the same ID	-
C16385	Invalid init value	-
C16386	Invalid initialization definition	-
C16387	Invalid array delimiters (brackets)	-
C16388	Empty init value	-
C16389	Empty array init value	-
C16390	Invalid repeated init value	-
C16391	Not implemented	-
C16392	Missing array delimiters (brackets)	-
C16393	Missing comma	-
C16394	Not implemented	-
C16395	Invalid (incomplete) string	-
D12289	Cannot allocate database	The memory space needed for a parameter database exceeds the space available on the target system. If possible, remove unused parameter records, menus and so on.
D12290	Cannot allocate database record	The memory space needed for a parameter database exceeds the space available on the target system. If possible, remove unused parameter records, menus and so on.
D12291	Database variable not found	Internal compiler error.
D12292	Invalid expression or expression syntax error	The database expression that has the result indicated is not correct, contains syntax errors or invalid operators.

Error code	Short description	Explanation
D12293	Invalid parameter reference in expression	The database expression that has the result indicated contains a parameter (as operand) that is not the same to which the expression refers to. The expression can use only PLC variables (including the variables associated with parameters) and the value of the parameter that is exchanged at the moment. For example: $pDELTA = DELTA / pRATIO + pOFFSET$ is correct because the parameter exchanged is DELTA and it is the only parameter value used in the expression. The expression: $pDELTA = DELTA / pRATIO + OFFSET$ is not correct because the parameter OFFSET used in the expression is not exchanged.
D12294	Recursive expression	The database expression that has the result indicated calls itself by using some operand used that contains the current expression result.
D12295	Unresolved variable in expression	The database expression that has the result indicated uses an operand that is not defined in the whole PLC project.
D12296	Unresolved expression result	Internal compiler error.
D12297	Invalid result type for expression	The parameter that is the result of the expression has a data type invalid (such as enumerative) or not defined.
D12298	Invalid operand in expression	The database expression that has the result indicated uses an invalid operand.
D12299	Invalid variable type for expression	The variable that is the result of the expression has a data type invalid (such as enumerative) or not defined.
D12300	Assembler error	Internal compiler error.
D12301	Cannot allocate database code	The code space needed for the expression is exhausted. Is necessary to remove some expressions from the parameter's database.
D12302	Invalid operation in expression	The database expression that has the result indicated uses an invalid operand.
F1025	Invalid network	The indicated FBD or LD network contains a connection error (the errors are normally indicated by red connections).
F1026	Unconnected pin	The indicated block (operator, function, contact, or coil) has an unconnected pin.
F1027	Invalid connection (incomplete, more than a source and so on)	Internal compiler error.
F1028	More than one network per block	The network indicated contains more networks of blocks and variables not connected between them.
F1029	Ambiguous network evaluation	The compiler is not able to find a univocal way to establish the order of blocks execution.
F1030	Temporary variables allocation error	Internal compiler error.
F1031	Inconsistent network	The network indicated does not have input or output variables.

Error code	Short description	Explanation
F1032	Invalid object connected to power rail	-
F1033	Invalid use of pin negation (ADR operator does not allow negated input	-
F1034	Invalid use of pin negation (SIZEOF operator does not allow negated input	-
G0001	Invalid operand number	The number of operands is not correct for the operand or the function indicated.
G0002	Variable not defined	The variable has not been defined in the local or global context.
G0003	Label not defined	The label indicated for the JMP operand is not defined in the current POU (program, function, or function block).
G0004	Function block not defined	The indicated instance refers to a function block not defined in the whole project.
G0005	Reference to object not defined	The indicated instance refers to an object not defined in the whole project.
G0006	Constant not admitted	The operation indicated does not allow to use constants (typically store or assign operations).
G0007	Code buffer overflow	The total size of code used for POU (programs, functions, and function blocks) exceed the space available on the target system.
G0008	Invalid access to variable	The access made to the indicated variable is not allowed. An attempt to write a read-only variable or to read a write-only variable has been made.
G0009	Program not found	The indicated program does not exist in the current project.
G0010	Program already assigned to a task	The indicated program has been assigned to more than one task of the target system.
G0011	Cannot allocate code buffer	There is not enough memory on the PC to create the image of the code of the target system.
G0012	Function not defined	The indicated function does not exist in the current project.
G0013	Cyclic declaration of function blocks	The indicated function block call itself directly or by using other functions.
G0014	Incompatible external declaration	The external variable declaration of the current function block does not match with the global variable definition it refers to (the one with the same name). Typically is the case of a type mismatch.
G0015	Accumulator extension	-
G0016	External variable not found	The external variable does not refer to any of the global variables of the project (for example: there is not a global variable with the same name).
G0017	Program is not assigned to a task	The indicated program has not been assigned to a task in the target system.

Error code	Short description	Explanation
G0018	Task not found in resources	The indicated task is not defined in the target system.
G0019	No task defined for the application	There are not task definitions for the target system. The target definition file (*.TAR) is missing or incomplete. Contact the target system vendor.
G0020	Far data allowed only for load/store operations in PROGRAMs	Huge memory access is not allowed for function blocks, only for programs (error code valid only for some target system with NEAR/FAR data access).
G0021	Invalid processor type	The processor indicated into the target definition file (*.TAR) is not correct or is not supported by the compiler.
G0022	Function block with process image variables cannot be used in event tasks	-
G0023	Process image variables cannot be used in event tasks	-
G0024	Accumulator undefined	-
G0025	Invalid index	-
G0026	Only constant index allowed	-
G0027	Illegal reference to the address of a register	-
G0028	Less than 10% of free code	-
G0029	Index exceeds array size	-
G0030	Access to array as scalar - assuming index 0	-
G0031	Number of indexes not matching the var size	-
G0032	Multidimensional variables not supported	-
G0033	Invalid data type	-
G0034	Invalid operand type	-
G0035	Assembler error	-
G0036	Aborted by user	-
G0037	Element not defined	-
G0038	Cyclic declaration of structures	-
G0039	Cyclic declaration of typedefs	-
G0040	Unresolved definition of typedef	-
G0041	Exceeding dimensions in typedef	-
G0042	Unable to allocate compiler internal data	-

Error code	Short description	Explanation
G0043	CODE GENERATOR INTERNAL ERROR	-
G0044	Real data not supported	-
G0045	Long real data not supported	-
G0046	Long data not supported	-
G0047	Operation not implemented	-
G0048	Invalid operator	-
G0049	Invalid operator value	-
G0050	Unbalanced parentheses	-
G0051	Data conversion	-
G0052	To be implemented	-
G0053	Invalid index data type	-
G0054	Negation without condition	-
G0055	Operation not allowed on boolean	-
G0056	Negation of a non-boolean operand	-
G0057	Boolean operand required	-
G0058	Floating point parameter not allowed	-
G0059	Operand extension	-
G0060	Division by zero	-
G0061	Illegal comparison	-
G0062	Function block must be instantiated	-
G0063	String operand not allowed	-
G0064	Operation not allowed on pointers	-
G0065	Destination may be too small to store current result	-
G0066	Cannot use a function block containing external variables with process image in more than one task	-
G0067	Cannot load the address of an explicit constant	-
G0068	Writing a real value into an integer variable	-
G0069	Cannot use complex variables in functions. Not implemented	-
G0070	Signed/unsigned mismatch	-
G0071	Conversion data types mismatch, possible loss of data	-

Error code	Short description	Explanation
G0072	Implicit type conversion of boolean to integer	-
G0073	Implicit type conversion of boolean to real	-
G0074	Implicit type conversion of integer to boolean	-
G0075	Implicit type conversion of integer to boolean	-
G0076	Implicit type conversion of real to boolean	-
G0077	Implicit type conversion of real to integer	-
G0078	Arithmetic operations require numerical operands	-
G0079	Bitwise logical operations require bitstring/integer operands	-
G0080	Comparison operations require elementary (that is, not user-defined) operands	-
G0081	Cannot take the address of a bit variable	-
G0082	Writing a signed value into an unsigned variable	-
G0083	Writing an unsigned value into a signed variable	-
G0084	Implicit conversion from single to double precision	-
G0085	Implicit conversion from double to single precision	-
G0086	Function parameter extension	-
G0087	Casting to the same type has no effects	-
G0088	Function parameters wrong number	-
G0089	Embedded target function not found	-
G0090	Recursive type declaration	-
G0091	Wrong initial value. Signed/unsigned mismatch	-
G0092	Wrong initial value. Conversion data types mismatch, possible loss of data	-
G0093	String will be truncated	-

Error code	Short description	Explanation
G0094	Init value type mismatch	-
G0095	Improper init value	-
G0096	Init value object not found	-
G0097	Invalid assignment to pointer	-
G0513	Invalid operator	The operator indicated is not allowed for the indicated operation.
G0514	Operation not implemented	The operator indicated is not supported by the target system.
G0515	Real data not supported	The target system in use does not support floating point operations.
G0516	Destination may be too small to store current result	The variable destination of the store/assignment operation has a data type smaller than the one of the accumulators. Data may be lost in the operation. For example, if the accumulator contains 340 and the destination operand is of SINT type, the assignment operation will lose data. If the operation is under the programmer's control an appropriate type conversion function (TO_SINT, TO_INT, TO_DINT and so on) can be used to eliminate the advisory message.
G0517	Long data not supported	The target system in use does not support long data operations.
G0518	Accumulator extension	The variable destination of the store/assignment operation has a data type bigger than the one of the accumulators. An extension operation has been performed automatically by the compiler. To eliminate this advisory message, use the appropriate type conversion function (TO_SINT, TO_INT, TO_DINT and so on).
G0519	Assembler error	Internal compiler error.
G0520	Negation allowed only on boolean	The 'N' modifier used for some IL operators (LDN, STN, ANDN and so on) cannot be used with operators having type other than boolean.
G0521	Operation allowed with boolean types	The IL operator indicated (typically 'S' or 'R') cannot be used when the accumulator has a type other than BOOL.
G0522	Instruction has constant result	The indicated operation has a result that is constant (for example multiply by 0, AND with FALSE).
G0523	Instruction is a NOP	The operation indicated has no influence on the value of the accumulator (for example multiply by 1, AND with TRUE).
G0524	Unbalanced parentheses	The number of opened parentheses does not match with the number of the closed parentheses in the indicated code block.
G0525	Operation not allowed on boolean	The indicated operation cannot be performed on boolean operands (for example the arithmetic operations).
G0526	Cannot perform modulo with long values	The current target system does not allow the modulo operation with long data types.

Error code	Short description	Explanation
G0527	Division by 0	The indicated division operation has the constant value 0 as denominator.
G0528	Negation without condition	The indicated operation (JMP or RET) has the negation modifier 'N' without the conditional evaluation modifier 'C'. Use JMPCN instead of JMPN or RETCN instead of RETN.
G0529	Initial value not defined	Internal compiler error.
G0530	Invalid initial value	The initial value of the variable is not indicated correctly.
G0531	Invalid accumulator type	The accumulator has a data type not allowed for the indicated operation (for example MUX operator with REAL accumulator).
G0532	Code generator internal error	Internal compiler error.
G0533	Invalid operator value	The operator has a value not acceptable for the indicated operation (for example SHL with constant value bigger than 32).
G0534	Accumulator undefined	The operation is performed without a previously loaded value into the accumulator.
G0535	Invalid index	The constant index value used in the indicated expression is too big for the array dimension. See the array declaration string.
G0536	Only constant index allowed	The use of variable as index for the indicated array is not supported by the compiler. This error is typically issued with boolean (bit) arrays.
G0537	Indexing of boolean constants not allowed	The use of variable as index for the indicated array is not supported by the compiler. This error is typically issued with boolean (bit) arrays.
G0538	Return not allowed from programs	The RET operator is not allowed in PROGRAM blocks.
G0539	Function block must be instantiated	A function block cannot be invoked directly with a CAL instruction. It must be instantiated before its use for example, must be a variable with data type corresponding to the function block instead.
G0540	Operation not allowed with real types	The indicated operation cannot be executed on REAL data types. Instructions of this kind are logical and bitwise operations.
G0541	Accumulator conversion	This advisory message informs that the data type of the accumulator has been automatically converted by the compiler. This operation is typically executed when the accumulator and the operand used in an arithmetic operation have different data types.
G0542	Real accumulator must be reloaded	Some target-specific implementations with software floating point emulation require that each store operation shall be preceded by a new load operation or an arithmetic sequence.

Error code	Short description	Explanation
G0543	Real accumulator not stored	Some target-specific implementations with software floating point emulation require that when the floating point stack has been loaded, the same shall be unloaded at the end of arithmetic sequence.
G0544	Long real data not supported	The long real data type LREAL is not supported by the compiler.
G0769	Invalid operator	The operator indicated is not allowed for the indicated operation.
G0770	Operation not implemented	The operator indicated is not supported by the current target system.
G0771	Assembler error	Internal compiler error.
G0772	Long real data not supported	The long real data type LREAL is not supported by the compiler.
G0773	Long data not supported	The long data type LINT is not supported by the compiler.
G0774	Negation of a non-boolean parameter	The negation modifier 'N' cannot be used in operations with data types different than boolean.
G0775	Operation not allowed on boolean	The indicated operation cannot be performed on boolean operands (for example the arithmetic operations).
G0776	Accumulator extension	The variable destination of the store/assignment operation has a data type bigger than the one of the accumulators. An extension operation has been performed automatically by the compiler. To eliminate this advisory message, use the appropriate type conversion function (TO_SINT, TO_INT, TO_DINT and so on).
G0777	Accumulator undefined	The operation is performed without a previously loaded value into the accumulator.
G0778	Destination may be too small to store current result	The variable destination of the store/assignment operation has a data type smaller than the one of the accumulators. Data may be lost in the operation. For example, if the accumulator contains 340 and the destination operand is of SINT type, the assignment operation will lose data. If the operation is under the programmer's control an appropriate type conversion function (TO_SINT, TO_INT, TO_DINT and so on) can be used to eliminate the advisory message.
G0779	Division by zero	The indicated division operation has the constant value 0 as denominator.
G0780	Operation allowed on real parameters only	The indicated operation cannot be executed on REAL data types. Instructions of this kind are logical and bitwise operations.
G0781	Illegal comparison	The indicated comparison operation is executed between non-homogeneous data types.
G0782	Negation without condition	The indicated operation (JMP or RET) has the negation modifier 'N' without the conditional evaluation modifier 'C'. Use JMPCN instead of JMPN or RETCN instead of RETN.

Error code	Short description	Explanation
G0783	Boolean parameter required	The IL operator indicated (typically 'S' or 'R') cannot be used when the accumulator has a type other than BOOL.
G0784	Operand extension	The data type of the operand has been extended to the data type of the accumulator. Then the operation is executed. The operand extension take place whenever the operand data type is smaller than the accumulator data type.
G0785	Does not support float accumulator	The accumulator has REAL data type and it is not allowed for the indicated operation (typically MUX operation).
G0786	Does not support boolean accumulator	The accumulator has boolean data type and is not allowed for the indicated operation (for example MUX operator).
G0787	Comparison of unsigned type and signed type	The compare operation indicated is performed using operators that have signed and unsigned data type. Undesired or uncontrolled result may be possible.
G0788	Illegal conversion	Internal compiler error.
G0789	Conversion may result in loss or corruption of data	Error code not used.
G0790	Illegal negation of a real parameter	Error code not used.
G0791	Writing a real value into an integer var / param	The parameter passed to the function is of REAL type instead of an integer data type as required by the function input variables definition.
G0792	Writing an integer value into a real var / param	The parameter passed to the function is of an integer data type instead of the REAL type as required by the function input variables definition.
G0793	Writing a signed value into an unsigned var / param	The assignment operation is performed on an unsigned data type variable but the accumulator data type has a signed data type. Undesired result may be possible.
G0794	Writing an unsigned value into a signed var / param	The assignment operation is performed on an unsigned data type variable but the accumulator data type has a signed data type. Undesired result may be possible.
G0795	Unbalanced parentheses	The number of opened parentheses does not match with the number of the closed parentheses in the indicated code block.
G0796	Error while extending parameters	Internal compiler error.
G0797	Invalid index	The constant index value used in the indicated expression is too big for the array dimension. See the array declaration string.
G0798	Using a boolean index to access an element of array	The indicated array access is incorrect because the index variable used has a boolean data type.
G0799	Return not allowed from programs	The RET operator is not allowed in PROGRAM blocks.
G0800	Boolean accumulator required	The indicated SEL operator requires that the accumulator has the boolean data type.
G0801	Operators have mismatching type	The selection performed by MUX and SEL operators shall be done between elements that have homogeneous data types.

Error code	Short description	Explanation
G0802	Function block must be instantiated	A function block cannot be invoked directly with a CAL instruction. It must be instantiated before its use for example, must be a variable with data type corresponding to the function block instead.
G1537	Using a boolean index to access an element of array	-
G1538	Does not support boolean accumulator	-
G1539	Does not support float accumulator	-
G1540	Error while extending operand(s)	-
G1541	Writing a signed value into an unsigned variable	-
G1542	Writing an unsigned value into a signed variable	-
G1543	Writing a real value into an integer variable	-
G1544	Writing an integer value into a real variable	-
G1545	Converting a string into a number	-
G1546	Converting a number into a string	-
G1547	FPU stack full	-
G1548	FPU stack empty	-
G1549	FPU stack size error	-
G1550	Illegal access to variable through function	-
G1551	Illegal reference to address of variable accessible through function	-
G1552	Invalid access through function	-
G1553	Two variables with the same handle	-
G1554	Invalid index for variable accessible through function	-
G1555	Invalid instruction with non-empty FPU stack	-
G1556	Function result of type string requires store to variable	-
G8193	Type definition of unknown data type	-
G8194	Type definition has exceeding array dimensions	-
G8195	Cyclic definition of data type	-

Error code	Short description	Explanation
G8196	Double pointers are not supported	-
G8197	No enumerative elements	-
G8199	Invalid or undefined initialization constant	-
G10241	Too many initializers for variable	-
G10242	Too less initializers for variable	-
G10243	Constant without init values	-
P2048	Cannot open parameters file	The source file for parameters (with PPC extension) cannot be opened because of is missing or is locked by the PC's file system.
P2049	Symbol table file not created	The symbol allocation file (with SYM extension) cannot be written because of disk write protection or insufficient disk space.
P2050	Cannot create parameters file	The parameters file (with PAR extension) cannot be written because of disk write protection or insufficient disk space.
P2051	Cannot create directory	The directory for the new project cannot be created. The problem arises when there is a disk write protection or when the new directory indicated for the project is more than one level deep from an existing disk directory. The compiler creates only one new directory level (the one with the name of the project) starting from an existing directory.
P2052	Cannot open source project	The source project indicated for creating the new project does not exist, is incomplete, or is locked by the file system.
P2053	Save project error	The new project cannot be saved due to disk write protection, non-existing destination directory, or file system lock.
P2054	Generic file error	A non-specific error occurred during file operations.
P2055	Cannot copy file	The indicated file cannot be copied because of missing source file, disk write protection or destination file existing and protected.
P2056	Cannot save file	The indicated file cannot be saved because of disk write protection or destination file existing and protected.
P2057	Object already exist in project	The indicated object (variable, function, function block, or program) is contained in the last loaded library but there is already another object with the same name in the current project.
P2058	Cannot open library file	The indicated library file does not exists or cannot be opened due to file system locking.
P2059	Listing file not created	-
P2060	Cannot create PROGRAMMING binary file	-

Error code	Short description	Explanation
P2061	Cannot open template project	-
P2062	Support for processor is not available	-
P2063	Less than 10% of free code	-
P2064	Less than 10% of free data	-
P2065	Less than 10% of free retain data	-
P2066	Less than 10% of free bit data	-
P2067	Task not found in resources	-
P2068	No task defined for the application	-
P2069	Project is in the old PPJ format. It will be saved in the current PPJX format	-
P2070	Cannot open auxiliary source file	-
P2071	Cannot read file	-
P2072	Application name is longer than 10 characters: only the first 10 characters will be downloaded into the target	-
P2073	Downloadable source code file is not password-protected	-
P2074	Downloadable PLC application binary file not created	-
P2075	Less than 10% of free ext/aux data	-
P2076	Project private copy of this library was missing and has been replaced with a new copy of the library (from the original path)	-
P2077	Cannot load library! Project private copy of this library was missing and the original path to the library is invalid: library has been dropped	-
P2078	PLC variables export file not created	-
P2079	Debug symbols package (for following download to the target device) not created	-
P2080	Source code package (for following download to the target device) not created	-
P2081	Invalid task definition	-
P2083	Invalid or incoherent task period	-
P2084	Broken library link	-
S1281	Generic ST error	-

Error code	Short description	Explanation
S1282	Too many expressions nested	-
S1283	No iteration to exit from	-
S1284	Missing END_IF	-
S1285	Invalid ST statement	-
S1286	Invalid assignment	-
S1287	Missing;	-
S1288	Invalid expression	-
S1289	Invalid expression or missing DO	-
S1290	Missing END_WHILE	-
S1291	Missing END_FOR	-
S1292	Missing END_REPEAT	-
S1293	Invalid expression or missing THEN	-
S1294	Invalid expression or missing TO	-
S1295	Invalid expression or missing BY	-
S1296	Invalid statement or missing UNTIL	-
S1297	Invalid assignment, := expected	-
S1298	Invalid address expression	-
S1299	Invalid size expression	-
S1300	Function return value ignored	-
S1301	Invalid parameter passing	-
S1302	Function parameter not defined	-
S1303	Useless expression	-
S1304	Unbalanced parentheses	-
S1305	Unknown function	-
S1306	Invalid function parameter(s) specification	-
S1307	Function parameter does not exist	-
S1308	Multiple assignment not allowed (in accordance with IEC 61131-3)	-
S1309	ST preprocessor buffer overflow	-
S1310	Function block invocation of a non-function block instance	-
S1311	Missing END_WAITING	-
S1312	Syntax error	-
S1537	Generic SFC error	-

Error code	Short description	Explanation
S1538	Initial step missing	-
S1539	Output connection missing	-
S1540	The output pin must be connected to a transition	-
S1541	Every output pin of a transition must be connected to a step/jump block	-
S1542	Transition expected	-
S1543	Step or jump expected	-
S1544	Could not find the associate program code	-
S1545	Could not find the condition code	-
S1546	Unknown-type transition	-
S1547	Invalid destination	-
S1548	Duplicates action. Same SFC action cannot be used in more than one step	-
T8193	Communication timeout	The communication with the target system was unsuccessful because there is no answer from the system itself. More common causes of this problem are incorrect cable connection, invalid target address in communication settings, invalid settings of communication parameters (such as baud rate), or inoperable target system.
T8194	Incompatible target version	Error code not used.
T8195	Invalid code file	The target system image file (with IMG extension) is invalid or corrupted. Try to upload and create new version of the image file using the "Communication Upload image file" menu option.
T8196	Invalid data block index	The image file (with IMG extension) contains a data block that has an index greater than the largest index supported by the target system. Try to upload and create new version of the image file using the "Communication Upload image file" menu option. If the Any numerical type problem persist, contact the target system vendor.
T8197	Invalid target information address	Internal compiler error.
T8198	Flash erase failure	The target system was not able to complete the flash erasure procedure. Contact the target system vendor for details.
T8199	Code write failure	The target system was not able to complete the flash programming procedure. Contact the target system vendor for details.

Error code	Short description	Explanation
T8200	Communication device unavailable	The compiler tried to communicate with the target system but the communication channel is not available. If the problem persists and there are other applications that communicate with the target system, deactivate the communication on the other applications and try again.
T8201	Invalid function index	Internal compiler error.
T8202	Invalid database information address	The address of the parameter's database memory area of the target system is not correct or valid. Try to upload and create new version of the image file using the "Communication Upload image file" menu option.
T8203	Invalid target information	-
T8204	Rebuild required	-
T8205	Invalid task	-
T8206	Application-level communication protocol error: PLC run-time was not able to understand the received command	-
T8209	No room for source file on the target	-
T8210	Error while uploading source code from target device	-
T8211	No room for debug symbols on the target	-
T8212	Memory read error	-
T8213	Memory write error	-
T8214	Not enough space available on the target device for the PLC application binary	-



!

%

According to the IEC standard, % is a prefix that identifies internal memory addresses in the logic controller to store the value of program variables, constants, I/O, and so on.

%I

According to the IEC standard, %I represents an input bit (for example, a language object of type digital IN).

%IW

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

%MW

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

%Q

According to the IEC standard, %Q represents an output bit (for example, a language object of type digital OUT).

A

analog input

Converts received voltage or current levels into numerical values. You can store and process these values within the logic controller.

analog output

Converts numerical values within the logic controller and sends out proportional voltage or current levels.

application

A program including configuration data, symbols, and documentation.

ARRAY

The systematic arrangement of data objects of a single type in the form of a table defined in logic controller memory. The syntax is as follows: `ARRAY [<dimension>] OF <Type>`

Example 1: `ARRAY [1..2] OF BOOL` is a 1-dimensional table with 2 elements of type `BOOL`.

Example 2: `ARRAY [1..10, 1..20] OF INT` is a 2-dimensional table with 10 x 20 elements of type `INT`.

ASCII

(*American standard code for Information Interchange*) A protocol for representing alphanumeric characters (letters, numbers, certain graphics, and control characters).

assigned variable

A variable is assigned if its location in the logic controller memory is known.

For example, the `Water_pressure` variable is said to be assigned through its association with memory location `%MW102`.

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.

byte

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

C

conditional element

Allows to implement conditions in the program in offline mode.

configuration

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

controller

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

D

digital I/O

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

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.

EEPROM

(*electrically erasable programmable read-only memory*) A type of non-volatile memory to store required data even when power is removed.

Ethernet

A physical and data link layer technology for LANs, also known as IEEE 802.3.

expansion I/O module

(*expansion input/output module*) Either a digital or analog module that adds additional I/O to the base controller.

F**FB**

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

FBD

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

function

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

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

function block

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

function block diagram

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

H**hex**

(*hexadecimal*)

I**I/O**

(*input/output*)

IEC 61131-3

Part 3 of a 3-part IEC standard for industrial automation equipment. IEC 61131-3 is concerned with controller programming languages and defines 2 graphical and 2 textual programming language standards. The graphical programming languages are ladder diagram and function block diagram. The textual programming languages include structured text and instruction list.

IL

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

instruction list language

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

IP

(*Internet protocol*) Part of the TCP/IP protocol family that tracks the Internet addresses of devices, routes outgoing messages, and recognizes incoming messages.

L**ladder diagram language**

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

LED

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

M

machine

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

master/slave

The single direction of control in a network that implements the master/slave mode.

Modbus

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

N

node

An addressable device on a communication network.

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.

PLC

(*programmable logic controller*) An industrial computer used to automate manufacturing, industrial, and other electromechanical processes. PLCs are different from common computers in that they are designed to have multiple input and output arrays and adhere to more robust specifications for shock, vibration, temperature, and electrical interference among other things.

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.

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

RJ45

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

RPDO

(*receive process data object*) An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

RS-232

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

RS-485

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

RTU

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

S

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.

symbol

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

T

TCP

(*transmission control protocol*) A connection-based transport layer protocol that provides a simultaneous bi-directional transmission of data. TCP is part of the TCP/IP protocol suite.

terminal block

(*terminal block*) The component that mounts in an electronic module and provides electrical connections between the controller and the field devices.

TPDO

(*transmit process data object*) An unconfirmed broadcast message or sent from a producer device to a consumer device in a CAN-based network. The transmit PDO from the producer device has a specific identifier that corresponds to the receive PDO of the consumer devices.

U**UDINT**

(*unsigned double integer*) Encoded in 32 bits.

UINT

(*unsigned integer*) Encoded in 16 bits.

user defined function

It allows you to create your own functions with one or more input parameters, local variables, and a return value. The user-defined function can then be called in operation blocks. A user-defined function is stored as part of the project and downloaded to the logic controller as part of the application.

W**WORD**

A type encoded in a 16-bit format.



A

- Action
 - assign to a step, *256*
- Application
 - definition of, *33*

B

- Block
 - information, *235, 246*
 - properties, *235, 245*
- Bookmark, *253*
 - IL, *229*
- Branches
 - insert, *250*

C

- Coil
 - insert, *242*
- Comments
 - insert, *249*
- Contact
 - insert, *240*
- Custom workspace
 - basic unit, *221*
 - elements, *223*
 - enable, *221*
 - operation, *222*

D

- Developing programs, stages of, *34*

E

- Edit
 - enumeration, *208*
 - function, *228, 252*
 - network, *235, 244, 261*
 - ST, *252*
 - structure, *206*
 - subrange, *210*
 - typedef, *205*
- Editor
 - Adding a Variable, *306, 307, 308*
 - FBD, *230*
 - global variable, *194*
 - Graphic, *47*
 - IL, *227*
 - LD, *236*
 - PLC, *226*
 - POU, *186*
 - SFC, *254*
 - ST, *251*
 - Text, *47*
 - variable, *263*
- Enumerator
 - create, *207*
 - delete, *208*
 - edit, *208*
- Expression
 - insert, *248*

F

- FBD
 - create, *231*
 - editor, *230*
- Function, *382, 386, 392, 394, 398, 399, 401, 407, 418, 423*
 - edit, *228, 252*

G

Global variable
edit, *194*

I

IL
bookmark, *229*
editor, *227*

J

Jump, *261*

L

LD
Coil / Contact Properties, *244*
create, *237*
editor, *236*
insert coil, *242*
insert contact, *240*

Library
export, *177*
import from, *178*
include, *175*
remove, *176*
undo import from, *179*
update, *180*

M

Menu
debug, *152*
developer, *77*
edit, *148*
file, *75, 147*
help, *78, 154*
on-line, *77, 151*
prg, *93*
project, *77, 150*
resources, visibility of, *94*
set, *93*
target, *95*
tools, *154*
variables, *153*
view, *76, 149*
window, *45, 154*

Minimum system requirements, *25*

N

Network
add / remove, *231, 238*
connect block, *233*
edit, *235, 244, 261*
insert block, *233, 243*
label, *232, 239*

Non-program data, *33*

O

Operating modes
offline, *39*
online, *39*
simulation, *39*

Operator, *386, 391, 393, 400, 418, 420, 420, 421*

P

POU
editor, *186*

Program

- associate to a task, *200*
- compiling, *155*
- definition of, *33*
- manage into task, *200*

Program development, stages of, *34*

Project

- close, *59*
- creating, *33*
- custom workspace, *221*
- definition of, *33*
- distribute, *60*
- download, *65*
- edit, *59*
- find in, *217*
- menu, *77*
- new, *55*
- open, *59*
- print, *56*
- save, *57*
- save as, *57*
- toolbar, *36, 156*
- window, content of, *182*

R

Registering EcoStruxure Machine Expert - HVAC software, *26*

S

SFC

- connect elements, *255*
- create, *254*
- editor, *254*
- insert element, *255*

ST

- create, *252*
- editor, *251*

Structure

- create, *206*
- delete, *207*
- edit, *206*

Subrange

- create, *209*
- delete, *210*
- edit, *210*

System requirements, *25*

T

Task

- assign a program at creation time, *184*
- associate a program, *200*
- configuration, *201*
- managing program, *200*
- Overview, *199*

Tool window, *38*

- management, *44*

Toolbar, *79, 327, 351*

- buttons, *155*
- configuration, *79*
- debug, *157*
- FBD, *158*
- LD, *160*
- main, *155*
- network, *161*
- project, *36, 156*
- SFC, *159*

Transition

- condition of, *258*
- conditional code, *259*

Trigger window, *326, 327*

- graphic, *347, 348, 350, 351, 353, 354, 354, 355, 356, 359, 361, 363, 364, 365, 367, 369*
- text, *328, 330, 331, 333, 334, 335, 336, 338, 340, 341, 343, 343, 345*

Typedef

- create, *203*
- delete, *205*
- edit, *205*

V

Variable

- copy, *267*
- create, *264*
- delete, *267*
- Edit, *264*
- editor, *263*
- global, *188*
- insert, *247*
- local, *195*
- Sampling, *309*
- sort, *267*

W

Watch list, *301*

- autosave, *302*

Watch window

- add item, *296*
- data format, *300*
- refresh, *299*
- remove item, *299*
- watch list, *301*

Workspace

- custom, *221*
- migration, *221*